ProT-GFDM: A Generative Fractional Diffusion Model for Protein Generation

Xiao Liang[1], Wentao Ma[2], Eric Paquet[3,4], Herna Lydia Viktor[4], and Wojtek Michalowski[1]

[1]Telfer School of Management, University of Ottawa, Ottawa, ON, K1N 6N5, Canada
[2]Department of Computer Science, University of Toronto, Toronto, ON, M5S 2E4, Canada
[3]National Research Council, 1200 Montreal Road, Ottawa, ON, K1A 0R6, Canada
[4]School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, ON, K1N 6N5, Canada

May 1, 2025

Abstract

This work introduces the generative fractional diffusion model for protein generation (ProT-GFDM), a novel generative framework that employs fractional stochastic dynamics for protein backbone structure modeling. This approach builds on the continuous-time score-based generative diffusion modeling paradigm, where data are progressively transformed into noise via a stochastic differential equation and reversed to generate structured samples. Unlike classical methods that rely on standard Brownian motion, ProT-GFDM employs a fractional stochastic process with superdiffusive properties to improve the capture of long-range dependencies in protein structures. Trained on protein fragments from the Protein Data Bank, ProT-GFDM outperforms conventional score-based models, achieving a 7.19% increase in density, a 5.66% improvement in coverage, and a 1.01% reduction in the Fréchet inception distance. By integrating fractional dynamics with computationally efficient sampling, the proposed framework advances generative modeling for structured biological data, with implications for protein design and computational drug discovery.

Keywords: Generative diffusion model, protein generation, fractional Brownian motion, stochastic differential equation, ordinary differential equation.

1 Introduction

Proteins are fundamental macromolecules for diverse biological functions. Their unique amino acid sequences have three-dimensional (3D) structures, which determine cellular roles, specificity, and stability [1, 2]. *De novo* [3] protein generation aims to produce new protein sequences that fold into stable and functional structures, creating novel

biomolecules for applications in medicine, biotechnology, and synthetic biology. Traditionally, protein design follows a two-step process: determining a backbone structure that meets the desired structural and biochemical properties and designing an amino acid sequence to fit that backbone. Often reliant on template-based fragment sampling and expert-defined topologies, this approach has limitations, including that the backbone may not be optimally designable or the sequence may fail to conform to the intended structure. These problems can lead to a relatively low success rate in computational protein design [5]. During the past decades, the exponential growth of protein sequence data has driven the development of myriad computational methods for de novo protein design [4] and protein function prediction [5]. A systematic review and comparison of methods is provided in [6].

Recent progress in artificial intelligence (AI) models has substantially changed the field of protein design. Traditionally, generative models, such as generative adversarial networks (GANs) [7], variational autoencoders (VAEs) [8], and flow-based models [9] have been widely applied to protein generation, enabling the design of novel proteins with the desired properties and facilitating the exploration of vast combinatorial spaces. For example, ProteinVAE [11] applies ProtBERT [12] to convert raw protein sequences into latent representations, using an encoder–decoder framework enhanced with positionwise multihead self-attention to capture long-range sequence dependencies. In contrast, ProT-VAE [13] employs a different pretrained language model, ProtT5NV, and incorporates an internal, family-specific encoder–decoder layer to learn parameters tailored to individual protein families. Conversely, ProteinGAN [14] employs a GAN architecture to generate protein sequences, and its capabilities are illustrated through the example of malate dehydrogenase, demonstrating its potential to produce fully functional enzymes.

However, Strokach et al. listed the disadvantages of generative models for protein design [15]. For instance, GANs and VAEs each have their drawbacks. For example, GANs can experience unstable training processes compared to other model types and often generate examples with low diversity due to mode collapse. Additionally, GANs lack mechanisms for mapping existing data to latent spaces or for calculating log-likelihoods. In contrast, VAEs are typically less effective at modeling data with a fixed dimensionality and often generate lower-resolution samples than GANs. Other deep learning approaches have also played a significant role in advancing protein design (e.g., [16, 17, 18, 19, 20]).

Probabilistic diffusion models have recently demonstrated significant potential in bioinformatics, particularly in such applications as protein generation. These models learn to approximate complex data distributions by sequentially corrupting and denoising data samples, enabling the synthesis of high-quality, realistic output [21]. **Denoising diffusion probabilistic models** (DDPMs) [22] define a discrete-time Markov chain that progressively adds Gaussian noise to data, transforming data into a nearly isotropic Gaussian distribution. Formally, for each training sample $\mathbf{x}_0 \sim p_{\text{data}}(\mathbf{x})$, the forward process follows a predefined variance schedule $0 < \beta_1 < \beta_2, \ldots, \beta_T < 1$ and constructs a sequence $\{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_T\}$, where the transition at each step is given by the following:

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}).$$

Expanding this recurrence yields direct mapping from $\mathbf{x}_0$ to $\mathbf{x}_t$:

$$\mathbf{x}_t = \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\boldsymbol{\varepsilon}_{t-1}, \quad t = 1, \ldots, T, \tag{1}$$

where $\boldsymbol{\varepsilon}_t \sim \mathcal{N}(0, \mathbf{I})$. Applying the **Markov property**, we can marginalize over all intermediate steps and derive the closed-form distribution of $\mathbf{x}_t$, given the original data sample:

$$p(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\alpha_t}\mathbf{x}_0, (1-\alpha_t)\mathbf{I}),$$

where $\alpha_t = \prod_{s=1}^{t}(1-\beta_s)$. The noise schedule is predefined such that $\alpha_T \to 0$, ensuring that $\mathbf{x}_T$ asymptotically approaches a standard Gaussian distribution (i.e., $p(\mathbf{x}_T) \approx \mathcal{N}(0, \mathbf{I})$). The **reverse process**, which transforms Gaussian noise back into structured data, is modeled as another Markov chain, parameterized as follows:

$$p(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}(t)),$$

where the mean function $\mu_\theta(\mathbf{x}_t, t)$ is learned using a neural network $\mathrm{NN}_\theta(\mathbf{x}_t, t)$. Once the model is trained, new samples can be generated by first drawing $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$, and iteratively updating:

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t + \beta_t\, \mathrm{NN}_{\theta^*}(\mathbf{x}_t, t)\right) + \sqrt{\beta_t}\mathbf{z}_t, \quad t = T, T-1, \ldots, 1,$$

where θ^* denotes the optimized parameters of the trained network NN_θ. The DDPM effectively recovers structured data from noise via this iterative denoising process. An alternative approach to generative modeling is **score-matching Langevin dynamics (SMLD)** [23], a method initially developed in physics to describe the stochastic motion of particles under deterministic and random forces. Langevin dynamics facilitates sampling from a target distribution $p(\mathbf{x})$ using an iterative update rule:

$$\mathbf{x}_t = \mathbf{x}_{t-1} + \tau\nabla_{\mathbf{x}} \log p(\mathbf{x}_{t-1}) + \sqrt{2\tau}\mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(0, \mathbf{I}), \tag{2}$$

where τ denotes the step size, and $\mathbf{x}_0$ is initialized from white noise. The term $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ represents the **score function**, also known as **Stein's score function**, which determines the optimal update direction for the sample. The score function $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ must be learned using neural networks to apply Langevin dynamics for generative modeling. Once trained, samples can be generated by iteratively refining noisy input via Langevin dynamics, guiding them toward the target data distribution. Yang et al. referred to these two model classes, DDPMs and SMLDs, together as **score-based generative models**. Both DDPMs and SMLDs can be expanded to scenarios encompassing an infinite number of time steps or noise levels, where perturbation and denoising procedures are characterized as solutions to stochastic differential equations (SDEs). This extended framework is referred to as a score-based SDE model (**ScoreSDE**) [21]. By interpreting the forward diffusion processes (1) and (2) as the discretization of an underlying SDE, we express the continuous-time limit as follows:

$$d\mathbf{x}_t = f(\mathbf{x}_t, t)dt + g(t)d\mathbf{w}_t, \tag{3}$$

where $f(\mathbf{x}, t)$ represents the **drift term**, governing the deterministic evolution of the process, $g(t)$ denotes the **diffusion coefficient**, often defined in terms of the noise schedule, and $d\mathbf{w}_t$ indicates a **Wiener process**, modeling stochastic perturbations. This generative framework generalizes previous score-based generative models by incorporating continuous-time SDEs and provides a mathematically flexible method to describe diffusion processes and generative modeling in continuous time.

In [24, 28], the authors explored the limitations and shortcomings of the classical ScoreSDE models equipped with Brownian motion (BM). When training data have unequal representation across modes, traditional diffusion models may fail to generate samples that properly represent all modes, focusing instead on dominant modes. In addition, generated samples might lack variety, producing outputs that are too similar or fail to explore the data distribution fully. These limitations might arise due to the nature of BM with a light-tailed distribution and independent increments. Moreover, BM relies on Gaussian noise; hence, it may struggle to model data distributions with heavy tails effectively (e.g., outliers or rare, extreme variations), whereas the independence assumption limits the ability to encode dependencies or correlations across time steps, which may be necessary to model complex, multimodal distributions accurately. To overcome this limitation, researchers have investigated Lévy processes and fractional BM (fBm) as potential alternatives to BM.

Yoon et al. [24] extended classical ScoreSDE models by replacing the underlying BM with a Lévy process, a stochastic process characterized by independent and stationary increments (i.e., the differential $d\mathbf{w}_t$ in Equation (3) is replaced by dL_t^α, where L_t^α is an α-stable Lévy process). Stable processes necessarily have a stability index of α, which lies in the range $(0, 2]$. When $\alpha = 2$, the process L_t^α reduces to the classical BM and, therefore, necessarily has continuous paths. These models enable more efficient exploration of the conformational space and are applied to protein generation. Motivated by [24], the authors of [25] presented innovative Lévy–Itō diffusion models that integrate Markovian and non-Markovian dynamics, incorporating fractional SDEs and Lévy distributions, with applications in protein generation.

An alternative for the driving process is fBm, a generalization of the standard BM that introduces long-range dependencies and controlled roughness. The authors of [26] presented the first continuous-time score-based generative model that employs fractional diffusion processes to govern its dynamics (i.e., the differential $d\mathbf{w}_t$ in Eq. (3) is replaced by dB_t^H, where B_t^H is a fBm, which features correlated increments and is characterized by the Hurst index $H \in (0, 1)$ with $H = 1/2$ corresponding to classical BM). Its precise definition is given in Definition 4.1. Moreover, BM and Lévy processes are semimartingales, meaning they can be decomposed into a sum of a local martingale and a finite variation process, enabling Itō calculus for stochastic integration [27]. A challenge in dealing with fBm is the nonsemimartingale nature, invalidating the use of classical Itō integrals. To maintain tractable inference and learning, the authors used a recently popularized Markov approximation of fBm (MA-fBm). They derived its reverse-time model, leading to the development of generative fractional diffusion models (GFDMs). In particular, SDEs driven by fBm are well-suited for capturing systems with temporal dependencies because they account for memory effects and correlations over time. Additional studies focusing on

fBm are found in [27, 28].

This work proposes a novel fractional diffusion-based generative model for protein design, ProT-GFDM, employing the mathematical framework of MA-fBm. Unlike traditional diffusion models that rely on BM, the proposed approach incorporates long-range dependencies and super-diffusive behavior, enabling more expressive and controllable generative dynamics. Compared to prior diffusion models [21], the proposed method achieves superior sample diversity and fidelity. This work also explores the effect of various noise schedules, including an alternative cosine noise schedule [29] that provides a distinct approach to noise control and influences generative behavior. Furthermore, this work systematically evaluates the adaptability of the proposed model to stochastic and deterministic solvers, demonstrating that fractional diffusion models can be effectively integrated with SDE solvers and ordinary differential equation (ODE) solvers, expanding their applicability in generative modeling. These contributions establish the approach as a flexible and robust framework for advancing generative protein design.

The paper is organized as follows. Section 2 discusses the background of ScoreSDE, laying the theoretical foundation and presenting the relevant prior work. Next, Section 3 explores protein-structure image generation, where the protein backbone is represented by an α-carbon distance map. Section 4 focuses on the core contribution of this work and is subdivided into parts that explore the fractional driving noise and its Markov approximation in subsection 4.1, generative fractional diffusion models in 4.2, and augmented score-matching techniques in 4.3. This section also includes a detailed explanation of sampling methods in 4.4, presenting multiple approaches, such as SDE and ODE solvers, for efficient sampling. Following this, Section 5 presents the experimental results, highlighting the performance and validation of the proposed methods. Section 6 presents the conclusions and future directions for research. Finally, Section 7 provides the notational conventions, clarifying the symbols and terminology used throughout the paper.

