

# University of Oxford



DEPARTMENT OF  
**STATISTICS**

## Score-Based Diffusion Models for Protein Backbone Generation

by

Yuanhao Jiang

Wolfson College

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

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

September 2025

This is my own work (except where otherwise indicated)

Candidate: Yuanhao Jiang

Signed:.....

Date:.....

## Abstract

Diffusion models have recently emerged as a powerful class of generative models, achieving strong performance in domains such as images, audio, and molecules. This dissertation adapts a score-based diffusion framework to protein structures, representing backbones as graphs of  $C_\alpha$  atoms connected by spatial and sequential edges. Several architectures are implemented as score models, including a UNet baseline on padded sequences and Transformer-based graph neural networks with and without sinusoidal positional embeddings of amino acid indices. Experiments are conducted on the CATH S40 dataset, a benchmark of diverse protein domains. Quantitative evaluation with RMSD and TM-score shows that the Transformer-based GNN with positional embeddings consistently outperforms both the UNet baseline and the graph model without positional information. Distributional analyses and qualitative visualisations further demonstrate that positional encoding plays a critical role in maintaining sequential structure and improving overall backbone fidelity. While the models are modest in scale and focus only on  $C_\alpha$  atoms, the results suggest that score-based diffusion combined with graph representations is a viable approach to protein backbone generation and highlight the importance of incorporating sequential information in graph models.

### **Acknowledgements**

I would like to express my sincere gratitude to my supervisor, Professor George Deligiannidis, for his guidance and support throughout this dissertation. I would also like to sincerely thank my family and friends for their encouragement and patience during my studies.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Background on Diffusion Models . . . . .	1
1.2	Motivation for Protein Structure Generation . . . . .	1
1.3	Diffusion Models for Protein Structures . . . . .	1
1.4	Structure of the Dissertation . . . . .	2
<b>2</b>	<b>Score-Based Diffusion for Generative Modelling</b>	<b>3</b>
2.1	Background and Motivation . . . . .	3
2.2	Forward Diffusion Processes . . . . .	3
2.2.1	Discrete-Time Formulation . . . . .	4
2.2.2	Continuous-Time Formulation . . . . .	4
2.3	Reverse-Time Diffusion Processes . . . . .	6
2.3.1	Discrete-Time Reverse Process . . . . .	6
2.3.2	Continuous-Time Reverse SDE . . . . .	6
2.4	Score Matching and Training Objective . . . . .	7
2.4.1	Score Matching and Denoising Score Matching . . . . .	7
2.5	Sampling Procedures . . . . .	9
2.5.1	Reverse SDE Sampling via Euler-Maruyama Method . . . . .	9
2.5.2	Predictor-Corrector Sampling . . . . .	9
2.6	Noise Schedules . . . . .	9
2.6.1	Linear Schedules . . . . .	9
2.6.2	Cosine Variance-Preserving Schedule . . . . .	10
2.7	Summary . . . . .	11
<b>3</b>	<b>Protein Structure Modelling with Graph Neural Networks</b>	<b>12</b>
3.1	Protein Structure and Motivation . . . . .	12
3.2	Graph Representation of Backbones . . . . .	13
3.3	Score Models Architectures . . . . .	15
3.3.1	UNet Architecture on Padded Data . . . . .	15
3.3.2	Graph Neural Network on Graph-Structured Data . . . . .	15
3.4	Summary . . . . .	17
<b>4</b>	<b>Experiments</b>	<b>18</b>
4.1	Dataset . . . . .	19
4.2	Training and Sampling . . . . .	19
4.3	Evaluation Metrics . . . . .	20
4.4	Quantitative Results . . . . .	20
4.5	Qualitative Visualisation . . . . .	23
4.6	Summary . . . . .	23
<b>5</b>	<b>Discussion and Conclusion</b>	<b>25</b>
5.1	Discussion of Results . . . . .	25
5.2	Limitations . . . . .	25
5.3	Related Work and Future Directions . . . . .	25

5.4	Conclusion	26
<b>A</b>	<b>Code Appendix</b>	<b>27</b>

## List of Figures

1	Linear schedule vs. cosine schedule.	11
2	Schematic diagram of an amino acid and polypeptide chain from Branden and Tooze (2012). In this dissertation, we are interested only in the backbone formed by $C_\alpha$ atoms.	12
3	PyMOL visualisation of generated backbone samples from different models	18
4	Distribution of backbone lengths.	19
5	Distribution and box plot of RMSD values across test structures. Lower is better.	21
6	Distribution and box plot of TM-scores across test structures. Higher is better.	22
7	RMSD and TM-score versus backbone length $L$ .	22

## List of Tables

1	Summary statistics of backbone lengths.	19
2	Quantitative evaluation of backbone reconstruction fidelity. Values reported are mean $\pm$ standard deviation.	21

# 1 Introduction

## 1.1 Background on Diffusion Models

Diffusion models have recently emerged as one of the most powerful classes of deep generative models, achieving strong performance across a range of domains, most prominently in image generation (Ho et al., 2020; Dhariwal and Nichol, 2021). These models construct a forward process that gradually corrupts data with Gaussian noise, and train a neural network to approximate the reverse-time dynamics that iteratively remove noise. The resulting generative process is both probabilistically principled and empirically effective, offering advantages in sample quality and training stability compared to earlier paradigms such as variational autoencoders (VAEs) (Kingma and Welling, 2022) and generative adversarial networks (GANs) (Goodfellow et al., 2020).

Score-based diffusion models (Song and Ermon, 2019, 2020; Song et al., 2021b,a) extend denoising diffusion models by shifting the focus from directly learning the reverse process to estimating the score function, i.e. the gradient of the log-density of the noisy distribution, and naturally generalising the formulation to continuous time via stochastic differential equations (SDEs), allowing different forward noise processes and corresponding reverse-time samplers to be employed. The resulting framework is highly flexible and has been successfully applied to a wide range of domains, including image and audio generation, point cloud modelling, and molecular conformations.

## 1.2 Motivation for Protein Structure Generation

Proteins are essential macromolecules whose biological functions are determined by their three-dimensional structures. The ability to generate novel protein backbones has potential applications in biomedical science, including de novo protein design, drug discovery, and enzyme engineering. However, protein structures are subject to strict constraints: bond lengths and bond angles are restricted to narrow ranges, torsional angles follow characteristic distributions, and global folds must remain physically plausible (Creighton, 1993; Branden and Tooze, 2012). These factors make the generative task significantly more challenging than in more common domains such as images or text.

Deep learning has already transformed structural biology. AlphaFold (Jumper et al., 2021), for example, demonstrated the power of data-driven methods for predicting native protein structures from amino acid sequences. However, the problem of generating realistic and diverse protein structures remains comparatively underexplored.

## 1.3 Diffusion Models for Protein Structures

The probabilistic formulation of score-based diffusion models makes them a natural candidate for modelling protein backbones. Unlike GANs, diffusion models provide a likelihood-based framework, and unlike VAEs, they do not impose restrictive assumptions on the latent space. Their iterative denoising dynamics also align well with the idea of progressively refining approximate structures toward physically plausible conformations.

Recent works highlight this promise. Watson et al. (2023) introduced a diffusion-based

framework for de novo design, achieving impressive results in generating novel folds and functional proteins. Wu et al. (2024) modelled folding trajectories using diffusion, while Yim et al. (2024) applied diffusion models to protein-ligand docking, demonstrating the versatility of the paradigm in structural biology. These advances suggest that diffusion models can capture rich geometric and biophysical constraints in a principled way.

A key challenge lies in representation. In this work, we represent the protein backbone as a graph where nodes correspond to  $C_\alpha$  atoms and edges encode both spatial proximity and sequential adjacency. This formulation enables the use of graph neural networks (GNNs) (Scarselli et al., 2009; Gilmer et al., 2017) as score models, incorporating geometric features such as radial basis encodings, directional vectors, and positional embeddings. In particular, we employ Transformer-based graph convolutional layers, which extend the expressive power of message passing beyond classical GCNs (Kipf and Welling, 2017). Training such models on structural datasets allows the score function to be learned over backbone conformational space, from which new structures can be sampled by reverse diffusion.

## 1.4 Structure of the Dissertation

The remainder of this dissertation is organised as follows. Section 2 introduces the theoretical framework of score-based diffusion models and their formulation via stochastic differential equations. Section 3 presents the graph-based representation of protein backbones and the neural architectures considered in this study. Section 4 reports experimental results on the CATH S40 dataset, including quantitative metrics and qualitative visualisations. Finally, Section 5 discusses the results, outlines limitations, and highlights directions for future work.

## 2 Score-Based Diffusion for Generative Modelling

### 2.1 Background and Motivation

Generative modelling aims to learn a distribution from which new samples can be drawn that resemble those observed in a dataset. Classical approaches include variational autoencoders (VAEs) (Kingma and Welling, 2022), which maximise a variational lower bound on the likelihood, and generative adversarial networks (GANs) (Goodfellow et al., 2020), which frame generation as an adversarial game. While each of these paradigms has been successful, they also suffer from characteristic drawbacks such as blurry reconstructions (VAEs) and mode collapse (GANs).

Diffusion models represent a more recent class of generative models that address many of these limitations. The central idea is deceptively simple: one defines a forward process that gradually corrupts data with noise until the signal is destroyed, and then learns a reverse process that reconstructs data from noise. By formulating the forward process as a diffusion and training a neural network to approximate the reverse dynamics, diffusion models combine the flexibility of deep networks with the probabilistic rigour of latent-variable models.

In their original formulation, denoising diffusion probabilistic models (DDPMs) (Ho et al., 2020) defined the forward process as a discrete-time Gaussian Markov chain. Subsequent work by Song et al. (2021b) showed that diffusion models can be generalised to continuous-time stochastic differential equations (SDEs). In this framework, the key object is the score function, i.e. the gradient of the log-density of the noisy distribution. Learning this score function allows one to simulate the reverse SDE and thereby generate new samples.

This chapter develops the theoretical foundations of score-based diffusion models in detail. We begin with the definition of the forward diffusion process, proceed to the derivation of the reverse-time dynamics, introduce score matching as the training objective, and describe practical sampling algorithms and noise schedules. Together, these elements form the mathematical basis for applying diffusion models to protein backbone generation in later chapters.

### 2.2 Forward Diffusion Processes

The first component of a diffusion generative model is the forward, or noising, process. This process gradually perturbs the data distribution into a tractable prior distribution, typically a standard Gaussian. The key requirement is that this transformation be both easy to sample from and analytically tractable, so that training objectives can be computed efficiently.

### 2.2.1 Discrete-Time Formulation

Denote clean data samples by  $x_0 \sim p_{\text{data}}(x)$ . Denoising diffusion probabilistic models (DDPMs) (Ho et al., 2020) define the forward process as a Markov chain of  $N$  steps:

$$q(x_{1:N} | x_0) := \prod_{i=1}^N q(x_i | x_{i-1}),$$

where each transition adds a small amount of Gaussian noise,

$$q(x_i | x_{i-1}) := \mathcal{N}\left(x_i; \sqrt{1 - \beta_i} x_{i-1}, \beta_i I\right), \quad i = 1, \dots, N. \quad (1)$$

This produces a sequence of progressively noisier variables  $\{x_i\}_{i=1}^N$ . Here,  $\{\beta_i\}_{i=1}^N$  is a variance schedule with  $\beta_i \in (0, 1)$  controlling the noise injected at each step.

A key property of the Gaussian forward process is that one can sample  $x_t$  at any arbitrary time step directly from the data  $x_0$ :

$$q(x_i | x_0) = \mathcal{N}\left(x_i; \sqrt{\bar{\alpha}_i} x_0, (1 - \bar{\alpha}_i)I\right), \quad (2)$$

where  $\alpha_i = 1 - \beta_i$  and  $\bar{\alpha}_i = \prod_{s=1}^i \alpha_s$ . This closed form is crucial for training, as it allows drawing noisy samples  $x_t$  without explicitly simulating all intermediate steps.

### 2.2.2 Continuous-Time Formulation

Song et al. (2021b) generalised the discrete diffusion process to continuous time by taking the limit as  $N \rightarrow \infty$  and  $\beta_i \rightarrow 0$ . In this setting, the forward process can be expressed as a stochastic differential equation (SDE) of the form

$$dx = f(x, t) dt + g(t) dw, \quad (3)$$

where  $w$  is the Brownian motion,  $f(x, t)$  is a drift term, and  $g(t)$  controls the diffusion magnitude. The SDE produces a trajectory of continuously noisier variables  $\{x(t)\}_{t \in [0, T]}$ .

For DDPM Gaussian transition kernel eq. (1), the chain is equivalent to

$$x_i = \sqrt{1 - \beta_i} x_{i-1} + \sqrt{\beta_i} z_{i-1}, \quad i = 1, \dots, N,$$

where  $z_{i-1} \sim \mathcal{N}(0, I)$ . Define functions  $\beta(\frac{i}{N}) := N\beta_i$ ,  $x(\frac{i}{N}) = x_i$  and  $z(\frac{i}{N}) = z_i$ , stepsize  $\Delta t := 1/N$  and finally reparameterise the time as  $t = i\Delta t \in \{0, \Delta t, \dots, N\Delta t = 1\}$ , we have

$$\begin{aligned} x(t) &= \sqrt{1 - \beta(t) \Delta t} x(t - \Delta t) + \sqrt{\beta(t) \Delta t} z(t - \Delta t) \\ &\stackrel{\Delta t \ll 1}{\approx} \left(1 - \frac{1}{2}\beta(t) \Delta t\right) x(t - \Delta t) + \sqrt{\beta(t) \Delta t} z(t - \Delta t), \end{aligned}$$

that is

$$x(t) - x(t - \Delta t) = -\frac{1}{2}\beta(t) \Delta t x(t - \Delta t) + \sqrt{\beta(t) \Delta t} z(t - \Delta t).$$

In the limit of  $\Delta t \rightarrow 0$ , the above equation converges to

$$dx = -\frac{1}{2}\beta(t)xdt + \sqrt{\beta(t)}dw. \quad (4)$$

Equation (4) is referred to as the Variance-preserving (VP) SDE, where  $\beta(t)$  is the noise schedule (e.g. linear, cosine). Just like the discrete case of DDPM, VP SDE has a closed form solution. Define

$$u(x(t), t) = \exp\left(\frac{1}{2} \int_0^t \beta(s) ds\right) x(t).$$

Then by Itô's chain rule,  $y(t) := u(x(t), t)$  has the SDE

$$\begin{aligned} dy &= \frac{1}{2}\beta(t)\exp\left(\frac{1}{2} \int_0^t \beta(s) ds\right)x(t)dt + \exp\left(\frac{1}{2} \int_0^t \beta(s) ds\right)dx \\ &= \exp\left(\frac{1}{2} \int_0^t \beta(s) ds\right)\left(\frac{1}{2}\beta(t)x(t)dt - \frac{1}{2}\beta(t)x(t)dt + \sqrt{\beta(t)}dw\right) \\ &= \exp\left(\frac{1}{2} \int_0^t \beta(s) ds\right)\sqrt{\beta(t)}dw. \end{aligned}$$