2 Background of the ScoreSDE Framework

To establish the connection between score-based generative models and SDEs, we examine the discrete-time formulation of DDPMs, as given in (1). We define a discrete step size $\Delta t = \frac{1}{N}$, where N represents the total number of discrete steps in the diffusion process. To facilitate the transition to a continuous formulation, we introduce an auxiliary noise schedule $\{\bar{\beta}_i\}_{i=1}^N$ where $\beta_i = \frac{\bar{\beta}_i}{N}$. By expressing β_i in terms of a continuous function, we obtain the following:

$$\beta_i = \bar{\beta}_i \cdot \frac{i}{N} = \beta\left(\frac{i}{N}\right) \cdot \frac{1}{N} = \beta(t + \Delta t)\Delta t,$$

where this work assumes that $\bar{\beta}_i \to \beta(t)$ as $N \to \infty$, which is a continuous time function for $0 \leq t \leq 1$. We let

$$\mathbf{x}_i = \mathbf{x}\left(\frac{i}{N}\right) = \mathbf{x}(t + \Delta t), \quad \boldsymbol{\varepsilon}_i = \boldsymbol{\varepsilon}\left(\frac{i}{N}\right) = \boldsymbol{\varepsilon}(t + \Delta t).$$

Hence, the Taylor expansion of $\sqrt{1-x}$ for approximation yields

$$\begin{aligned}
\mathbf{x}_i &= \sqrt{1-\beta_i}\mathbf{x}_{i-1} + \sqrt{\beta_i}\boldsymbol{\varepsilon}_{i-1} \\
\mathbf{x}_i &= \sqrt{1-\frac{\bar{\beta}_i}{N}}\mathbf{x}_{i-1} + \sqrt{\frac{\bar{\beta}_i}{N}}\boldsymbol{\varepsilon}_{i-1} \\
\mathbf{x}(t+\Delta t) &= \sqrt{1-\beta(t+\Delta t)\Delta t}\cdot\mathbf{x}(t) + \sqrt{\beta(t+\Delta t)\Delta t}\cdot\boldsymbol{\varepsilon}(t) \\
\mathbf{x}(t+\Delta t) &\approx \left(1-\frac{1}{2}\beta(t+\Delta t)\Delta t\right)\cdot\mathbf{x}(t) + \sqrt{\beta(t+\Delta t)\Delta t}\cdot\boldsymbol{\varepsilon}(t) \\
\mathbf{x}(t+\Delta t) &\approx \mathbf{x}(t) - \frac{1}{2}\beta(t)\Delta t\,\mathbf{x}(t) + \sqrt{\beta(t)\Delta t}\cdot\boldsymbol{\varepsilon}(t).
\end{aligned}$$

As $N \to \infty$, the discrete Markov chain in Eq. (1) converges to the following SDE:

$$d\mathbf{x}_t = -\frac{1}{2}\beta(t)\mathbf{x}_t\,dt + \sqrt{\beta(t)}\,d\mathbf{w}_t. \tag{4}$$

Therefore, the DDPM forward update iteration can be written equivalently as an SDE. The variance $\beta(t)$ remains bounded as $t \to \infty$ because β_i is constrained, ensuring a controlled noise increase that prevents the data from being completely overwhelmed. This bounded nature of variance is why it is referred to as the **variance-preserving (VP) SDE**.

The SMLD (2) framework has no explicit forward diffusion process as in DDPMs. However, we can approximate it using N discrete noise scales. Under this formulation, the recursive updates naturally follow a Markov chain:

$$\mathbf{x}_i = \mathbf{x}_{i-1} + \sqrt{\sigma_i^2 - \sigma_{i-1}^2}\mathbf{z}_{i-1}, \quad i = 1, \cdots, N.$$

Assuming that the limit $\{\sigma_i\}_{i=1}^N$ becomes the continuous-time function $\sigma(t)$ for $t \in [0,1]$, $\mathbf{z}_i$ becomes $\mathbf{z}(t)$, and $\{\mathbf{x}_i\}_{i=1}^N$ becomes $\mathbf{x}_t$, where $\mathbf{x}_i = \mathbf{x}(\frac{i}{N})$, then we obtain the following:

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \sqrt{\sigma^2(t+\Delta t) - \sigma^2(t)}\cdot\mathbf{z}(t) \approx \mathbf{x}(t) + \sqrt{\frac{d}{dt}\sigma^2(t)\Delta t}\cdot\mathbf{z}(t)$$

At the limit when $\Delta t \to 0$, the equation converges to

$$d\mathbf{x} = \sqrt{\frac{d}{dt}\sigma^2(t)}\,d\mathbf{w}. \tag{5}$$

Unlike the VP SDE, where noise remains bounded, the diffusion term in (5) grows exponentially with time. Hence, the variance of the data distribution diverges as $t \to \infty$, which is why this formulation is called **variance exploding SDE**.

Although SMLD and DDPM have demonstrated strong generative capabilities, they suffer from limitations, including slow sampling due to their iterative denoising steps and difficulties in likelihood evaluation. To address these challenges, the ScoreSDE model extends diffusion models to a continuous-time framework, offering critical advantages,

including efficient likelihood computation via its connection to a probability-flow ODE (PF-ODE) and greater flexibility in sampling methods. The model is formulated as a stochastic process governed by a pair of SDEs: the forward and reverse-time SDEs.

Forward SDE (data-to-noise): A general framework for many score-based generative models is where noise is injected into the data distribution $p_{\text{data}} \equiv p_0$ via a forward SDE with more general drift and diffusion coefficients:

$$d\mathbf{x}_t = f(\mathbf{x}_t, t)dt + g(\mathbf{x}_t, t)d\mathbf{w}_t, \quad \mathbf{x}_0 \sim p_0, \tag{6}$$

where $t \in [0, T]$ resides in the continuous-time domain and $\mathbf{w}_t$ is the standard Wiener process (a.k.a. BM). The drift coefficient $f(\cdot, t) : \mathbb{R}^d \to \mathbb{R}^d$ suggests how the diffusion particle should flow to the lowest energy state, whereas the diffusion coefficient $G(\cdot, t) : \mathbb{R}^d \to \mathbb{R}^d$ describes how the particle would randomly walk from one position to another, determining the strength of the random movement (i.e., the intensity of the random fluctuation).

Reverse-time SDE (noise-to-data): To generate new samples, the reverse-time SDE inverts this process, starting from noise and gradually reconstructing structured data by removing perturbations. A crucial result from [30] reveals that the reverse of a diffusion process is a diffusion process, running backward in time and governed by the **reverse-time SDE**:

$$d\mathbf{x}_t = \left\{ f(\mathbf{x}_t, t) - g(\mathbf{x}_t, t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t) \right\} dt + g(\mathbf{x}_t, t)d\bar{\mathbf{w}}_t, \tag{7}$$

where $\bar{\mathbf{w}}_t$ denotes the standard Wiener process in reverse time (i.e. time flows backward from T to 0). When the reverse-time SDE is well-constructed, we can simulate it with numerical approaches to generate new samples from p_0.

When discretizing SDEs, the step size Δt is limited by the randomness of the stochastic process [31]. A large step size (consequently, a small number of steps) often causes nonconvergence, especially in high-dimensional spaces, and the numerical scheme might fail to capture the correct dynamics, leading to a poor approximation of the solution's distribution and negatively affecting the capability of high-quality sample generation. However, when the step size is small, the iterative process that generates high-quality samples, often involving hundreds or thousands of denoising steps, makes the generation process computationally expensive and slow.

Additionally, SDE solvers do not offer a method to compute the exact log-likelihood of score-based generative models. To address those limitations, Song et al. [21] introduced a sampler based on ODEs, called the *PF ODE*, by converting any SDE into an ODE without changing its marginal distributions $\{p_t(\mathbf{x})\}_{t\in[0,T]}$. Song et al. suggested that each SDE has a corresponding PF-ODE that produces deterministic processes, sampling from the same distribution as the SDE at each timestep.

Probability-Flow ODE: The PF-ODE corresponding to Eq. (6) follows the general form:

$$d\mathbf{x}_t = \left\{ f(\mathbf{x}, t) - \frac{1}{2}\nabla \cdot \left[G(\mathbf{x}, t)G(\mathbf{x}, t)^\top \right] - \frac{1}{2} G(\mathbf{x}, t)G(\mathbf{x}, t)^\top \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t) \right\} dt. \tag{8}$$

Appendix D.1 of [21] provides a detailed derivation of transforming the reverse SDE into the PF-ODE. The proof is based on the Fokker–Planck (or the Kolmogorov forward) equation. Trajectories obtained by solving the PF-ODE (8) have the same marginal distributions as the SDE trajectories (7). The critical component in (8) is the score function $\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)$, which is approximated by neural networks. This approximation creates a parallel with neural ODEs [32], inherits all properties of neural ODEs, and enables sampling via ODE solvers and the precise computation of log-likelihoods. The PF-ODE also eliminates the stochasticity from the reverse process and allows for faster sampling in generative diffusion models while maintaining high-quality output [21]. Although the reverse process becomes deterministic, multiple diverse outputs can still be generated by starting from different random noise vectors because the ODE's initial conditions vary accordingly.

Remark 2.1. *The equations (SDEs or ODEs) above describe how a system changes or evolves over time. Numerical solvers are algorithms that take these equations and produce a series of points that form a "trajectory" or "path" of how the system evolves by breaking time into small steps and applying approximate formulas. This trajectory is an approximation rather than an exact representation; however, it is typically sufficient to capture and understand the behavior of a dynamic system. In generative modeling, we generate new images by numerically solving Eqs.* (7) *and* (8).

3 Protein Backbone Representation: α-carbon Distance Map

Proteins are macromolecules comprising linear chains of amino acids. Each amino acid residue contributes three principal backbone atoms: nitrogen (N), alpha carbon (C_α), and carbonyl carbon (C), as well as carbonyl oxygen (O). These repeating units form the polypeptide backbone $N - C_\alpha - C - O$, with side chains branching from the C_α atom. Various computational methods have been developed to predict protein function. These methods can be categorized into four groups based on the information they employ (there is overlap and correlation between them). Sequence-based methods include BLAST [33], DEEPred [34], and DeepGOPlus [35], etc.). The 3D structure-based methods include DeepFRI [36], LSTM-LM [37], and HEAL [38]. In addition, the PPI network-based methods include GeneMANIA [39] and deepNF [40], among others. Finally, the hybrid information-based methods include the CAFA challenge [41]. A comprehensive review and comparison of those methods is presented in [42]. Diffusion models have recently been adopted into protein engineering applications and have demonstrated strong performance in generating novel protein structures and sequences [43, 44, 45].

The dataset was derived from the Protein Data Bank (PDB) [46], a globally recognized repository archiving detailed 3D structural biomolecule data, including proteins, nucleic acids, and complex assemblies. This benchmark protein dataset was established in 1971, serving as a vital resource for researchers in structural biology, bioinformatics, and related fields and facilitating advances in drug discovery, molecular biology, and biotechnology.

The PDB offers free access to the structural data submitted by scientists worldwide in a standardized format that supports computational analysis and visualization.

Figure 1 illustrates the data preparation process, beginning with extracting atomic coordinates from a protein structure. The 3D model on the left represents the atomic arrangement of 101M sperm whale myoglobin, obtained from the PDB. The table in the center displays a segment of the PDB file, detailing atomic positions in a structured format. Using the extracted (X, Y, Z) coordinates, the Euclidean distances between pairs of C_α atoms are computed to construct the distance matrix, visualized as a heatmap (right). The color gradient (yellow to blue) encodes distance magnitudes, with the diagonal representing zero distance, corresponding to self-comparisons. An invariant representation is necessary to analyze protein structures independent of their spatial orientation. Proteins can adopt arbitrary poses in 3D space; hence, direct coordinate-based representations are sensitive to translation and rotation. Computing pairwise distances between C_α atoms yields a structured representation, the distance matrix, which remains invariant to such transformations while preserving critical structural information. This representation is useful because it retains the sequential ordering of amino acids, defined by the N-terminus to C-terminus directionality, ensuring uniqueness, and it preserves sufficient information to recover the structure [47].

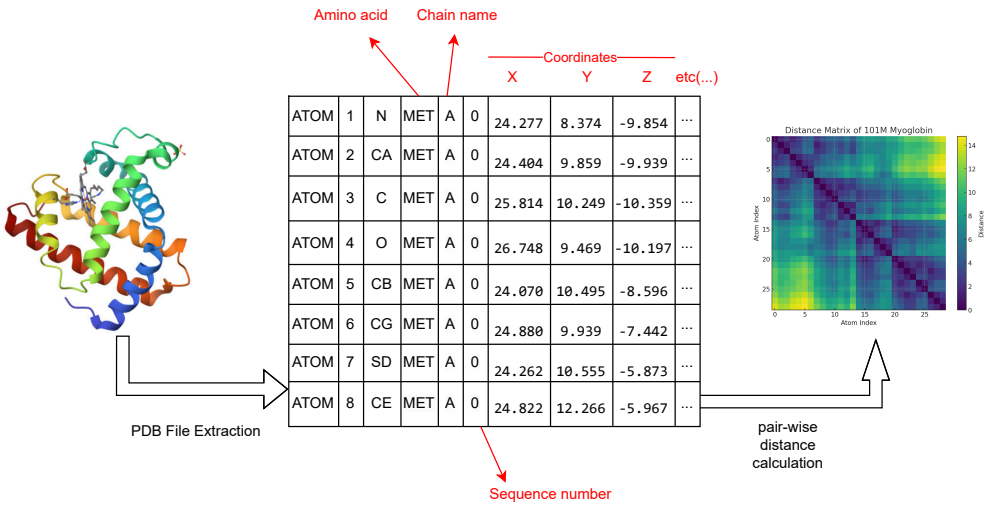

Figure 1: Protein structure representation and distance matrix.