Integrate from 0 to  $t$  yields

$$y(t) = y(0) + \int_0^t \exp\left(\frac{1}{2} \int_0^s \beta(u) du\right) \sqrt{\beta(s)} dw_s.$$

Now undo the change of variables we have

$$\begin{aligned} x(t) &= \left[ \exp\left(\frac{1}{2} \int_0^t \beta(s) ds\right) \right]^{-1} y(t) \\ &= \exp\left(-\frac{1}{2} \int_0^t \beta(s) ds\right) x(0) + \int_0^t \exp\left(-\frac{1}{2} \int_s^t \beta(u) du\right) \sqrt{\beta(s)} dw_s. \end{aligned}$$

The initial condition  $x(0) = x_0$  is a sample from dataset, i.e., a constant, so the posterior of perturbation (or the prior of the reverse sampling) is a Gaussian distribution with mean

$$\mathbb{E}[x(t) | x(0) = x_0] = \exp\left(-\frac{1}{2} \int_0^t \beta(s) ds\right) x_0 \quad (5)$$

and variance

$$\begin{aligned} \text{Var}[x(t) | x(0) = x_0] &= \int_0^t \exp\left(-\int_s^t \beta(u) du\right) \beta(s) ds I \\ &= \left(1 - \exp\left(-\int_0^t \beta(s) ds\right)\right) I \end{aligned} \quad (6)$$

that is

$$p(x_t | x_0) = \mathcal{N}\left(x_t : x_0 e^{-\frac{1}{2} \int_0^t \beta(s) ds}, \left(1 - e^{-\int_0^t \beta(s) ds}\right) I\right). \quad (7)$$

Obviously the variance is always bounded, hence the name variance-preserving. This continuous formulation provides a flexible mathematical foundation that unifies discrete DDPMs with stochastic differential equations, and it allows the use of stochastic calculus tools to analyse and manipulate diffusion models. In particular, it enables the derivation of the reverse-time SDE, which defines the generative process and will be introduced in the next section.

Other choices of  $f$  and  $g$  in eq. (3) yield different diffusion processes, Song et al. (2021b) shown three categories,

- **Variance-preserving (VP) SDE:** maintains the marginal variance of  $x_t$  at one throughout the process.
- **Variance-exploding (VE) SDE:** variance grows unbounded as  $t \rightarrow T$ , pushing the distribution to white noise.
- **Sub-variance-preserving (sub-VP) SDE:** an interpolation between VP and VE.

We will focus on VP SDE in this dissertation.

## 2.3 Reverse-Time Diffusion Processes

The forward diffusion process, whether in discrete or continuous form, progressively perturbs data until its distribution approaches a simple prior such as an isotropic Gaussian. To perform generative modelling, we require a process that inverts this corruption: starting from noise and evolving back toward the data distribution. This is formalised through the theory of time-reversal for diffusion processes.

### 2.3.1 Discrete-Time Reverse Process

In the discrete DDPM formulation Ho et al. (2020), the generative model is defined as a reverse Markov chain

$$p_\theta(x_{0:N}) := p(x_N) \prod_{i=1}^N p_\theta(x_{i-1} | x_i),$$

with  $p(x_N) = \mathcal{N}(x_N; 0, I)$  as the prior. The reverse conditionals are modelled as Gaussians

$$p_\theta(x_{i-1} | x_i) := \mathcal{N}(x_{i-1}; \mu_\theta(x_i, i), \Sigma_\theta(x_i, i)),$$

where  $\mu_\theta$  and  $\Sigma_\theta$  are learned using a neural network with parameter  $\theta$ . Training then consists of minimising the Kullback-Leibler divergence between the true posterior  $q(x_{i-1} | x_i, x_0)$  and the model distribution  $p_\theta(x_{i-1} | x_i)$ .

### 2.3.2 Continuous-Time Reverse SDE

In the continuous-time setting, Anderson (1982) established that the time reversal of an Itô SDE (3) is itself an SDE with modified drift term,

$$dx = [f(x, t) - g^2(t) \nabla_x \log p_t(x)] dt + g(t) d\bar{w}, \quad (8)$$

where  $\bar{w}$  is the Brownian motion running backward in time,  $\nabla_x \log p_t(x)$  is the score function at time  $t$ , that is, the gradient of the log-density of the perturbed, marginal, data distribution  $p_t(x)$ . This equation defines the reverse-time SDE. It depends explicitly on the score function  $\nabla_x \log p_t(x)$  of the forward process marginal distribution  $p_t(x)$ .

For VP SDE eq. (4), the reverse-time SDE is given by

$$dx = \left[ -\frac{1}{2}\beta(t)x - \beta(t)\nabla_x \log p_t(x) \right] dt + \sqrt{\beta(t)}d\bar{w}.$$

Though it has no closed form solution, if one can estimate the score function accurately for all times  $t$ , it becomes possible to simulate the reverse SDE and generate samples from the data distribution.

This result provides the key insight underlying score-based diffusion: generative modelling reduces to score estimation. Rather than directly modelling likelihoods or sampling distributions, we train a neural network  $s_\theta(x, t)$  to approximate the score  $\nabla_x \log p_t(x)$ . The network is then used to guide the drift of the reverse SDE, producing realistic samples when integrated from Gaussian noise at  $t = 1$  back to  $t = 0$ .

## 2.4 Score Matching and Training Objective

The reverse-time SDE depends on the score function  $\nabla_x \log p_t(x)$ , which is generally intractable. Thus, the central learning problem in score-based diffusion is to approximate this score with a neural network  $s_\theta(x, t)$ , called time dependent score network (TDSN) (Song et al., 2021b). Training requires a loss function that encourages  $s_\theta$  to match the true score across noise levels.

### 2.4.1 Score Matching and Denoising Score Matching

For each time  $t$ , the score function of  $x_t$  can be estimated by minimising the Fisher divergence between the parameterised model and the true score:

$$J_t(\theta) = \mathbb{E}_{p(x_t)} \left[ \| s_\theta(x_t, t) - \nabla_{x_t} \log p(x_t) \|_2^2 \right]. \quad (9)$$

Since the true score  $\nabla_{x_t} \log p(x_t)$  is unknown, Hyvärinen (2005) proposed score matching,

$$J_t^{\text{SM}}(\theta) = \mathbb{E}_{p(x_t)} \left[ \text{tr}(\nabla_{x_t} s_\theta(x_t, t)) + \frac{1}{2} \| s_\theta(x_t, t) \|_2^2 \right],$$

which eliminates this term via integration by parts. However, the divergence term  $\text{tr}(\nabla_{x_t} s_\theta)$  requires computing second-order derivatives, which is computationally expensive in high dimensions.

To address this issue, Vincent (2011) showed that training a denoising autoencoder is equivalent to minimising the score matching objective, which motivates the denoising score matching (DSM) objective. To see the connection, expand eq. (9):

$$J_t(\theta) = \mathbb{E}_{p(x_t)} \left[ \| s_\theta(x_t, t) \|_2^2 \right] - 2 \mathbb{E}_{p(x_t)} \left[ s_\theta(x_t, t)^\top \nabla_{x_t} \log p(x_t) \right] + C, \quad (10)$$

where  $C := \mathbb{E}_{p(x_t)} [\|\nabla_{x_t} \log p(x_t)\|_2^2]$  is constant w.r.t.  $\theta$ . Using the law of total expectation, the first term becomes

$$\mathbb{E}_{p(x_t)} [\|s_\theta(x_t, t)\|_2^2] = \mathbb{E}_{p(x_0)} \mathbb{E}_{p(x_t|x_0)} [\|s_\theta(x_t, t)\|_2^2], \quad (11)$$

while, under certain conditions, the cross term can be rewritten by conditioning on  $x_0$ :

$$\begin{aligned} \mathbb{E}_{p(x_t)} [s_\theta(x_t, t)^\top \nabla_{x_t} \log p(x_t)] &= \int_{x_t} s_\theta(x_t, t)^\top \nabla_{x_t} p(x_t) dx_t \\ &= \int_{x_t} s_\theta(x_t, t)^\top \nabla_{x_t} \int_{x_0} p(x_t|x_0) p(x_0) dx_0 dx_t \\ &= \int_{x_0} \int_{x_t} s_\theta(x_t, t)^\top \nabla_{x_t} \log p(x_t|x_0) p(x_t|x_0) p(x_0) dx_t dx_0 \\ &= \mathbb{E}_{p(x_0)} \mathbb{E}_{p(x_t|x_0)} [s_\theta(x_t, t)^\top \nabla_{x_t} \log p(x_t|x_0)]. \end{aligned} \quad (12)$$

Substituting eqs. (11) and (12) into eq. (10) and completing the square yields

$$J_t(\theta) = \mathbb{E}_{p(x_0)} \mathbb{E}_{p(x_t|x_0)} [\|s_\theta(x_t, t) - \nabla_{x_t} \log p(x_t|x_0)\|_2^2] + C',$$

where  $C'$  is constant w.r.t.  $\theta$ . Dropping constants gives the DSM loss:

$$J_t^{\text{DSM}}(\theta) = \mathbb{E}_{x_0 \sim p_{\text{data}}} \mathbb{E}_{x_t \sim p(x_t|x_0)} [\|s_\theta(x_t, t) - \nabla_{x_t} \log p(x_t|x_0)\|_2^2]. \quad (13)$$

This equivalence, first established by Vincent (2011), is the basis of modern score-based generative modeling (Song and Ermon, 2019; Song et al., 2021b). In the variance-preserving (VP) SDE, the conditional distribution is Gaussian  $p(x_t|x_0) = \mathcal{N}(\mu_t, \sigma_t^2 I)$ , where  $\mu_t$  and  $\sigma_t^2$  are given by eqs. (5) and (6), respectively, so the conditional score is analytic:

$$\nabla_{x_t} \log p(x_t|x_0) = -\frac{x_t - \mu_t}{\sigma_t^2},$$

yielding the practical training loss

$$J_t^{\text{DSM}}(\theta) = \mathbb{E}_{x_0 \sim p_{\text{data}}(x_0)} \mathbb{E}_{x_t \sim p(x_t|x_0)} \left[ \left\| s_\theta(x_t, t) + \frac{x_t - \mu_t}{\sigma_t^2} \right\|_2^2 \right].$$

Thus, the neural network learns to regress directly to the conditional score, making DSM a tractable surrogate for score matching in diffusion models.

The denoising score matching objective  $J_t^{\text{DSM}}$  is for a specific time  $t$ . To train the score model across all time  $t \in [0, 1]$  in one go, the overall objective is given by a weighted average (or sum) (Song et al., 2021b)

$$J^{\text{DSM}}(\theta) = \mathbb{E}_{t \sim \mathcal{U}(0,1)} [\lambda(t) J_t^{\text{DSM}}(\theta)], \quad (14)$$

where  $\lambda(t)$  is a positive weighting function.

The learning problem for score-based diffusion therefore reduces to denoising score matching: the neural network  $s_\theta(x, t)$  is trained to approximate the score function, and, if the objective is minimised exactly, it recovers the true score  $\nabla_x \log p_t(x)$  across all noise levels, i.e.  $s_\theta(x, t) = \nabla_x \log p_t(x)$  for all  $t \in [0, 1]$ . Once trained, this network can be used to drive the reverse SDE, enabling generation of new samples from Gaussian noise.

## 2.5 Sampling Procedures

Once the score network  $s_\theta(x, t)$  has been trained, new samples can be generated by simulating the reverse diffusion process. This requires integrating the reverse-time SDE (8) starting from Gaussian noise  $x(t=1) \sim \mathcal{N}(0, I)$  and evolving toward  $t = 0$ .

### 2.5.1 Reverse SDE Sampling via Euler-Maruyama Method

The simplest approach is to discretise the reverse SDE using the Euler-Maruyama method. For a decreasing sequence of time steps  $\{t_i\}_{i=0}^N$  with  $t_N = 1$  and  $t_0 = 0$ , the update rule is

$$x_{t_{i-1}} = x_{t_i} + [f(x_{t_i}, t_i) - g(t_i)^2 s_\theta(x_{t_i}, t_i)] \Delta t + g(t_i) \sqrt{\Delta t} z_i, \quad (15)$$

where  $z_i \sim \mathcal{N}(0, I)$  and  $\Delta t = t_{i-1} - t_i$ . This procedure generates approximate samples from the data distribution. While straightforward, its quality depends heavily on the number of discretisation steps: smaller steps yield higher fidelity at the cost of computational time.

### 2.5.2 Predictor-Corrector Sampling

The Predictor-Corrector sampler was proposed by Song et al. (2021b) as an improved sampling method. It is based on combining two components:

1. **Predictor step:** one update using the reverse SDE (e.g., via Euler-Maruyama method, eq. (15)), advancing the trajectory toward lower noise levels.
2. **Corrector step:** one or more steps of stochastic refinement, implemented as Langevin Monte Carlo:

$$x \leftarrow x + \epsilon s_\theta(x, t) + \sqrt{2\epsilon} z, \quad z \sim \mathcal{N}(0, I),$$

where  $\epsilon$  is a step size. This step locally improves sample quality by pushing the state toward regions of higher density according to the trained score network.

By alternating predictor and corrector steps, the algorithm achieves a better trade-off between quality and efficiency than pure reverse SDE sampling. In practice, only a few corrector iterations per time step are needed to significantly improve sample fidelity.

In this dissertation, we use predictor-corrector sampling as our default procedure.

## 2.6 Noise Schedules

The design of the noise schedule plays a central role in the performance of score-based diffusion models. The schedule determines how the variance of the forward diffusion process evolves over time, which directly influences both training stability and the quality of generated samples.

### 2.6.1 Linear Schedules

In the original DDPM formulation (Ho et al., 2020), the variance schedule  $\{\beta_i\}_{i=1}^N$  was chosen to increase linearly from a small value to a larger one over  $N$  steps. Equivalently, in

the continuous SDE formulation (Song et al., 2021b), this corresponds to a linear growth of the noise rate

$$\beta(t) = \beta_0 + (\beta_1 - \beta_0) t.$$

While simple, this schedule can be suboptimal: early time steps may inject too little noise, leading to poor coverage of high-noise regions during training, while later steps may over-noise the data, resulting in wasted computation.