Following the generation of the C_α distance map using a trained model, the 3D structure of the protein backbone can be reconstructed from the resulting image. Recovering or folding the 3D structure of a protein from the α-carbon distance matrix is a classical inverse problem in structural biology, which many studies have investigated. In the existing literature on 3D protein structure recovery, notable methods include the hidden Markov model (HMM)–based approaches FB5-HMM [48] and TorusDBN [49], the multiscale torsion angle GAN, 3DGAN [50], and a full-atom GAN.

Moreover, Anand et al. [47] introduced a GAN-based approach to generate sequence-

sensitive, fixed-length protein structure fragments. Using these GANs, they produced C_α pairwise distance maps and used the alternating direction method of multipliers (ADMM) alongside Rosetta minimization [51] to reconstruct the full protein structures from the derived distance constraints. The ADMM is an algorithm proposed in [52] that solves convex optimization problems by breaking a complicated one into smaller pieces, each of which is easier to handle. The ADMM follows a decomposition-coordination strategy, in which solutions to smaller local subproblems are synchronized to solve a broader global problem. Building on [47], Anand et al. [53] further refined the method by training deep neural networks to recover and refine full-atom pairwise distance matrices accurately for fixed-length fragments.

4 Generative Fractional Diffusion Models

This section introduces the driving process, the fBm, along with its Markov approximation. Next, the generative fractional diffusion models are presented, followed by score-matching techniques designed to estimate the corresponding score function. Finally, this section discusses the SDE and ODE solvers used to model the data generation and sampling process in detail.

4.1 Fractional driving noise and its Markov approximation

Definition 4.1 (Fractional Brownian Motion (Types I and II))**.** *The fBm family of continuous-time Gaussian processes is parameterized by the Hurst index $H \in (0,1)$, which controls the smoothness and correlation of the process. It generalizes the standard BM and exhibits self-similarity and stationary increments. Two common types of fBm exist:*

- ***Type I:*** *Denoted by W_t^H, it contains the covariance function:*

$$\mathbb{E}\big[W_t^H W_s^H\big] = \frac{1}{2}\big(|t|^{2H} + |s|^{2H} - |t-s|^{2H}\big).$$

- ***Type II:*** *Denoted by B_t^H, its covariance function is given by*

$$\mathbb{E}\big[B_t^H B_s^H\big] = \frac{1}{(\Gamma(H+\frac{1}{2}))^2}\int_0^s \big((t-u)(s-u)\big)^{H-\frac{1}{2}}\,\mathrm{d}u, \quad s<t,$$

where $\Gamma(\cdot)$ denotes the Gamma function. Type II is often referred to as the **Riemann–Liouville Volterra process***, which is commonly applied in financial mathematics. The Hurst index H governs the process behavior:*

- *$H = 1/2$ corresponds to the standard BM.*
- *If $H > 1/2$, the process increments are positively correlated.*
- *If $H < 1/2$, the process increments are negatively correlated.*

Figure 2 illustrates how the Hurst index influences the smoothness of the fBm paths. When the Hurst index $H > 1/2$, the increments of the fBm are positively correlated, making its trajectory smoother than the standard BM. Conversely, when $H < 1/2$, the increments are negatively correlated, resulting in a rougher path compared to the BM.

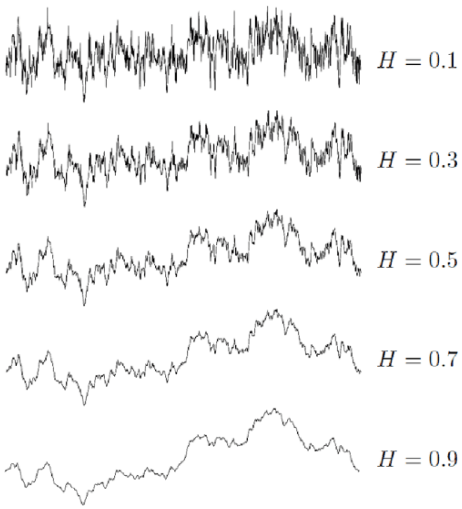

Figure 2: Sample paths of fractional Brownian motion (fBm) by Hurst index H values ([54]).

The fBm differs from standard BM in that it has long-range dependence, and its increments are not independent, making it non-Markovian. More precisely, the Markov property implies that future states depend only on the present, whereas fBm incorporates memory effects, where past values influence future behavior via the Hurst parameter. Additionally, fBm is not a semimartingale because it lacks the necessary decomposition as a local martingale and a finite variation process, restricting it from being applied directly in stochastic calculus like standard BM. This nonsemimartingale property limits its compatibility with Itō calculus.

The nature of fBm poses a challenge for score-based generative models, which typically rely on Markovian SDEs where the future state depends only on the present. In [27], the authors developed a framework for approximating fBm using Markov processes, called MA-fBm. Their critical contribution is constructing finite-dimensional, affine Markov processes approximating the behavior of fBm, which lacks the Markov property due to its long-range dependence. This approximation is crucial for the efficient simulation of fractional diffusion processes and ensures compatibility with score-based generative modeling. This approach enables the application of existing score-based generative models and well-established methods in the standard diffusion framework, facilitating the training and sampling of models driven by fractional stochastic processes.

Definition 4.2 (Markov Approximation of fBm)**.** *Choose $K \in \mathbb{N}$ Ornstein–Uhlenbeck (OU) process with speeds of mean reversion $\gamma_1, \ldots, \gamma_K$ and dynamics $\mathrm{d}Y_t^k = -\gamma_k Y_t^k \mathrm{d}t + \mathrm{d}B_t$:*

$$Y_t^k = Y_0^k e^{-\gamma_k t} + \int_0^t e^{-\gamma_k (t-s)} \mathrm{d}B_s, t \geq 0,$$

where $Y_0^k = \int_{-\infty}^0 e^{\gamma_k s} \mathrm{d}B_s$ *(Type I) and* $Y_0^k = 0$ *(Type II). Given a Hurst index* $H \in (0,1)$ *and a geometric space grid* $\gamma_k = r^{k-n}$ *with* $r > 1$ *and* $n = \frac{K+1}{2}$*, the process*

$$\hat{B}_t^H := \sum_{k=1}^K \omega_k (Y_t^k - Y_0^k), H \in (0,1), t \geq 0,$$

is the MA-fBm with approximation coefficients $\omega_1, \ldots, \omega_K \in \mathbb{R}$*.*

To approximate fBm B_t^H with minimal errors, we choose a geometrically spaced grid

$$\gamma_k = (r^{1-n}, r^{2-n}, \ldots, r^{K-n}),$$

where $n = \frac{K+1}{2}$ and $r > 1$. The optimal approximation coefficients $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_K) \in \mathbb{R}^K$, for a given Hurst index $H \in (0,1)$ and terminal time $T > 0$, are obtained by minimizing the $L^2(P)$-error:

$$\varepsilon(\boldsymbol{\omega}) := \int_0^T \mathbb{E}\Big[\big(B_t^H - \hat{B}_t^H\big)^2\Big] \mathrm{d}t.$$

The optimal coefficients satisfy the following closed-form system:

$$\mathbf{A}\boldsymbol{\omega} = \mathbf{b},$$

where matrix $\mathbf{A}$ and vector $\mathbf{b}$ are given for the **Type I fBm approximation** by

$$\mathbf{A}_{i,j} := \frac{2T + \frac{e^{-\gamma_i T} - 1}{\gamma_i} + \frac{e^{-\gamma_j T} - 1}{\gamma_j}}{\gamma_i + \gamma_j},$$

$$\mathbf{b}_k := \frac{2T}{\gamma_k^{H+\frac{1}{2}}} - \frac{T^{H+\frac{1}{2}}}{\gamma_k \Gamma(H + \frac{3}{2})} + \frac{e^{-\gamma_k T} - Q(H + \frac{1}{2}, \gamma_k T) e^{\gamma_k T}}{\gamma_k^{H+\frac{3}{2}}},$$

where $Q(z,x)$ represents the **regularized upper incomplete gamma function**:

$$Q(z,x) = \frac{1}{\Gamma(z)} \int_x^\infty t^{z-1} e^{-t} \mathrm{d}t.$$

For **Type II fBm approximation**, the corresponding expressions for $\mathbf{A}$ and $\mathbf{b}$ are

$$\mathbf{A}_{i,j} := \frac{T + \frac{e^{-(\gamma_i + \gamma_j)T} - 1}{\gamma_i + \gamma_j}}{\gamma_i + \gamma_j},$$

$$\mathbf{b}_k := \frac{T}{\gamma_k^{H+\frac{1}{2}}} P\big(H + \frac{1}{2}, \gamma_k T\big) - \frac{H + \frac{1}{2}}{\gamma_k^{H+\frac{3}{2}}} P\big(H + \frac{3}{2}, \gamma_k T\big),$$

where $P(z,x)$ denotes the **regularized lower incomplete gamma function**:

$$P(z,x) = \frac{1}{\Gamma(z)} \int_0^x t^{z-1} e^{-t} \mathrm{d}t.$$

For further discussion on practical considerations in selecting geometric sequences γ_k and the time horizon for optimizing the coefficients $\boldsymbol{\omega}$, see [28]. The experiments in **Section 5** focus on the Type II fBm approximation.

4.2 Fractional noise-driven generative diffusion model

The framework is based on a simplified SDE where the diffusion coefficient does not depend on $\mathbf{X}_t$, that is, the diffusion scale g in (6) is only time-dependent, rather than time- and state-dependent:

$$\mathrm{d}\mathbf{X}_t = u(t)\mathbf{X}_t\mathrm{d}t + g(t)\mathrm{d}\hat{\boldsymbol{B}}_t^H, \quad \mathbf{X}_0 \sim p_0. \tag{9}$$

The OU processes Y_t^k, $k = 1, \ldots, K$ approximate $\hat{\boldsymbol{B}}_t^H$ satisfying the dynamics $\mathrm{d}Y_t^k = -\gamma_k Y_t^k \mathrm{d}t + \mathrm{d}B_t$; thus, we have the following:

$$\mathrm{d}\hat{\boldsymbol{B}}_t^H = -\sum_{k=1}^{K} \omega_k \gamma_k Y_t^k \mathrm{d}t + \sum_{k=1}^{K} \omega_k \mathrm{d}\mathbf{B}_t. \tag{10}$$

Rewriting the dynamics (9) with (10) yields

$$\mathrm{d}\mathbf{X}_t = \Big[u(t)\mathbf{X}_t - g(t)\sum_{k=1}^{K} \omega_k \gamma_k Y_t^k\Big]\mathrm{d}t + \sum_{k=1}^{K} \omega_k g(t)\mathrm{d}\mathbf{B}_t. \tag{11}$$

Considering the forward and OU processes defining the driving noise $\hat{B}_t^H$, we have an augmented vector of the correlated process $Z \equiv (X, Y^1, \ldots, Y^K) = (Z_t)_{t\in[0,T]}$, driven by the same BM

$$\mathrm{d}\mathbf{Z}_t = \mathbf{F}(t)\mathbf{Z}_t\mathrm{d}t + \mathbf{G}(t)\mathrm{d}\mathbf{B}_t, \quad t \in [0, T], \tag{12}$$

where

$$\mathbf{F}(t) = \begin{pmatrix} u(t) & -g(t)\omega_1\gamma_1 & -g(t)\omega_2\gamma_2 & \ldots & -g(t)\omega_K\gamma_K \\ 0 & -\gamma_1 & 0 & \ldots & 0 \\ 0 & 0 & -\gamma_2 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & -\gamma_K \end{pmatrix}_{(K+1)\times(K+1)}$$

and

$$\mathbf{G}(t) = \begin{pmatrix} \sum_{k=1}^{K} \omega_k g(t) & 0 & 0 & \ldots & 0 \\ 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \end{pmatrix}_{(K+1)\times(K+1).}$$

The reverse time model of the GFDM (12) is given by the backward dynamics

$$\mathrm{d}\mathbf{Z}_t = \Big\{\mathbf{F}(t)\mathbf{Z}_t - \mathbf{G}(t)\mathbf{G}(t)^T\nabla_{\mathbf{z}}\log\, p_t(\mathbf{Z}_t)\Big\}\mathrm{d}t + \mathbf{G}(t)\mathrm{d}\mathbf{B}_t, \quad t \in [0, T], \tag{13}$$

and its PF-ODE is given by

$$\mathrm{d}\mathbf{z}_t = \left\{ \mathbf{F}(t)\mathbf{z}_t - \frac{1}{2}\mathbf{G}(t)\mathbf{G}(t)^T \nabla_{\mathbf{z}} \log\, p_t(\mathbf{z}_t) \right\} \mathrm{d}t, \quad t \in [0, T]. \tag{14}$$