### 2.6.2 Cosine Variance-Preserving Schedule

Nichol and Dhariwal (2021) proposed a cosine schedule for the noising process in DDPM, which has since become widely adopted (Dhariwal and Nichol, 2021; Saharia et al., 2022; Watson et al., 2023). The proposed cosine schedule corresponds to using

$$\bar{\alpha}_i = \frac{\cos^2\left(\frac{\pi}{2} \cdot \frac{i/N+s}{1+s}\right)}{\cos^2\left(\frac{\pi}{2} \frac{s}{1+s}\right)}$$

in eq. (2) for noising process, where  $s$  is a very small offset (typically  $10^{-4}$ ) to prevent singularities, that is (ignoring the denominator in  $\bar{\alpha}_i$  as it is nearly 1 for small  $s$ ),

$$x_i = \sqrt{\bar{\alpha}_i} x_0 + \sqrt{1 - \bar{\alpha}_i} z_i, \quad \bar{\alpha}_i \approx \cos^2\left(\frac{\pi}{2} \cdot \frac{i/N+s}{1+s}\right), \quad z_i \sim \mathcal{N}(0, I).$$

This is a DDPM perturbation scheme. To generalise to continuous time, we do exactly the same as in section 2.2.2: define function  $\bar{\alpha}\left(\frac{i}{N}\right) := \bar{\alpha}_i$ , stepsize  $\Delta t := 1/N$  and time as  $t = i\Delta t \in \{0, \Delta t, \dots, N\Delta t = 1\}$ . This yields the continuous time schedule

$$x(t) = \sqrt{\bar{\alpha}(t)} x(0) + \sqrt{1 - \bar{\alpha}(t)} z(t), \quad \bar{\alpha}(t) = \cos^2\left(\frac{\pi}{2} \cdot \frac{t+s}{1+s}\right), \quad z(t) \sim \mathcal{N}(0, I).$$

Match with previous derivation, eq. (7), we are left to solve

$$\bar{\alpha}(t) = e^{-\int_0^t \beta(s) ds}.$$

It follows that

$$\begin{aligned} \beta(t) &= -\frac{d}{dt} \log \cos^2\left(\frac{\pi}{2} \cdot \frac{t+s}{1+s}\right) \\ &= -\frac{1}{\cos^2\left(\frac{\pi}{2} \cdot \frac{t+s}{1+s}\right)} \left[ 2 \cos\left(\frac{\pi}{2} \cdot \frac{t+s}{1+s}\right) \right] \left[ -\sin\left(\frac{\pi}{2} \cdot \frac{t+s}{1+s}\right) \right] \frac{d}{dt} \left(\frac{\pi}{2} \cdot \frac{t+s}{1+s}\right) \\ &= \frac{\pi}{1+s} \tan\left(\frac{\pi}{2} \cdot \frac{t+s}{1+s}\right). \end{aligned} \tag{16}$$

Instead of linear growth, the cosine schedule ensures a smoother increase in noise variance, the noising strength changes much slower near  $i = 1$  and  $i = N$ , as illustrated by fig. 1. This schedule maintains higher signal levels for longer, enabling the score network to learn more effectively across a wider range of noise intensities.

Empirically, the cosine schedule improves sample quality and accelerates convergence compared to linear schedules. It balances the trade-off between injecting sufficient noise

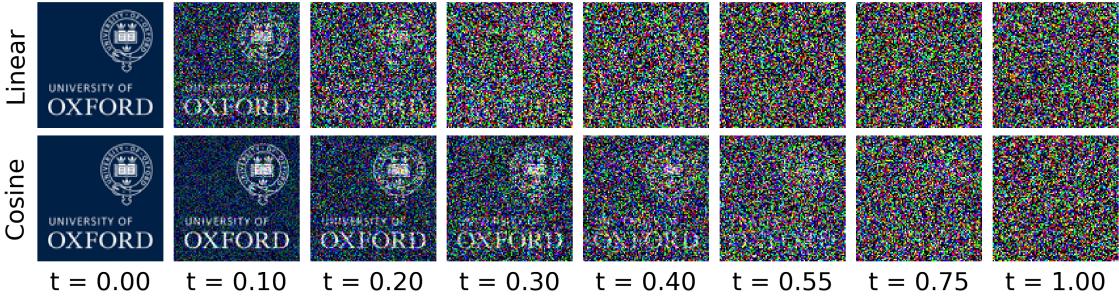


Figure 1: Linear schedule vs. cosine schedule.

for diversity and preserving enough signal for stable training. Moreover, because it avoids excessively noisy late stages, fewer discretisation steps are needed during sampling, reducing computational cost.

In this dissertation, we adopt this cosine schedule as the forward diffusion process. This choice is motivated by its demonstrated effectiveness in image and structural generative tasks (Dhariwal and Nichol, 2021; Saharia et al., 2022; Watson et al., 2023), and by its compatibility with the variance-preserving SDE formulation used in our score model, eq. (16). All experiments in later chapters therefore use the cosine schedule as the default setting.

## 2.7 Summary

This chapter has established the theoretical foundations of score-based diffusion models, which form the core generative framework employed in this dissertation. We began by introducing the forward diffusion process, first in its discrete DDPM formulation and then in its continuous-time generalisation as a stochastic differential equation. We then derived the reverse-time SDE, showing that generative modelling reduces to the problem of score estimation. This led naturally to the denoising score matching objective, which enables training a neural network to approximate the score function across all noise levels. Finally, we discussed practical sampling algorithms, including reverse SDE integration and predictor-corrector methods, and examined the role of noise schedules, with particular emphasis on the cosine variance-preserving schedule adopted in this work.

Together, these elements provide a principled probabilistic framework for deep generative modelling. Unlike GANs or VAEs, score-based diffusion models avoid common issues such as mode collapse or restrictive latent assumptions, while offering strong theoretical guarantees and flexible sampling procedures. The remainder of this dissertation builds upon these foundations to adapt score-based diffusion to the setting of protein backbone generation, where structural constraints and geometric representations introduce unique modelling challenges.

### 3 Protein Structure Modelling with Graph Neural Networks

#### 3.1 Protein Structure and Motivation

Proteins are built from amino acids, small organic molecules that share a common structure: a central alpha carbon ( $C_\alpha$ ) bonded to an amino group, a carboxyl group, a hydrogen atom, and a variable side chain (Branden and Tooze, 2012), as illustrated in fig. 2a. Amino acids are joined linearly by peptide bonds, which link the amino group of one amino acid to the carboxyl group of the next, forming a polypeptide chain (Branden and Tooze, 2012), as illustrated in fig. 2b. The sequence of amino acids in this chain defines the primary structure of a protein, while the chemistry of their side chains drives folding into higher-order structures (Creighton, 1993; Branden and Tooze, 2012). Understanding and modelling protein structures is a central challenge in computational biology because structure underlies stability, dynamics, and molecular interactions.