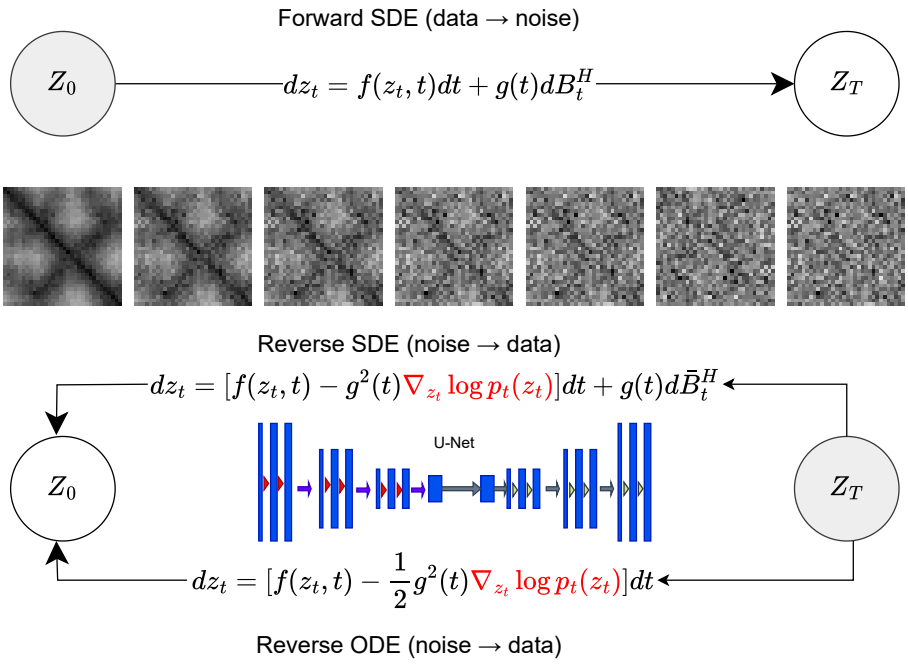

Figure 3: Overview of score-based generative modeling through stochastic and ordinary differential equations (SDEs and ODEs).

Figure 3 breaks down the process of the ScoreSDE model, illustrating its critical components and their interactions. The ScoreSDE employs SDEs to model data generation as a diffusion process. The forward SDE progressively perturbs actual data into a Gaussian noise distribution. In contrast, the reverse SDE inverts this process by iteratively denoising the samples, guided by a learned score function $\nabla_{z_t} \log p_t(z_t)$. Alternatively, a reverse ODE can be applied for a deterministic sampling approach, improving stability and efficiency. A U-Net architecture is commonly employed to parameterize the score function $\nabla_{z_t} \log p_t(z_t)$, facilitating precise noise estimation and denoising.

4.3 Augmented score-matching technique

The score function (the gradient of the log-probability density p_{data}), $\nabla_{\mathbf{x}} \log p_t(x_t)$, which appears in (7) or (8), guides the transformation of pure noise into coherent images by supplying directional information on how to increase the data likelihood at each step of the reverse diffusion process. **Score matching** is a nonlikelihood-based method to perform sampling on an unknown data distribution and aims to address many of the limitations of likelihood-based methods (e.g., VAE and autoregressive models) and adversarial methods

(GANs). A good estimation of the score function is crucial because it directly determines the quality, realism, and fidelity of the data generated using score-based generative models. In contrast, an inaccurate score estimation can lead to poor generation results, including artifacts, mode collapse, or failure to capture an accurate data distribution.

In practice, the score function is learned using a neural network $\mathbf{s}_\theta(\mathbf{x})$ parameterized by θ, obtained by minimizing the Fisher divergence between the true score function and the network:

$$J(\theta) = \mathbb{E}_{p_{\text{data}}(\mathbf{x})}\left[\|\mathbf{s}_\theta(\mathbf{x}) - \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})\|_2^2\right]. \tag{15}$$

The problem is that p_{data} is never accessible. Various methods can estimate the score function without knowledge of the ground-truth data score.

Implicit score matching (ISM) Hyvärinen and Dayan [55] introduced the implicit score matching method and demonstrated its equivalence to $J(\theta)$ in (15), up to a constant, under mild regulatory conditions:

$$J_{ISM}(\theta) = \mathbb{E}_{p_{\text{data}}(\mathbf{x})}\left[\frac{1}{2}\|\mathbf{s}_\theta(\mathbf{x})\|_2^2 + \text{tr}(\nabla_{\mathbf{x}}\mathbf{s}_\theta(\mathbf{x}))\right] + \text{constant}, \tag{16}$$

where $\text{tr}(\nabla_{\mathbf{x}}\mathbf{s}_\theta(\mathbf{x}))$ denotes the trace of the Hessian matrix of the log-probability with respect to $\mathbf{x}$. Equation (16) is derived by simplifying (15) using the generalized integration by parts (mutidimensional) technique. Minimizing $J_{ISM}(\theta)$ does not require the true target scores $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$, and we only need to compute an expectation with respect to the data distribution, which can be implemented using finite samples from the dataset using the Markov chain Monte Carlo method.

Denoising score matching (DSM) An alternative approach is DSM, proposed by Vincent in [58]. The DSM approach introduces noise perturbations to the data $\mathbf{x}$, creating a smoothed version of the data distribution $\tilde{\mathbf{x}}$. Instead of directly modeling the original distribution, the model learns the score function of the corrupted distribution. Given noisy data $\tilde{\mathbf{x}} = \mathbf{x} + \sigma\boldsymbol{\epsilon}$, where σ controls noise intensity, the model minimizes the following DSM objective:

$$\begin{aligned} J_{DSM}(\theta) &= \mathbb{E}_{q_\sigma(\tilde{\mathbf{x}},\mathbf{x})}\left[\frac{1}{2}\|\mathbf{s}_\theta(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})\|_2^2\right] \\ &= \mathbb{E}_{q_\sigma(\tilde{\mathbf{x}},\mathbf{x})}\left[\frac{1}{2}\left\|\mathbf{s}_\theta(\tilde{\mathbf{x}}) - \frac{\mathbf{x}-\tilde{\mathbf{x}}}{\sigma^2}\right\|_2^2\right] \end{aligned}$$

over the joint density on original-corrupt data pairs $(\tilde{\mathbf{x}}, \mathbf{x})$, which is $q_\sigma(\tilde{\mathbf{x}}, \mathbf{x}) = q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})p_{\text{data}}(\mathbf{x})$. Moreover, $\nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})$ is not the data score, but it provides the information of the direction of moving from $\tilde{\mathbf{x}}$ back to the original $\mathbf{x}$ (that is the reason this method is called "denoising").

Sliced score matching (SSM) The loss function in (16) requires computing the trace, which remains intractable in high-dimensional spaces, as analyzed in [56]. Song et al. proposed SSM [57, 59] to scale up the computation of score matching considerably. They projected the scores in random directions, such that the vector fields of the scores of

the data and model distribution become scalar fields, motivated by the fact that a one-dimensional data distribution is much easier to estimate for score matching. Then, they analyzed the scalar fields to assess the disparity between the model and data distributions. The trackable training objective can be considered a random projected version of the Fisher divergence of (15):

$$J_{SSM}(\theta) = \mathbb{E}_{p_{\mathbf{v}}}\mathbb{E}_{p_{\text{data}}(\mathbf{x})}\left[\frac{1}{2}\|\mathbf{s}_\theta(\mathbf{x})\|^2 + \mathbf{v}^T\nabla_{\mathbf{x}}\mathbf{s}_\theta(\mathbf{x})\mathbf{v}\right] + \text{constant},$$

where $\mathbf{v} \sim N(0, \mathbf{I})$ represents the random projection direction, and $p_{\mathbf{v}}$ denotes its distribution. The SSM approach is considered one alternative to DSM that is consistent and computationally efficient.

Noise-conditioned score network In the previous DSM approach, the strength σ determines how well the corrupted data distribution $q_\sigma(\tilde{\mathbf{x}})$ aligns with the original distribution $p_{\text{data}}(\mathbf{x})$. A trade-off occurs in selecting σ. A larger σ value helps capture low-density regions of the data distribution, improving the score estimation. However, if σ is too large, the data distribution becomes excessively corrupted, making learning challenging. Conversely, a smaller σ preserves the original data distribution more accurately but fails to perturb the data sufficiently, poorly covering low-density regions. To address this problem, Song and Ermon introduced a multiscale noise perturbation approach [59, 60], applying noise at multiple levels simultaneously. Specifically, they defined L noise-perturbed data distributions, each associated with a different noise scale σ, ordered such that

$$\sigma_1 > \sigma_2 > \cdots > \sigma_L,$$

with higher indices corresponding to lower noise levels. The core idea behind the noise-conditioned score network method is perturbing the data distribution with the isotropic Gaussian noise $N(0, \sigma_i^2\mathbf{I}), i = 1, \ldots, L$ to obtain a noise-perturbed distribution:

$$p_{\sigma_i}(\mathbf{x}) = \int p(\mathbf{y})N(\mathbf{x}; \mathbf{y}, \sigma_i^2\mathbf{I})d\mathbf{y}.$$

The next step is to train the noise-conditioned score-based model $\mathbf{s}_\theta(\mathbf{x}, i)$ to estimate the score function of each noise-perturbed distribution, for $i = 1, \ldots, L$:

$$\mathbf{s}_\theta(\mathbf{x}, i) \approx \nabla_{\mathbf{x}} \log p_{\sigma_i}(\mathbf{x}).$$

The training objective is the weighted sum of Fisher divergences for all noise scales:

$$\sum_{i=1}^{L} \lambda(i)\mathbb{E}_{p_{\sigma_i}(\mathbf{x})}\left[\|\mathbf{s}_\theta(\mathbf{x}, i) - \nabla_{\mathbf{x}} \log p_{\sigma_i}(\mathbf{x})\|_2^2\right],$$

where $\lambda(i) \in \mathbb{R}$ denotes a positive weighting function, often set to $\lambda(i) = \sigma_i^2$.

Remark 4.3. *In diffusion generative models, two primary approaches are commonly employed to define the model task:* data input prediction models *and* noise prediction models.

For data input prediction models*, the model is trained to directly predict the original data (e.g., the clean image)* $\mathbf{x}_0$ *from the noisy data* $\mathbf{x}_t$*. At each step in the reverse process, the model estimates the clean data conditioned on the noisy input. For* noise prediction models*, the model is trained to predict the added noise* $\boldsymbol{\varepsilon}$ *in the noisy data* $\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\varepsilon}$*. This approach focuses on estimating the noise rather than the clean data.*

This work applies augmented score-matching techniques to learn the score function $\nabla_{\mathbf{z}} \log p_t$, which appears in (13). The authors in [26] suggested that one can condition the score function $\nabla_{\mathbf{Z}} \log p_t(\mathbf{Z}_t)$ on a data sample $\mathbf{X}_0 \sim p_0$ and on the states of the stacked vector $\mathbf{Y}_t^{[K]} := (\mathbf{Y}_1^1, \ldots, \mathbf{Y}_t^K)$ of augmenting processes. A time-dependent score model $s_{\boldsymbol{\theta}}$ can be trained by minimizing the following augmented score-matching loss:

$$\mathcal{L}(\boldsymbol{\theta}) := \mathbb{E}_{t\sim U[0,T]} \Bigg\{ \mathbb{E}_{\mathbf{X}_0, \mathbf{Y}_t^{[K]}} \mathbb{E}_{\mathbf{X}_t | \mathbf{Y}_t^{[K]}, \mathbf{X}_0} \Big\| s_{\boldsymbol{\theta}}\Big(\mathbf{X}_t - \sum_k \eta_t^k Y_t^k, t\Big) - \nabla_{\mathbf{x}} \log p_{0t}(\mathbf{X}_t | \mathbf{Y}_t^{[K]}, \mathbf{X}_0) \Big\|_2^2 \Bigg\}.$$

The weights $\eta_t^k, \ldots, \eta_t^k$ arise from conditioning $\mathbf{Z}_t$ on $\mathbf{Y}_t^{[K]}$. We assume that $s_{\boldsymbol{\theta}}$ is optimal with respect to the augmented score matching loss $\mathcal{L}$. The score model

$$S_{\boldsymbol{\theta}}(\mathbf{Z}_t, t) := \Bigg(s_{\boldsymbol{\theta}}\Big(\mathbf{X}_t - \sum_k \eta_t^k Y_t^k, t\Big), -\eta_t^1 s_{\boldsymbol{\theta}}\Big(\mathbf{X}_t - \sum_k \eta_t^k Y_t^k, t\Big), \ldots, - \eta_t^K s_{\boldsymbol{\theta}}\Big(\mathbf{X}_t - \sum_k \eta_t^k Y_t^k, t\Big)\Bigg)$$

yields the optimal L^2 approximation of $\nabla_{\mathbf{z}} \log\, p_t(\mathbf{Z}_t)$ via

$$\nabla_{\mathbf{z}} \log\, p_t(\mathbf{Z}_t) \approx S_{\boldsymbol{\theta}}(\mathbf{Z}_t, t) + \nabla_{\mathbf{Y}} \log\, q_t(\mathbf{Y}_t^{[K]}). \tag{17}$$