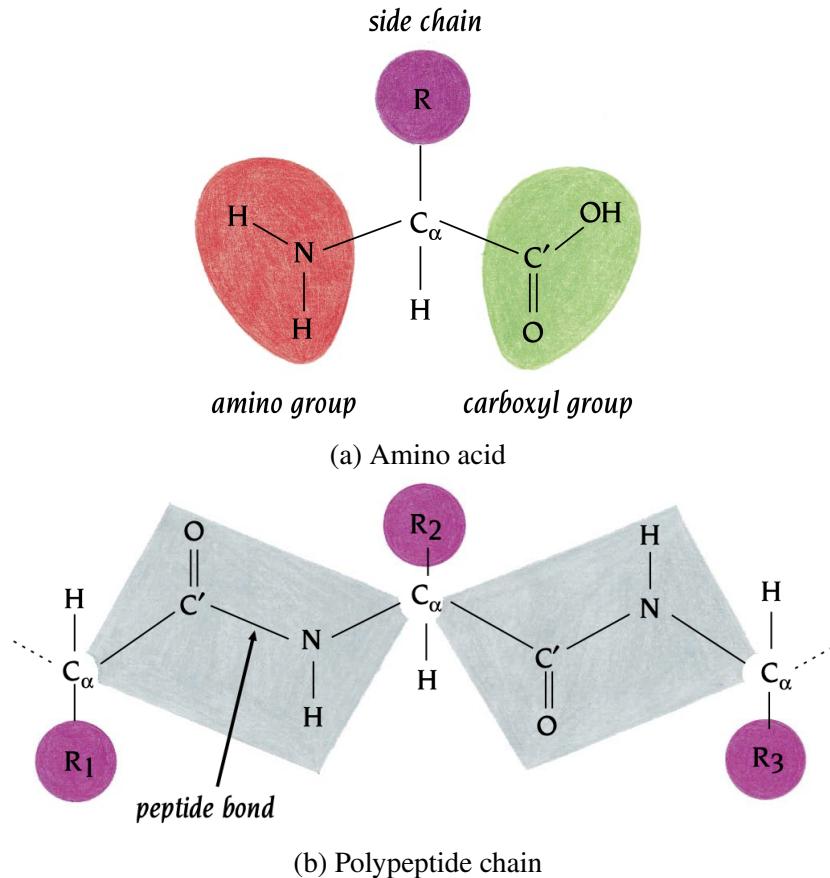


Figure 2: Schematic diagram of an amino acid and polypeptide chain from Branden and Tooze (2012). In this dissertation, we are interested only in the backbone formed by  $C_\alpha$  atoms.

Because the  $C_\alpha$  atoms define the backbone geometry of the chain, they provide a natural coarse-grained representation for structural modelling. In this work we adopt a  $C_\alpha$ -only representation: for a protein (chain of amino acid)  $x$  of length  $L$ , each amino acid

$i \in \{1, \dots, L\}$  is associated with a 3D coordinate  $x^i \in \mathbb{R}^3$ , so  $x \in \mathbb{R}^{L \times 3}$ . This coarse-grained representation is widely used in structural modelling because it captures global fold geometry and secondary structure while keeping the state space compact, making it well-suited to generative modelling (Watson et al., 2023; Zhang and Skolnick, 2005; Senior et al., 2020).

Even at this simplified level, protein geometry is far from arbitrary. The distance between successive  $C_\alpha$  atoms is consistently about

$$\|x^i - x^{i-1}\| \approx 3.8 \text{ \AA}.$$

Local backbone conformation can be described by the angle formed by three consecutive  $C_\alpha$  atoms,  $\angle(x^{i-1}, x^i, x^{i+1})$ , and by the dihedral angle defined by four consecutive  $C_\alpha$  atoms,  $\text{dihedral}(x^{i-2}, x^{i-1}, x^i, x^{i+1})$ . The empirically observed distributions of these geometric quantities are restricted by steric hindrance and the planarity of the peptide bond. Beyond these local constraints, valid structures must also avoid steric clashes, where non-adjacent amino acids are placed unrealistically close in space, and they typically fold into compact, physically plausible conformations (Creighton, 1993; Branden and Tooze, 2012).

These intricate three-dimensional dependencies make protein backbones fundamentally different from unstructured data such as images or text. Consequently, generative models for proteins must account for spatial and structural constraints explicitly, motivating the use of geometry-aware representations and neural architectures.

**Scope of this work.** Our goal is backbone-level generative modelling using score-based diffusion. We do not impose hard geometric constraints during training or sampling (e.g. fixed bond lengths or bond angles); instead, we learn the distribution over  $C_\alpha$  coordinate trajectories and evaluate fidelity against native backbones. Section 4 reports RMSD/TM results and discusses the limitation that high RMSD or low TM can still correspond to physically valid but non-native folds.

## 3.2 Graph Representation of Backbones

A protein backbone of length  $L$  is represented as a graph  $G = (V, E)$ , where each amino acid (or its corresponding  $C_\alpha$  atoms) corresponds to a node and edges capture both geometric proximity and sequential connectivity. This formulation enables the use of graph neural networks as score models, allowing information to be propagated through local neighbourhoods while respecting the spatial structure of the backbone.

**Nodes and node features.** Each node  $i \in V$  corresponds to a amino acid with coordinate  $x^i \in \mathbb{R}^3$  representing the position of its  $C_\alpha$  atom. For simplicity, we do not include amino acid type or side-chain information in this work, focusing purely on the geometry of the  $C_\alpha$  atoms backbone. In addition to the coordinates (perturbation noises), extra node features are initialised as positional embeddings derived from the amino acid index (see section 3.3, eq. (19)), along with time-conditioning features from the diffusion process (see section 3.3, eq. (18)). The full node feature vector is then obtained by concatenation:

$$\text{Node feature} = [\text{noises} \parallel \text{positional embedding} \parallel \text{time embedding}].$$

**Edges.** Edges are constructed using a radius graph: for each node  $i \in V$ , we connect edges to all neighbours  $j$  within a cutoff distance  $r$ , subject to a maximum of  $k$  nearest neighbours. Formally,

$$(i, j) \in E \Leftrightarrow \|x^i - x^j\| \leq r \quad \text{and} \quad j \text{ is among the } k \text{ closest to } i.$$

This design ensures that local spatial relationships are captured without requiring a fixed sequence length. In addition to geometric edges, self-loops are included to stabilise message passing and allow each node to retain its own features. Edges are directed: for each  $(i, j) \in E$ , messages are passed from  $j$  (target) to  $i$  (source). This convention is consistent with the implementation in PyTorch Geometric, and we do not duplicate edges to create an undirected graph.

**Edge features.** For each edge  $(i, j) \in E$ , we compute a feature vector combining geometric and diffusion-time information. The feature vector includes the following:

- **Radial basis encoding:** The distance  $\|x^i - x^j\|$  is expanded into a vector of length  $M$  using Gaussian radial basis functions,

$$\phi_\ell(\|x^i - x^j\|) = \exp\left(-\frac{(\|x^i - x^j\| - c_\ell)^2}{2\epsilon^2}\right), \quad \ell = 1, \dots, M,$$

where centres  $\{c_\ell\}_{\ell=1}^M$  are distributed uniformly between 0 and cutoff distance  $r$ , length scale  $\epsilon = r/M$ . This provides a smooth representation of distances. It is implemented via `e3nn.soft_one_hot_linspace` from Geiger and Smidt (2022); Geiger et al. (2025).

- **Directional encoding:** The normalised vector

$$d^{(i,j)} = \frac{x^i - x^j}{\|x^i - x^j\|} \in \mathbb{R}^3$$

encodes the relative orientation of the neighbour. This directional feature allows the network to capture local geometry beyond mere distances (Watson et al., 2023).

- **Time embedding:** The diffusion timestep  $t$  is embedded into high-dimensional features (see section 3.3, eq. (18)) and projected through a linear layer. For each edge  $(i, j)$ , we associate the time embedding with the source node  $i$ , enabling time-dependent conditioning of messages.

The full edge feature vector is then obtained by concatenation:

$$h_{ij} = [\phi_{1:M} \parallel d^{(i,j)}, \forall (i, j) \in E \parallel \text{time embedding}].$$

**Rationale.** This graph construction has several desirable properties: (i) it scales linearly with the number of amino acids ( $C_\alpha$  atoms)  $L$ , (ii) it incorporates both distance and orientation information, which are critical for protein geometry, and (iii) it enables variable-length backbones to be handled naturally without padding. These features make it well-suited for score-based diffusion over protein structures.