The random vector $\mathbf{Y}_t^{[K]}$ is a centered Gaussian process with a covariance matrix $\boldsymbol{\Sigma}_t^{\mathbf{y}} \in \mathbb{R}^{K,K}$, where $[\boldsymbol{\Sigma}_t^{\mathbf{y}}]_{k,l} = \mathbb{E}[Y_t^k Y_t^l]$. We can directly calculate the related score function using

$$\nabla_{\mathbf{Y}} \log\, q_t(\mathbf{Y}_t^{[K]}) = -\boldsymbol{\Sigma}_t^{\mathbf{y}} \mathbf{Y}_t^{[K]}.$$

After training the score model $S_{\boldsymbol{\theta}}(\mathbf{Z}_t, t)$ using augmented score matching, we can simulate the reverse-time model by running it backward and generate samples by discretizing the following reverse-time SDE (**augmented reverse-time SDE**) from T to 0:

$$\mathrm{d}\mathbf{Z}_t = \Big\{ \mathbf{F}(t)\mathbf{Z}_t - \mathbf{G}(t)\mathbf{G}(t)^T \Big[S_{\boldsymbol{\theta}}(\mathbf{Z}_t, t) + \nabla_{\mathbf{Z}_t} \log q_t(\mathbf{Y}_t^{[K]}) \Big] \Big\} \mathrm{d}t + \mathbf{G}(t) \mathrm{d}\bar{\mathbf{B}}_t, \tag{18}$$

or the corresponding **augmented PF-ODE**:

$$\mathrm{d}\mathbf{z}_t = \Big\{ \mathbf{F}(t)\mathbf{z}_t - \frac{1}{2}\mathbf{G}(t)\mathbf{G}(t)^T \Big[S_{\boldsymbol{\theta}}(\mathbf{z}_t, t) + \nabla_{\mathbf{z}_t} \log q_t(\mathbf{y}_t^{[K]}) \Big] \Big\} \mathrm{d}t. \tag{19}$$

Commonly, (18) and (19) are called the **diffusion SDE** and **diffusion ODE**, respectively. Because diffusion ODEs have no random term, one can obtain an exact solution $\mathbf{x}_0$, given $\mathbf{x}_T$, by solving the diffusion ODEs using the corresponding numerical solvers.

4.4 Sampling methods

In generative modeling,"sampling" refers to the function of generating new data points from the learned model, creating output that resembles the training data. Efficient sampling is critical for practically deploying these models in real-world applications, such as protein, image, and audio generation, where computational resources and time are oftentimes limited. For instance, although the DDPM generates high-fidelity samples, their practical utility is limited by slow sampling speeds.

Researchers have significantly advanced the study of sampling theory with diffusion models. For example, in [61], the sampling approach was extended to the denoising diffusion implicit model (DDIM) scheme. Unlike DDPMs, which rely on a stochastic reverse diffusion process, DDIMs use a deterministic sampling process and reduce the number of sampling steps from hundreds (in DDPMs) to as few as 10 to 50 steps without sacrificing image quality.

In the ScoreSDE model, the generation process is governed by simulating the trajectories of differential equations. Samplers for diffusion models typically discretize either the reverse-time SDE (Eq. (7)) or the PF-ODE (Eq. (8)). Numerical solvers for SDEs or ODEs are employed to transform random noise iteratively into realistic samples. The importance of sampling methods in diffusion models cannot be overstated, and these methods often trade between speed (sample efficiency) and accuracy (sample quality).

SDE solver When numerically simulating an SDE, one typically employs various methods, such as the Euler–Maruyama scheme, stochastic Runge–Kutta (RK) method [62] or more sophisticated schemes, such as Milstein's method [63]. Numerical solvers display varying behaviors in terms of approximation errors. Directly applying a standard numerical solver to an SDE may introduce varying degrees of error.

Song et al. [21] introduced a hybrid sampling algorithm, refining the Euler–Maruyama method using a **predictor-corrector (PC) sampler**, as outlined in Algorithm 1. This approach is inspired by PC methods, a class of numerical techniques commonly used for solving systems of equations [64]. The framework was designed to reduce discretization errors in reverse-time SDEs by introducing corrective steps during sample generation. The method operates in two stages per iteration:

1. **Prediction** – A numerical solver (e.g., Euler–Maruyama) estimates the sample at the next time step, serving as a "predictor."

2. **Correction** – A score-based Markov chain Monte Carlo method adjusts the estimated sample's marginal distribution, acting as a "corrector."

By iteratively refining the sample at each time step, the PC sampler improves stability and accuracy in the generative process.

Algorithm 1 Predictor-corrector (PC) sampling

Require: N: Number of discretization steps for the reverse-time SDE
Require: M: Number of corrector steps
Initialize $\mathbf{x}_N \sim p_T(\mathbf{x})$
for $i = N-1$ to 0 **do**
  $\mathbf{x}_i \leftarrow \text{Predictor}(\mathbf{x}_{i+1})$
  for $j = 1$ to M **do**
    $\mathbf{x}_i \leftarrow \text{Corrector}(\mathbf{x}_i)$
  end for
end for
return $\mathbf{x}_0$

In general, the predictor can be any numerical solver for the reverse-time SDE with a fixed discretization strategy (e.g., a *reverse diffusion sampler* (Eq. (46) in [21]) or an *ancestral sampling* (Eq. (4) in [21])). The corrector is typically Markov chain Monte Carlo based, such as Langevin dynamics or Hamiltonian Monte Carlo, and solely relies on a score function. For example, Algorithm 2 illustrates that the reverse diffusion SDE solver can be set as the predictor and annealed Langevin dynamics as the corrector, where θ^* denotes the optimal parameter of the networks s_θ and $\{\epsilon_i\}_{i=0}^{N-1}$ denotes the step sizes for Langevin dynamics.

Algorithm 2 PC sampling (VP SDE)

Require: N: Number of discretization steps for the reverse-time SDE
Require: M: Number of corrector steps
$\mathbf{x}_N \sim \mathcal{N}(0, \mathbf{I})$
for $i = N-1$ to 0 **do**
  $\mathbf{x}'_i \leftarrow (2 - \sqrt{1-\beta_{i+1}})\mathbf{x}_{i+1} + \beta_{i+1}\mathbf{s}_{\theta^*}(\mathbf{x}_{i+1}, i+1)$
  $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$
  $\mathbf{x}_i \leftarrow \mathbf{x}'_i + \sqrt{\beta_{i+1}}\mathbf{z}$ Predictor
  for $j = 1$ to M **do**
    $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$ Corrector
    $\mathbf{x}_i \leftarrow \mathbf{x}_i + \epsilon_i \mathbf{s}_{\theta^*}(\mathbf{x}_i, i) + \sqrt{2\epsilon_i}\mathbf{z}$
  end for
end for
return $\mathbf{x}_0$

ODE solver The sampling of the continuous-time diffusion model can also be created by solving the corresponding PF-ODEs of the generative model because such an ODE has the remarkable property that its solution at each time t shares the same marginal distribution as the original SDE solution at that time. Because the PF-ODE is deterministic, it often enables more sophisticated and adaptive ODE solvers. These solvers can take larger steps or apply adaptive step sizing without worrying about capturing random increments at each step. This approach might result in faster numerical methods with better convergence properties, especially when scaling up to high-dimensional problems. One potential

advantage of working with the PF-ODE as an alternative is its ability to enable faster sampling. Moreover, Chen et al. [65] demonstrated the polynomial-time convergence guarantees for PF-ODEs in the context of score-based generative models, emphasizing their theoretical robustness and efficiency.

5 Experimental Setup and Results

Explicit fractional forward dynamics The framework presented in [26] is not limited to a specific choice of drift and diffusion coefficients. The experiments focus on **fractional VP (FVP)** dynamics with different noise schedule values, which are given by

$$dX_t = -\frac{1}{2}\beta(t)X_t\,dt + \sqrt{\beta(t)}\,d\tilde{B}_t^H, \quad t \in [0,T].$$

Noise schedule Noise schedules (i.e., f and G in Eq. (6)) in diffusion models determine the addition and removal of noise during the diffusion process, significantly influencing sampling quality, training stability, and convergence. The correct selection of the noise schedule and steps is critical to optimizing model performance. In [22], the authors introduced linear or quadratic schedules because they are simple and intuitive to implement. However, some authors [29, 68] have criticized this approach, emphasizing that the steep decline in the initial time steps creates challenges for the neural network model during generation. Alternative noise scheduling functions with a more gradual decay have been proposed to address this problem. In particular, a cosine noise schedule [29] improves log-likelihoods compared with the conventional linear noise schedule in diffusion models. Unlike the linear schedule, the cosine schedule ensures a smoother progression of noise levels, introducing noise more gradually at the beginning and end while accelerating the transition in the middle stages. This adaptive noise scaling preserves structural information early on while maintaining efficient denoising in later steps to improve sample quality. Table 1 summarizes these two types of noise schedules.

Noise Schedule	**Mathematical Expression**
Linear	$\beta(t) = \beta_{\min} + (\beta_{\max} - \beta_{\min})t$
Cosine	$\beta(t) = \beta_{\min} + (\beta_{\max} - \beta_{\min})\left(\frac{1}{2}(1 - \cos(\pi t))\right)$

Table 1: Noise schedule

In Table 1, hyperparameters $(\beta_{\min}, \beta_{\max}) = (0.1, 20)$, remaining consistent with the settings in [22, 26].

Architecture details for the neural networks All experiments employed the U-Net [69] architecture to evaluate the designed score function and an Adam optimizer using PyTorch's OneCycle learning rate scheduler. The U-Net architecture is structured with multiple stages, each containing three residual blocks. The architecture applies a channel multiplication strategy defined by the sequence $[1, 2, 2, 2, 2]$, indicating that the number of channels increases progressively from the first stage, and the following stages double the

number of feature channels, enhancing the ability of the network to capture increasingly complex and abstract features at deeper layers. The models were trained with a maximal learning rate of 10^{-4} for 50k iterations and a batch size of 1,024. The training was conducted on a single Nvidia A6000 GPU, requiring approximately 17 h per training session. Table 2 summarizes the training hyperparameters.

Hyperparameter	**Value**
Model architecture	Conditional U-Net
Optimizer	Adam
Learning rate	10^{-4}
Batch size	1024
Training steps	50,000
Residual blocks	3
Channel multiplication	$[1, 2, 2, 2, 2]$
Input size	32×32
Attention resolution	$[4, 2]$
Dropout	0.1

Table 2: Training hyperparameter summary.

Evaluation metrics The Fréchet inception distance (FID) is a widely used metric in generative modeling to evaluate the quality of generated samples. The FID quantifies the similarity between the distribution of the generated data and that of the real data by computing the Fréchet distance between their feature representations. The mathematical formula for the FID is provided in Eq. (20):

$$\mathrm{FID} := \|\mu_r - \mu_g\|_2^2 + \mathrm{Tr}\left(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{\frac{1}{2}}\right), \tag{20}$$

where μ_g and Σ_g represent the mean and covariance of the generated distribution, and μ_r and Σ_r represent the mean and covariance of the distribution known to the model.

A lower FID score signifies a higher similarity between the generated and real data, indicating better generative quality. The FID provides a useful and widely adopted measure of generative model quality by capturing differences in distributional statistics. However, as a single scalar value, this metric condenses the comparison between two distributions, making it less informative regarding distinct aspects, such as fidelity (realism of generated samples) and diversity (variety of generated samples) [70]. Although the FID remains a valuable benchmark metric, it is complementary to other evaluation metrics because it does not explicitly identify one-to-one matches between distributions, can be sensitive to outliers, and depends on specific evaluation settings [72]. Combining the FID with additional measures provides a more comprehensive assessment of generative model performance. The *precision and recall metric* [70] and its improved version [71] were introduced as measures of fidelity and diversity. Nevertheless, the authors in [72] noted that these two evaluation techniques are unsuitable for practical application because they fail to fulfill the essential criteria for effective evaluation metrics.

To overcome the shortcomings of the FID in distinguishing between distinct aspects of generative quality, fidelity, and diversity, the authors in [70] presented the *density and coverage metrics*, specifically designed to provide a more detailed assessment of generative performance. By incorporating a simple yet carefully designed manifold estimation procedure, this approach enhances the empirical reliability of fidelity-diversity metrics and provides a foundation for theoretical analysis. Table 3 presents a concise overview of the listed metrics, whereas the full details of the density and coverage metric are omitted for brevity.

We assume that it is possible to draw samples $\{X_i\}$ from a real distribution $P(X)$ and $\{Y_j\}$ from a generative model $Q(Y)$, respectively. Let $B(x, r)$ be the sphere in $\mathbb{R}^D$ around x with radius r and let N and M be the number of real and fake samples. The manifold $\mathcal{M}(X_1, \ldots, X_N) := \bigcup_{i=1}^{N} B(X_i, \mathrm{NND}_k(X_i))$, where $\mathrm{NND}_k(X_i)$ denotes the distance from X_i to the kth nearest neighbor among $\{X_i\}$, excluding itself.

Metric	Mathematical expression	Explanation
Precision [71]	$\frac{1}{M}\sum_{j=1}^{M} 1_{Y_j \in \mathcal{M}(X_1,\ldots,X_N)}$	Measures the proportion of generated samples Y_j falling in the real data manifold $\mathcal{M}(X_1,\ldots,X_N)$. Higher precision means generated samples are realistic but do not guarantee diversity.
Recall [71]	$\frac{1}{N}\sum_{i=1}^{N} 1_{X_i \in \mathcal{M}(Y_1,\ldots,Y_M)}$	Evaluates how well the generator covers the real data distribution by checking whether real samples X_i lie in the generated data manifold $\mathcal{M}(Y_1,\ldots,Y_M)$. Higher recall implies better diversity but does not ensure realism.
Density [72]	$\frac{1}{kM}\sum_{j=1}^{M}\sum_{i=1}^{N} 1_{Y_j \in B(X_i,\mathrm{NND}_k(X_i))}$	Quantifies how densely generated samples Y_j populate the real data space by counting how many real-sample neighborhoods contain Y_j. Higher density indicates greater sample concentration.
Coverage [72]	$\frac{1}{N}\sum_{i=1}^{N} 1_{\exists j \text{ s.t. } Y_j \in B(X_i,\mathrm{NND}_k(X_i))}$	Measures the proportion of real samples X_i that have at least one generated sample Y_j in their neighborhood. Unlike recall, coverage is less sensitive to outliers, offering a more robust assessment of generative diversity.

Table 3: Comparison of generative model evaluation metrics.

When evaluating generative models, the range of values that density and coverage can take must be considered. The density is not upper bound by 1 because it considers multiple generated samples per the real data point. A higher density indicates a greater concentration of generated samples. Coverage is not normalized, but higher values are better, meaning the generated data distribution better represents the actual data. Cov-

erage is bounded between 0 and 1, and the maximum value of 1 indicates that every real sample has at least one generated counterpart in its neighborhood. (See [71, 72] and the references therein for a comprehensive understanding of these metrics, including their mathematical definitions and implications.)

Experimental results We downloaded the corresponding protein chain from `http://www.pdb.org` and extracted the 3D coordinate information of α-carbon (C_α) atoms for each protein. The number of amino acids considered in the distance matrix is a critical parameter in experiments and influences the resolution and effectiveness of structural modeling. For computational efficiency, we consider only the first 32 amino acids. The dataset contains $60,000$ proteins and is divided into training (80%) and testing (20%) sets. During each experiment, 12,000 samples were sampled to compute the metrics with the testing set. The models were evaluated based on *density and coverage* and the *FID*, indicated with arrows ↑ implying that higher values are better. The best results arein bold.

Importance of modeling This section presents quantitative comparisons of model performance under various Hurst parameters $H = (0.2, 0.5, 0.8)$ and noise schedules (linear and cosine).

First, we evaluated the generative performance of the FVP model with various selections of K (the number of OU processes to approximate fBm) under a linear noise schedule and compared it with the standard VP-SDE baseline. For all models, we employed the classical Euler–Maruyama scheme to solve the corresponding reverse-time SDE numerically, ensuring consistency across experiments. Table 4 summarizes the model performance across configurations.

	$H = 0.2$			$H = 0.5$			$H = 0.8$		
FVP (**Linear**)	Density ↑	Coverage ↑	FID ↓	Density ↑	Coverage ↑	FID ↓	Density ↑	Coverage ↑	FID ↓
VP (baseline)	-	-	-	1.043	0.884	75.368	-	-	-
$K = 2$	0.942	0.858	77.219	0.965	0.896	75.508	1.0142	0.922	74.899
$K = 3$	0.932	0.840	77.174	1.028	0.894	75.934	**1.118**	**0.934**	**74.614**

Table 4: Quantitative results for fractional variance-preserving dynamics with various H and linear noise schedules.

The results reveal that increasing the Hurst parameter H generally improves performance, with higher density and coverage and a lower FID at $H = 0.8$. The best overall model is $H = 0.8$, $K = 3$, achieving the highest density (1.118) and coverage (0.934) and the lowest FID (74.614), indicating superior generative quality. Increasing K slightly enhances coverage and the FID but has a minimal effect on the density. Compared to the VP baseline ($H = 0.5$), the FVP models perform better at $H = 0.8$, highlighting the advantage of fractional modeling in capturing complex data distributions.

We generate bar charts for the linear noise schedule, separately visualizing the density, coverage, and FID across Hurst parameter H values. Each chart presents the results for $K = 2$ and $K = 3$ to facilitate a comparative analysis of their influence on model performance.

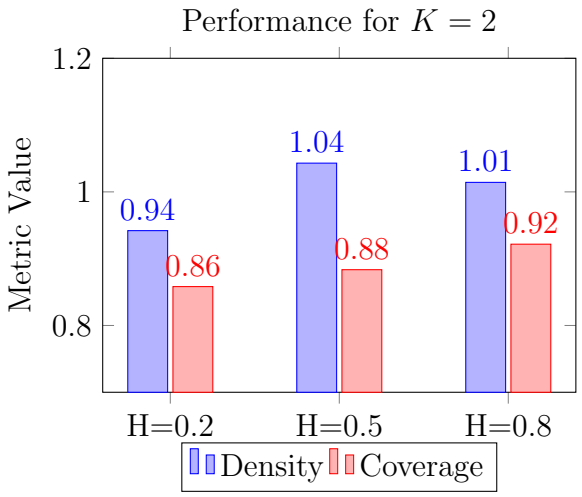

Figure 4: Quantitative results for $K = 2$

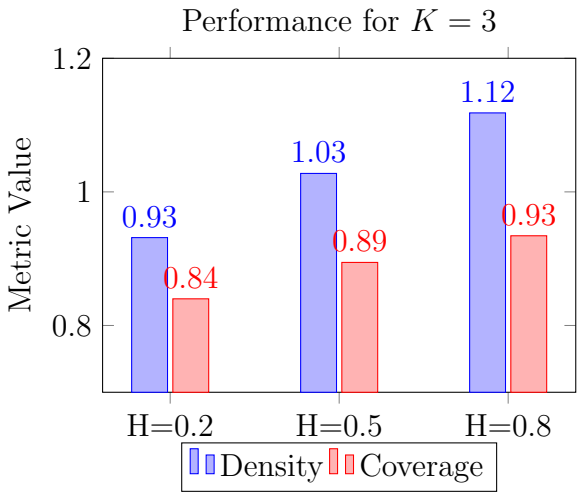

Figure 5: Quantitative results for $K = 3$

Table 5 presents the FID bar plots for the linear case, separated for $K = 2$ and $K = 3$. Table 5 compares model performance under the cosine noise schedule and Hurst parameters $H = (0.2, 0.5, 0.8)$.

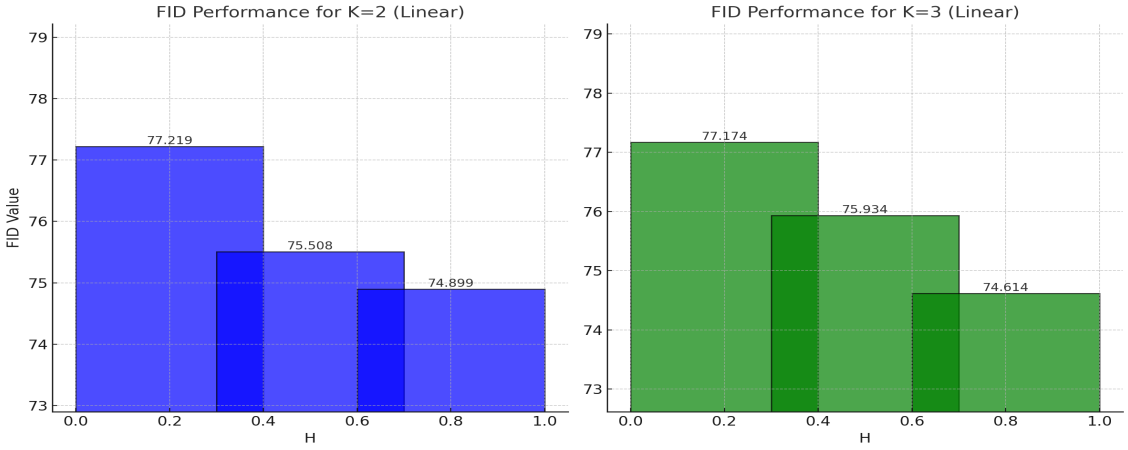

Figure 6: Comparison of the Fréchet inception distance (FID) performance

	$H = 0.2$			$H = 0.5$			$H = 0.8$		
FVP (**Cosine**)	Density ↑	Coverage ↑	FID ↓	Density ↑	Coverage ↑	FID ↓	Density ↑	Coverage ↑	FID ↓
$K = 2$	0.960	0.936	74.367	0.799	0.853	78.907	**1.04**	**0.954**	75.160
$K = 3$	0.960	0.923	**73.452**	0.832	0.861	79.305	1.011	0.941	75.390

Table 5: Quantitative results for the fractional variance-preserving dynamics with various H and cosine noise schedules.

The results indicate that higher Hurst parameters H improve diversity, as observed in increased coverage, but do not always yield a lower FID. The best FID is achieved at

$H = 0.2$ and $K = 3$ (FID = 73.452), suggesting optimal generative quality under the cosine schedule. Moreover, $H = 0.8$ and $K = 2$ (density = 1.04, coverage = 0.954) offer the best balance between fidelity and diversity.

Figure 7 presents line plots comparing density, coverage, and the FID across Hurst parameters H for linear and cosine noise schedules, with separate curves for $K = 2$ and $K = 3$. The objective is to analyze how the choice of noise schedule and order K affects model performance across generative quality metrics.

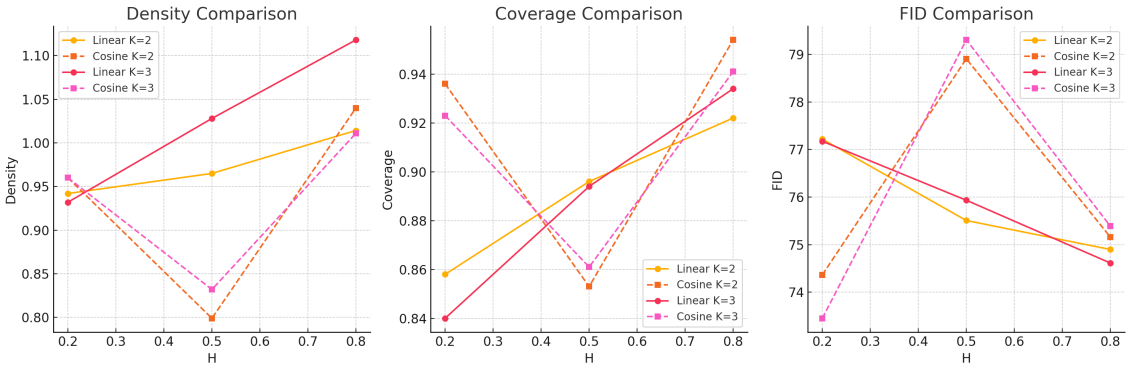

Figure 7: Comparison of density, coverage, and the Fréchet inception distance (FID) for linear and cosine noise schedules.

Figure 7 reveals that the cosine schedule performs better at lower H values, achieving the best FID at $H = 0.2$ and $K = 3$, suggesting it preserves fine details early in the diffusion process. In contrast, the linear schedule outperforms others at $H = 0.8$, yielding higher density and coverage, indicating better long-range diversity. The FID peaks at $H = 0.5$ for both schedules, implying that BM introduces slight degradation in generative quality. Across all settings, density and coverage increase with H, whereas the choice for the noise schedule influences the balance between the sample fidelity and diversity rather than consistently improving performance.

Importance of sampling methods This section evaluates the influence of various SDE and ODE solvers on generative performance. The number of score function evaluations (NFEs) is employed to assess computational efficiency, whereas density, coverage, and the FID measure generative quality. We examined solver performance with 1,000 and 2,000 discretization steps, analyzing how the step count influences sample fidelity and diversity.

First, we considered the SDE solvers, using the classical Euler ODE as a baseline, along with the Euler–Maruyama method and PC sampler. We applied the reverse diffusion SDE as the predictor and Langevin dynamics as the corrector. Table 6 presents the results.

SDE Solver									
	Euler (baseline)			Euler-Maruyama			PC sampler		
Iteration	Density ↑	Coverage ↑	FID ↓	Density ↑	Coverage ↑	FID ↓	Density ↑	Coverage ↑	FID ↓
1000	0.908	0.847	74.395	1.118	0.934	74.614	**1.224**	**0.965**	**74.432**
2000	0.907	0.859	74.490	1.035	0.949	75.003	1.17	0.938	74.884

Table 6: **Density and coverage** and the **Fréchet inception distance (FID)** for stochastic differential equation (SDE) solvers.