### 3.3 Score Models Architectures

The score network  $s_\theta(x, t)$  is parameterised by a neural network that maps noisy protein backbones to score estimates. We explore several architectures of increasing complexity, from a padding-based UNet model to a Graph Neural Network that utilises the graph representation.

#### 3.3.1 UNet Architecture on Padded Data

As a baseline, we implemented a UNet architecture (Ronneberger et al., 2015) with one dimensional convolutional neural networks applied directly to backbone coordinates represented as sequences of fixed length. Each protein was zero-padded to a common maximum length, and the UNet operated on the resulting coordinate tensors. This design provides a simple convolutional benchmark but does not handle variable-length inputs naturally and discards explicit geometric relationships between amino acids. It serves primarily as a sanity check to compare graph-based approaches against a conventional deep learning model.

#### 3.3.2 Graph Neural Network on Graph-Structured Data

**Graph Neural Networks.** Neural networks can be extended to graph-structured data through Graph Neural Networks (GNNs) (Scarselli et al., 2009; Gilmer et al., 2017; Battaglia et al., 2018). In this framework, each node updates its hidden representation by aggregating information, or messages, from its neighbours, typically following the generic message-passing scheme (Gilmer et al., 2017):

$$h_i^{(\ell+1)} = U\left(h_i^{(\ell)}, M\left(\left\{h_j^{(\ell)}, e_{ij} : j \in \mathcal{N}(i)\right\}\right)\right), \quad (17)$$

where  $h_i^{(\ell)}$  is the hidden state of node  $i$  at layer  $\ell$ ,  $\mathcal{N}(i)$  denotes its neighbourhood,  $e_{ij}$  are edge features,  $M$  is a function computing messages, and  $U$  is an update function that integrates messages into the node state.

One of the earliest and most widely used specialisations of this framework is the Graph Convolutional Network (GCN) (Kipf and Welling, 2017), which aggregates neighbour states through a normalised sum, analogous to convolution neural network (CNN). While effective, GCNs are limited in expressiveness, as they treat all neighbours uniformly and rely only on node features.

Subsequent work has introduced more expressive GNN variants. In this study we build on the Transformer-based graph convolution layer from Shi et al. (2021), which enhances message passing by using an attention mechanism over both node and edge features. This allows the update in eq. (17) to assign different weights to different neighbours depending on their features, improving the ability of the model to capture the geometric and sequential structure of protein backbones. Specifically, the functions  $M$  and  $U$  in eq. (17) are defined via

$$\begin{aligned} q_i &= W_q h_i^{(\ell)}, \quad k_j = W_k h_j^{(\ell)}, \\ \alpha_{ij}^{(\ell)} &= \text{softmax}_{j \in \mathcal{N}(i)}(q_i^\top (k_j + W_e e_{ij})), \end{aligned}$$

$$h_i^{(\ell+1)} = \sum_{j \in N(i)} \alpha_{ij}^{(\ell)} W_v h_j^{(\ell)},$$

where  $e_{ij}$  is the edge feature vector,  $\alpha_{ij}^{(\ell)}$  are the attention weights, and  $W_q, W_k, W_e, W_v$  are learnable projections. In practice, multiple attention heads are computed in parallel and concatenated, but we present the single-head formulation for clarity. This formulation allows geometric encodings such as distances, orientations, and diffusion time embeddings to directly influence the attention weights, making the layer particularly well suited to protein backbone graphs.

**Time conditioning via Gaussian Fourier features.** Following Song et al. (2021b), we embed the diffusion time step  $t \in (0, 1)$  using Gaussian Fourier features (Tancik et al., 2020). Let  $d_t$  denote the dimension of time embedding, and let  $W \in \mathbb{R}^{d_t/2}$  have independent components  $W_k \sim \mathcal{N}(0, s^2)$  for  $k = 1, \dots, d_t/2$ . Then the embedding is defined by

$$\gamma(t) = [\cos(2\pi Wt), \sin(2\pi Wt)] \in \mathbb{R}_t^d, \quad (18)$$

where cosine and sine are applied elementwise. This provides a smooth and expressive encoding of time, allowing the score network to capture the evolution of noise levels across the diffusion process. In practice,  $\gamma(t)$  is passed through a shallow multi-layer perceptron (MLP) before being concatenated with node and edge features to condition the network on the current diffusion step.

**Sinusoidal positional embeddings of amino acid indices.** Graph-based models are inherently invariant to node permutations, which can be a limitation when modeling protein backbones where the sequential order of amino acids carries biologically meaningful directionality along the chain. To inject this ordering information, each node  $i \in \{1, \dots, L\}$  is assigned a normalized scalar index  $i/L \in (0, 1]$ . This scalar is mapped to a high-dimensional representation using sinusoidal positional encodings (Vaswani et al., 2017),

$$\rho(i) = [\sin(\omega_1 i/L), \cos(\omega_1 i/L), \dots, \sin(\omega_{d_i/2} i/L), \cos(\omega_{d_i/2} i/L)] \in \mathbb{R}^{d_i}, \quad (19)$$

where  $d_i$  is the dimension of positional embedding, and  $\{\omega_k\}$  are fixed, exponentially spaced frequencies. The resulting positional embedding is passed through a shallow multi-layer perceptron (MLP) and concatenated with the initial node features, enabling the network to distinguish amino acids by their sequential position along the backbone.

**Residual structure.** Training deep graph networks is challenging because repeated aggregation can lead to oversmoothing of node representations and vanishing gradients (Li et al., 2018, 2019). Residual connections (He et al., 2016) have since become a standard component of graph neural networks (Kipf and Welling, 2017; Li et al., 2019). They allow each layer to refine representations relative to the previous ones rather than completely replacing them, facilitating gradient flow through many stacked layers.

In this work we adopt a residual structure for every GNN block, so that the output of each layer is added to its input before passing through nonlinearities and normalisation. To further stabilise training, we employ GraphNorm (Cai et al., 2021), a normalisation technique

specifically designed for graph neural networks. GraphNorm normalises node embeddings using graph-level statistics, improving convergence speed and reducing training instability compared to BatchNorm or LayerNorm.

This combination of residual connections and GraphNorm improves optimisation stability, mitigates oversmoothing, and enables effective training of multi-layer architectures. Empirically, such designs have been shown to enhance performance on long chains and large graphs, which is particularly important in our setting of protein backbones where sequence lengths vary widely and capturing long-range dependencies is essential.

**Remark 3.1.** *The UNet baseline provides a simple non-graph benchmark. In practice, the inclusion of positional encodings has a noticeable effect on model quality. To quantify this contribution, we include a second baseline that employs the aforementioned GNN architecture without positional encodings. This variant captures local geometric neighbourhood information but lacks explicit representations of sequential order, thereby limiting its ability to model the directional structure of protein backbones.*

### 3.4 Summary

In this section we described the methodology for adapting score-based diffusion models to the task of protein backbone generation. We began by outlining the structural constraints of proteins and motivated the use of a  $C_\alpha$ -only representation. We then formulated backbones as graphs, with nodes corresponding to amino acids and edges defined through a radius neighbourhood, augmented by radial basis encodings, directional features, and diffusion-time embeddings. This representation captures geometric information in a permutation-invariant manner while naturally supporting variable-length backbones.