The classical Euler ODE baseline provides a reference point but lacks the stochastic flexibility for optimal generative modeling. The PC sampler at 1,000 steps performs best overall, with the highest density (1.224) and coverage (0.965) and the lowest FID (74.432), indicating superior generative quality. Although increasing the discretization steps to 2,000 marginally improves the coverage, it does not consistently enhance the density or FID, particularly for the PC sampler, where a slight performance degradation occurs. The Euler–Maruyama method remains computationally efficient but is outperformed by the PC method, which yields better sample diversity and fidelity at a similar computational cost. Figure 8 displays the comparative bar plots for the density, coverage, and FID metrics across SDE solvers (Euler, Euler–Maruyama, and PC sampler) at 1,000 and 2,000 iterations.

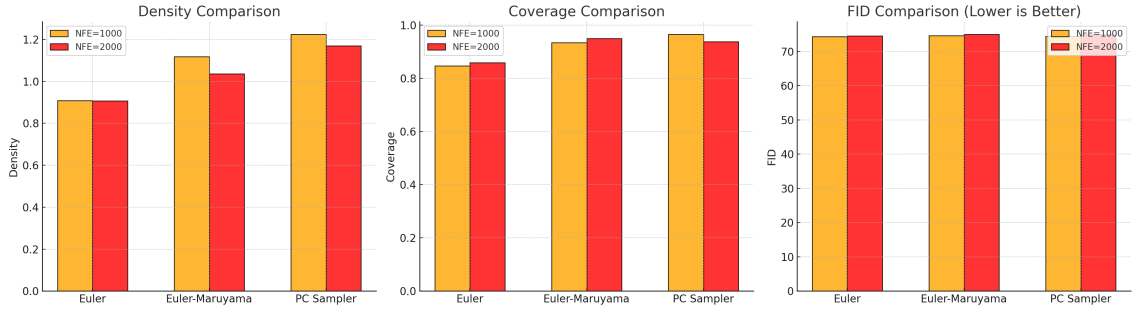

Figure 8: Comparison of density, coverage, and Fréchet inception distance (FID) metrics for the Euler ordinary differential equation solver and two stochastic differential equation solvers at two iteration counts.

After examining the influence of various SDE solvers, this work focuses on evaluating the effects of ODE solvers in the generative modeling process. We investigated how several classical ODE solvers influence sample quality, diversity, and computational efficiency. This experiment considers three numerical solvers. The Euler method provides a reference for comparison. The commonly applied fixed-step solver, fourth-order RK (RK4), is known for improved accuracy, and the adaptive-step solver RK 45 (RK45) dynamically adjusts the step size for better precision and efficiency. Table 7 summarizes the quantitative results.

Classical ODE Solvers									
	Euler (baseline)			4th order Runge–Kutta (RK4)			RK45		
Iteration	Density ↑	Coverage ↑	FID ↓	Density ↑	Coverage ↑	FID ↓	Density ↑	Coverage ↑	FID ↓
1000	0.908	0.847	74.395	0.83	0.75	77.9	**1.00**	**0.96**	**73.8**
2000	0.907	0.859	74.490	0.886	0.862	74.3	0.973	0.89	74.2

Table 7: **Density and coverage** and the **Fréchet inception distance** for ordinary differential equation solvers.

Table 7 reveals that the RK45 solver at 1,000 steps performs best, achieving the highest density (1.00) and coverage (0.96) and the lowest FID (73.8), indicating superior sample fidelity and diversity. Although RK4 displays degraded FID performance, the Euler method remains a computationally simple alternative but is outperformed by RK45. Increasing discretization steps marginally improves coverage but does not significantly enhance the FID, suggesting that adaptive solvers, such as RK45, balance efficiency and accuracy better than fixed-step methods. The bar plots in Figure 9 compare the density, coverage, and FID metrics for ODE solvers (Euler, RK4, and RK45) evaluated at 1,000 and 2,000 iterations.

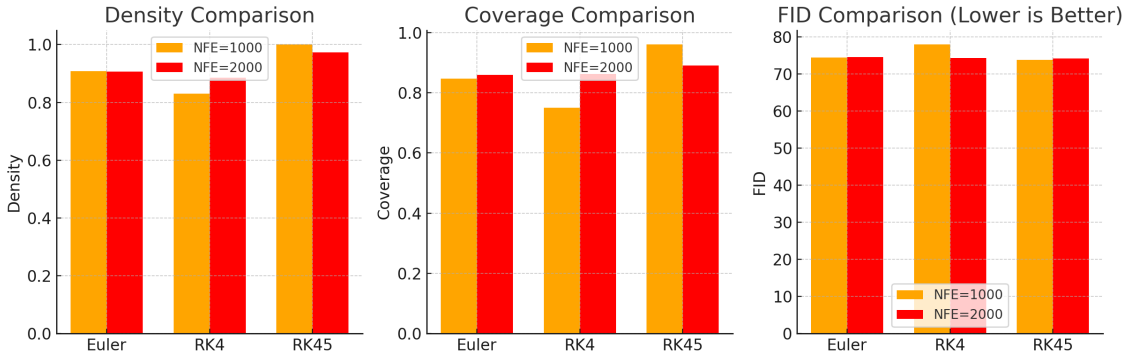

Figure 9: Comparison of density, coverage, and the Fréchet inception distance (FID) metrics for ordinary differential equation solvers at two iteration counts.

6 Conclusion and Future work

Conclusion This work proposes ProT-GFDM, a novel generative framework for protein generation. This model employs fractional stochastic dynamics to model long-range dependencies in protein backbone structures more effectively. The detailed analysis highlights the importance of modeling choices, demonstrating that the Hurst parameter, noise scheduling, and solver selection factors significantly influence generative performance.

The quantitative results in Tables 4 and 5 highlight the Hurst parameter H and the number of OU processes K as critical modeling factors that significantly influence generative performance. Based on the results in those tables, higher H generally improves density and coverage in both noise schedules, suggesting that higher Hurst parameters

allow the model to capture long-range dependencies better. The FID trends vary, with a lower FID observed at $H = 0.2$ for the cosine schedule (best FID = 73.452, $K = 3$), whereas, at $H = 0.8$, the linear schedule achieves a lower FID (best FID = 74.614, $K = 3$), implying that noise scheduling influences generative quality differently across Hurst values. Increasing K from 2 to 3 results in marginal improvements in coverage and the FID but does not significantly affect density, indicating that additional OU processes may slightly refine the sample diversity but do not drastically alter generative performance.

Most previous literature studies have focused on experiments with linear noise schedules, whereas this work explores an alternative nonlinear noise schedule, the cosine noise schedule, to demonstrate how the choice of noise schedule affects model performance. The cosine noise schedule is introduced to explore its potential practical benefits, particularly regarding performance enhancement and stability. However, as indicated by the comparison between linear (Table 4) and cosine (Table 5) noise schedules, changing the noise schedule does not consistently improve performance across all metrics . Still, it does influence the trade-off between fidelity and diversity. The cosine schedule achieves the lowest FID at $H = 0.2$ and $K = 3$ (FID = 73.452), suggesting better generative quality in some cases. In contrast, the linear schedule yields higher density and Coverage at $H = 0.8, K = 3$ (Density = 1.118, Coverage = 0.934), favoring sample diversity. The cosine schedule tends to preserve more structure in early diffusion steps but then applies noise more aggressively in later stages. This may lead to suboptimal score-matching behavior, where the model struggles to accurately learn denoising at high noise levels. Some samplers may be better suited for linear noise decay, while cosine-based schedules may require a more specialized adaptive step-size or fine-tuned solver. The interaction between the choice of noise schedule and the sampler remains unclear. There might be other potential factors that could explain this outcome. A very recent paper [73] challenges the prevailing assumption that noise conditioning is essential for the success of denoising diffusion models. Their findings reveal that most models degrade gracefully without noise conditioning, and some even perform better. This study encourages the research community to reconsider the foundational assumptions and formulations of denoising generative models.

Our findings from Table 6 and Table 7 underscore the significant impact of sampling methods on generative performance. Comparing SDE and ODE solvers, we observe that the Predictor-Corrector (PC) Sampler outperforms Euler-based methods, achieving the highest Density (1.224), Coverage (0.965), and lowest FID (74.432) at 1000 iterations. This suggests that correcting steps via Langevin dynamics enhances sample fidelity and diversity. For ODE solvers, the adaptive RK45 solver demonstrates superior performance, reaching the best FID (73.8) at 1000 iterations, outperforming Euler and RK4. While increasing the number of iterations (2000 steps) marginally improves Coverage, it does not consistently enhance FID, indicating that an increase in the discretization steps does not always improve performance, with PC Sampler and RK45 slightly degrading at 2000 steps, possibly due to overcorrection effects. In practice, selecting an appropriate solver for the ScoreSDE model requires balancing computational cost (measured by the NFEs) with sample quality, as reflected by density, coverage, and FID. However, the theoretical understanding of the qualitative differences between sampling from the SDE and the ODE

remains an open question, highlighting the need for further analysis.

Future work For the driving process, the choice of the number of OU processes, K, used to approximate the fBm depends on the data. There is a trade-off: a higher K provides a more accurate approximation of fBm but comes at a higher computational cost. Regarding the Hurst index, we use the classical case $H = \frac{1}{2}$, while for super-diffusion ($H > \frac{1}{2}$), we set $H = 0.8$, and for sub-diffusion ($H < \frac{1}{2}$), we set $H = 0.2$. The selection of the Hurst index in these experiments is based on practical considerations, as the true value is unknown. A more precise estimation of the Hurst index could be directly derived from the data [28].

More advanced ODE solvers can also be considered. Recall the general form of Diffusion ODE

$$d\mathbf{x}_t = \left\{ f(t)\mathbf{x}_t - \frac{1}{2} g^2(t)\mathbf{s}_\theta(\mathbf{x}_t, t) \right\} dt. \tag{21}$$

The model is characterized by a **semi-linear structure**, comprising two main components: a linear function of the data variable $\mathbf{x}_t$ and a nonlinear function of $\mathbf{x}_t$ parameterized by neural networks $s_\theta(\hat{\mathbf{x}}, t)$. The solution at time t can be exactly formulated by the "variation of constants" formula:

$$\mathbf{x}_t = e^{\int_s^t f(\tau)\, d\tau} \mathbf{x}_s + \int_s^t \left(e^{\int_\tau^t f(r)\, dr} \frac{1}{2} g^2(\tau) \mathbf{s}_\theta(\mathbf{x}_\tau, \tau) \right) d\tau.$$

We can consider DPM-Solver [66] and DEIS (Diffusion Exponential Integrator Sampler) [67], which aim to solve the PF-ODE in diffusion models efficiently. They share a key similarity: both exploit the semi-linear structure of the PF-ODE to improve efficiency and accuracy. Lu et al. [74] introduced an improved version of the DPM solver with *DPM-solver++*, a high-order solver for the guided sampling of DPMs, to further speed up high-quality sample generation. For DEIS, the authors of [75] proposed a modified version of DEIS, called the *DEIS-SN*, based on a simple new score parametrization – to normalize the score estimate using its average empirical absolute value at each timestep (computed from high NFE offline generations). This leads to consistent improvements in FID compared to classical DEIS.

Acknowledgements The authors gratefully acknowledge the support of the Artificial Intelligence for Design Challenge program from the National Research Council Canada for funding this project.

7 Notational conventions

Definitions of the mathematical symbols used in this work.

Symbol	Definition
$[0, T]$	Time horizon with terminal time $T > 0$
$X = (X_t)_{t\in[0,T]}$	Stochastic forward process taking values in $\mathbb{R}$
$D \in \mathbb{N}$	Data dimension
$\mathbf{X}$	Vector-valued stochastic forward process $\mathbf{X} = (X_t)_{t\in[0,T]}$, $X_t = (X_{t,1}, \dots, X_{t,D})$
f	Function $f : \mathbb{R}^D \times [0, T] \to \mathbb{R}^D$
μ, g	Functions $\mu, g : [0, T] \to \mathbb{R}$
p_0	Data distribution
p_t	Marginal density of (augmented) forward process at $t \in [0, T]$
B	Brownian motion (BM)
H	Hurst index $H \in (0, 1)$
W^H	Type I fractional Brownian motion (fBm)
B^H	Type II fractional Brownian motion (fBm)
$Y^\gamma = (Y^\gamma_t)_{t\in[0,T]}$	Ornstein–Uhlenbeck (OU) process with speed of mean reversion $\gamma \in \mathbb{R}$
$K \in \mathbb{N}$	Number of approximating processes
$\gamma_1, \dots, \gamma_K$	Geometrically spaced grid
$\omega_1, \dots, \omega_K$	Approximation coefficients
ω	Optimal approximation coefficients $\omega = (\omega_1, \dots, \omega_K)$
$\tilde{B}^H$	Markov-approximate fractional Brownian motion (MA-fBm)
k	$k \in \mathbb{N}$ with $1 \le k \le K$
Y^k	OU processes $Y^k = Y^{\gamma^k}$
$Y^1, \dots, Y^K$	Augmenting processes with $Y^k = (Y^k, \dots, Y^K)$
$\mathbf{F}, \mathbf{G}$	Vector-valued functions $\mathbf{F}, \mathbf{G} : [0, T] \to \mathbb{R}^{D\cdot(K+1)}$
$\mathbf{Z}$	By $Y^1, \dots, Y^K$ augmented forward process
$Y^{[K]}$	Stacked vector of augmenting processes
q_t	Marginal density of $Y^{[K]}$ at $t \in [0, T]$
θ	Weight vector of a neural network

References

[1] Morris R, Black KA, Stollar EJ. Uncovering protein function: from classification to complexes. Essays Biochem. 2022;66:255–85.

[2] Luo Y. Sensing the shape of functional proteins with topology. Nat Comput Sci. 2023;3(2):124–5.

[3] Huang PS, Boyken SE, Baker D. The coming of age of de novo protein design. Nature. 2016;537(7620):320.

[4] Pan X, Kortemme T. Recent advances in de novo protein design: principles, methods, and applications. J Biol Chem 2021;296:100558. https://doi.org/10.1016/j.jbc.2021.100558.

[5] Lai B. Leveraging deep generative model for computational protein design and optimization. arXiv preprint arXiv:2408.17241, 2024. https://doi.org/10.48550/arXiv.2408.17241.

[6] Lin B, Luo X, Liu Y, Jin X. A comprehensive review and comparison of existing computational methods for protein function prediction. Brief Bioinform 2024;25:bbae289. https://doi.org/10.1093/bib/bbae289.

[7] Goodfellow I, Pouget-Abadie J, Mirza M, et al. Generative adversarial nets. Adv Neural Inf Process Syst. 2014;27.

[8] Kingma DP, Welling M. Auto-encoding variational bayes. arXiv preprint. 2013;arXiv:1312.6114.

[9] Rezende D, Mohamed S. Variational inference with normalizing flows. International Conference on Machine Learning. PMLR; 2015. p. 1530–8.

[10] Anand N, Huang P. Generative modeling for protein structures. In: Bengio S, et al., editors. Advances in Neural Information Processing Systems. Vol. 31. Curran Associates; 2018.

[11] Lyu S, Sowlati-Hashjin S, Garton M. ProteinVAE: Variational autoencoder for translational protein design. bioRxiv. 2023.

[12] Elnaggar A, Heinzinger M, Dallago C, et al. ProtTrans: Toward understanding the language of life through self-supervised learning. IEEE Trans Pattern Anal Mach Intell. 2021;44:7112–27.

[13] Sevgen E, Moller J, Lange A, et al. Prot-VAE: Protein transformer variational autoencoder for functional protein design. bioRxiv. 2023.

[14] Repecka D, Jauniskis V, Karpus L, et al. Expanding functional protein sequence spaces using generative adversarial networks. Nat Mach Intell. 2021;3:324–33. https://doi.org/10.1038/s42256-021-00310-5.

[15] Strokach A, Kim PM. Deep generative modeling for protein design. Curr Opin Struct Biol. 2022;72:226–36. https://doi.org/10.1016/j.sbi.2021.11.008.

[16] Wang J, Cao H, Zhang JZ, Qi Y. Computational protein design with deep learning neural networks. Sci Rep. 2018;8(1):1–9.

[17] Qi Y, Zhang JZ. DenseCPD: improving the accuracy of neural-network-based computational protein sequence design with DenseNet. J Chem Inf Model. 2020;60(3):1245–52.

[18] Anishchenko I, et al. De novo protein design by deep network hallucination. Nature. 2021;600:547–52.

[19] Dauparas J, et al. Robust deep learning-based protein sequence design using ProteinMPNN. Science. 2022;378:49–56.

[20] Ferruz N, Schmidt S, Höcker B. ProtGPT2 is a deep unsupervised language model for protein design. Nat Commun. 2022;13:4348.

[21] Song Y, Sohl-Dickstein J, Kingma DP, Kumar A, Ermon S, Poole B. Score-based generative modeling through stochastic differential equations. Int Conf Learn Represent. 2020.

[22] Ho J, Jain A, Abbeel P. Denoising diffusion probabilistic models. NeurIPS. 2020.

[23] Song Y, Ermon S. Generative modeling by estimating gradients of the data distribution. Adv Neural Inf Process Syst. 2019;32.

[24] Yoon EB, Park K, Kim S, Lim S. Score-based generative models with Lévy processes. Adv Neural Inf Process Syst. 2023;36:40694–707.

[25] Paquet E, Soleymani F, Viktor HL, Michalowski W. Annealed fractional Lévy–Itō diffusion models for protein generation. Comput Struct Biotechnol J. 2024.

[26] Nobis G, Aversa M, Springenberg M, Detzel M, Ermon S, Nakajima S, Murray-Smith R, Lapuschkin S, Knochenhauer C, Oala L, et al. Generative fractional diffusion models. arXiv preprint. 2023;arXiv:2310.17638.

[27] Harms P, Stefanovits D. Affine representations of fractional processes with applications in mathematical finance. Stoch Proc Appl. 2019;129(4):1185–2228. https://doi.org/10.1016/j.spa.2018.04.010.

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

[29] Nichol AQ, Dhariwal P. Improved denoising diffusion probabilistic models. Int Conf Mach Learn. 2021;8162–71.

[30] Anderson BD. Reverse-time diffusion equation models. Stochastic Process Appl. 1982;12(3):313–26.

[31] Kloeden PE, Platen E. Numerical solution of stochastic differential equations. Springer; 1992.

[32] Chen TQ, Rubanova Y, Bettencourt J, Duvenaud D. Neural ordinary differential equations. Adv Neural Inf Process Syst. 2018;31:6572–83.

[33] Altschul SF, Gish W, Miller W, et al. Basic local alignment search tool. J Mol Biol. 1990;215:403–10. https://doi.org/10.1016/S0022-2836(05)80360-2.

[34] Rifaioglu AS, Doğan T, Martin MJ, et al. DeepRED: automated protein function prediction with multi-task feed-forward deep neural networks. Sci Rep. 2019;9:7344.

[35] Kulmanov M, Hoehndorf R. DeepGOPlus: improved protein function prediction from sequence. Bioinformatics. 2020;36:422–9. https://doi.org/10.1093/bioinformatics/btz595.

[36] Gligorijević V, Renfrew PD, Kosciolek T, et al. Structure-based protein function prediction using graph convolutional networks. Nat Commun. 2021;12:3168. https://doi.org/10.1038/s41467-021-23303-9.

[37] Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput. 1997;9:1735–80. https://doi.org/10.1162/neco.1997.9.8.1735.

[38] Zhonghui G, Luo X, Chen J, et al. Hierarchical graph transformer with contrastive learning for protein function prediction. Bioinformatics. 2023;39:btad410. https://doi.org/10.1093/bioinformatics/btad410.

[39] Mostafavi S, Ray D, Warde-Farley D, et al. GeneMANIA: a real-time multiple association network integration algorithm for predicting gene function. Genome Biol. 2008;9:1–15.

[40] Gligorijević V, Barot M, Bonneau R. DeepNF: deep network fusion for protein function prediction. Bioinformatics. 2018;34:3873–81. https://doi.org/10.1093/bioinformatics/bty440.

[41] Zhou N, Jiang Y, Bergquist TR, et al. The CAFA challenge reports improved protein function prediction and new functional annotations for hundreds of genes through experimental screens. Genome Biol. 2019;20:1–23.

[42] Lin B, Luo X, Liu Y, Jin X. A comprehensive review and comparison of existing computational methods for protein function prediction. Brief Bioinform. 2024;25(4):bbae289.

[43] Mardikoraem M, Wang Z, Pascual N, Woldring D. Generative models for protein sequence modeling: recent advances and future directions. Brief Bioinform. 2023;24:bbad358.

[44] Wang J, et al. Scaffolding protein functional sites using deep learning. Science. 2022;377:387–94.

[45] Lee JS, Kim J, Kim PM. Score-based generative modeling for de novo protein design. Nat Comput Sci. 2023;3:382–92.

[46] Bernstein FC, Koetzle TF, Williams GJB, Meyer EF, Brice MD, Rodgers JR, Kennard O, Shimanouchi T, Tasumi M. The Protein Data Bank: a computer-based archival file for macromolecular structures. Arch Biochem Biophys. 1978;185(2):584–91.

[47] Anand N, Huang P. Generative modeling for protein structures. Adv Neural Inf Process Syst. 2018;31.

[48] Hamelryck T, Kent JT, Krogh A. Sampling realistic protein conformations using local structural bias. PLoS Comput Biol. 2006;2(9):e131.

[49] Boomsma W, Mardia KV, Taylor CC, Ferkinghoff-Borg J, Krogh A, Hamelryck T. A generative, probabilistic model of local protein structure. Proc Natl Acad Sci U S A. 2008;105(26):8932–7.

[50] Wu J, Zhang C, Xue T, Freeman B, Tenenbaum J. Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling. Adv Neural Inf Process Syst. 2016;82–90.

[51] Rohl CA, Strauss CE, Misura KM, Baker D. Protein structure prediction using Rosetta. Methods Enzymol. 2004;383:66–93.

[52] Boyd S, Parikh N, Chu E, Peleato B, Eckstein J, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. Found Trends Mach Learn. 2011;3(1):1–122.

[53] Anand N, Eguchi R, Huang PS. Fully differentiable full-atom protein backbone generation. ICLR 2019 Workshop DeepGenStruct. 2019. `https://openreview.net/forum?id=SJxnVL8YOV`. Accessed 6 June 2024.

[54] Shevchenko G. Fractional Brownian motion in a nutshell. Int J Mod Phys Conf Ser. 2015;36. `https://doi.org/10.1142/S2010194515600022`.

[55] Hyvärinen A, Dayan P. Estimation of non-normalized statistical models by score matching. J Mach Learn Res. 2005;6(4).

[56] Martens J, Sutskever I, Swersky K. Estimating the Hessian by backpropagating curvature. Proc 29th Int Conf Mach Learn. 2012;963–70.

[57] Song Y, Garg S, Shi J, Ermon S. Sliced score matching: A scalable approach to density and score estimation. Conf Uncertainty Artif Intell. 2019;204.

[58] Vincent P. A connection between score matching and denoising autoencoders. Neural Comput. 2011;23(7):1661–74.

[59] Song Y, Ermon S. Generative modeling by estimating gradients of the data distribution. Adv Neural Inf Process Syst. 2019;11895–907.

[60] Song Y, Ermon S. Improved techniques for training score-based generative models. Adv Neural Inf Process Syst. 2020;33.

[61] Song J, Meng C, Ermon S. Denoising diffusion implicit models. Int Conf Learn Represent. 2020.

[62] Kloeden PE, Platen E. Numerical solution of stochastic differential equations. Vol. 23. Springer Science & Business Media; 2013.

[63] Mil'shtein GN. Approximate integration of stochastic differential equations. Theory Probab Appl. 1975;19(3):557–000.

[64] Allgower EL, Georg K. Numerical continuation methods: an introduction. Vol. 13. Springer Science & Business Media; 2012.

[65] Chen S, Chewi S, Lee H, Li Y, Lu J, Salim A. The probability flow ODE is provably fast. arXiv preprint. 2023;arXiv:2305.11798.

[66] Lu C, Zhou Y, Bao F, Chen J, Li C, Zhu J. DPM-solver: A fast ODE solver for diffusion probabilistic model sampling in around 10 steps. arXiv preprint. 2022;arXiv:2206.00927.

[67] Zhang Q, Chen Y. Fast sampling of diffusion models with exponential integrator. arXiv preprint. 2022;arXiv:2204.13902.

[68] Kingma D, Salimans T, Poole B, Ho J. Variational diffusion models. Adv Neural Inf Process Syst. 2021;34:21696–707.

[69] Ronneberger O, Fischer P, Brox T. U-Net: convolutional networks for biomedical image segmentation. Int Conf Med Image Comput Comput Assist Interv. 2015;234–41.

[70] Sajjadi MS, Bachem O, Lucic M, Bousquet O, Gelly S. Assessing generative models via precision and recall. Adv Neural Inf Process Syst. 2018;31.

[71] Kynkäänniemi T, Karras T, Laine S, Lehtinen J, Aila T. Improved precision and recall metric for assessing generative models. Adv Neural Inf Process Syst. 2019;3929–38.

[72] Naeem MF, Oh SJ, Uh Y, Choi Y, Yoo J. Reliable fidelity and diversity metrics for generative models. Int Conf Mach Learn. 2020;7176–85.

[73] Sun Q, Jiang Z, Zhao H, He K. Is noise conditioning necessary for denoising generative models? arXiv preprint. 2025;arXiv:2502.13129.

[74] Lu C, Zhou Y, Bao F, Chen J, Li C, Zhu J. DPM-Solver++: fast solver for guided sampling of diffusion probabilistic models. arXiv preprint. 2022;arXiv:2211.01095.

[75] Xia G, Danier D, Das A, Fotiadis S, Nabiei F, Sengupta U, Bernacchia A. Score normalization for a faster diffusion exponential integrator sampler. NeurIPS Workshop Diffusion Models. 2023.