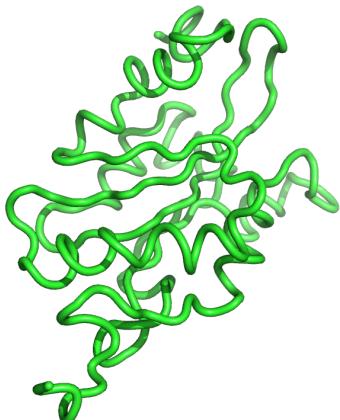
We constructed a graph neural network for the score network: a residual Transformer-based graph convolutional network with positional embeddings. The latter integrates geometric, sequential, and temporal information, supported by residual connections and GraphNorm.

The next section presents experimental results, benchmarking these models on the CATH S40 dataset using both quantitative metrics (RMSD, TM-score) and qualitative structural visualisations.

## 4 Experiments

This section presents the experimental evaluation of the score-based diffusion models described in sections 2 and 3. We benchmark three architectures on the CATH S40 dataset and assess their performance using both quantitative metrics and qualitative visualisations.

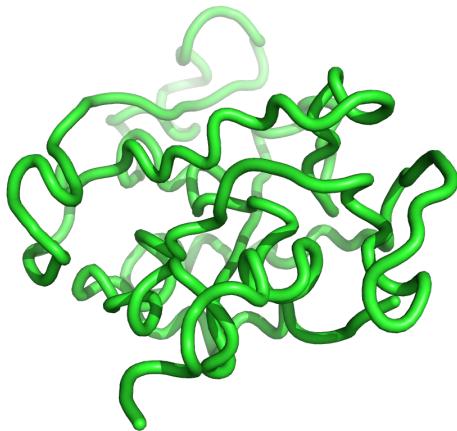
Figure 3 provides intuition before presenting numerical results. It shows samples of generated backbones compared with a native structure, illustrating the kind of structural fidelity and variability we aim to capture with our models. The following subsections then present dataset details, training and sampling setup, evaluation metrics, and quantitative benchmarking results, before returning to a more complete analysis of visualisations in section 4.5.



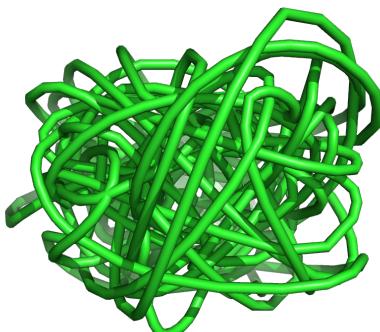
(a) True sample from CATH S40.



(b) Sample from Unet model.



(c) Sample from GNN w/ positional encoding.



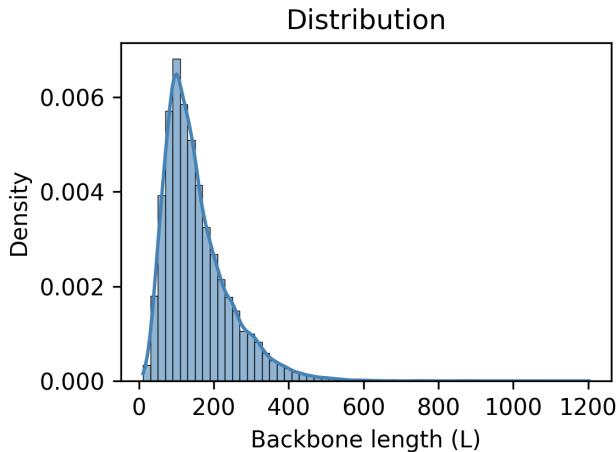
(d) Sample from GNN w/o positional encoding.

Figure 3: PyMOL visualisation of generated backbone samples from different models

## 4.1 Dataset

We train our models on the CATH S40 dataset (Orengo et al., 1997; Waman et al., 2024; Sillitoe et al., 2021; Lewis et al., 2018), a widely used benchmark for structural modelling tasks. The dataset clusters protein domains at 40% sequence identity, ensuring reduced redundancy while maintaining broad structural diversity.

For each protein domain, we extract the backbone coordinates by retaining only the  $C_\alpha$  atom of each amino acid, yielding a variable-length sequence of coordinates  $\{x^i \in \mathbb{R}^3\}_{i=1}^L$ , where  $L$  denotes the chain length (see section 3.1). The distribution and summary statistics of backbone lengths are shown in fig. 4 and table 1. Backbones are then converted into graphs as described in section 3.2, and coordinates are normalised before training.



Length ( $L$ )	
Min	11
Max	1202
Mean	155
Median	133
Std	89

Table 1: Summary statistics of backbone lengths.

Figure 4: Distribution of backbone lengths.

## 4.2 Training and Sampling

**Training setup.** For the forward noising process we adopt the variance-preserving (VP) SDE formulation (eq. (4)) with the cosine noise schedule (eq. (16)). The training objective is the denoising score matching loss (eq. (14)), optimising the score networks described in section 3.3. Models are trained using the Adam optimiser (Kingma and Ba, 2017) with a batch size dynamically adapted to GPU memory constraints.

**Sampling.** New backbone structures are generated by simulating the reverse SDE using predictor-corrector (PC) sampling, starting from Gaussian noise (section 2.5.2). This procedure enables diverse yet structurally plausible backbone generations.

**Implementation.** All models are implemented in Python using PyTorch and PyTorch Geometric (Fey and Lenssen, 2019; Fey et al., 2025). Experiments were conducted on NVIDIA GPUs with 48GB memory.

### 4.3 Evaluation Metrics

We evaluate reconstruction fidelity using two widely adopted measures of structural similarity: root-mean-square deviation (RMSD) and TM-score.

**Root-mean-square deviation (RMSD).** RMSD provides a direct measure of average atomic displacement between two aligned structures. Given a generated backbone  $X = \{x_i\}_{i=1}^L$  and its reference  $Y = \{y_i\}_{i=1}^L$ , the RMSD is defined as

$$\text{RMSD}(X, Y) = \sqrt{\frac{1}{L} \sum_{i=1}^L \|x_i - y_i\|^2}.$$

Lower values indicate closer correspondence to the reference. Although widely used, RMSD can be overly sensitive to local misalignments, and large deviations in one region may dominate the score even if the overall fold is preserved.

**TM-align and TM-score.** To complement RMSD, we use TM-score (Zhang and Skolnick, 2004; Xu and Zhang, 2010), a structural similarity measure in  $(0, 1]$ , computed via the TM-align algorithm (Zhang and Skolnick, 2005). Unlike RMSD, TM-score is less affected by local errors and more reflective of global fold similarity. In practice, TM-align performs a sequence-independent superposition of two protein structures using heuristic dynamic programming, returning both an optimal alignment and the corresponding TM-score. Interpretation of TM-scores follows established conventions: values below 0.2 typically correspond to unrelated structures, while scores above 0.5 generally indicate that two proteins share the same overall fold within SCOP/CATH classification. Thus, TM-score serves as a more robust global measure of structural fidelity and complements RMSD in our evaluation.

**Remark 4.1.** *Because each generated backbone is compared to its specific native target of the same length  $L$ , RMSD and TM-score measure reconstruction fidelity, not absolute physical validity. This is appropriate for our benchmarking setting (paired evaluation), but one should interpret low TM or high RMSD with caution: a generated backbone may still represent a plausible alternative fold while scoring poorly against the chosen reference.*

### 4.4 Quantitative Results

We benchmarked three model variants: (i) a UNet baseline on padded coordinate sequences, (ii) a Transformer-based graph convolutional network with and without positional encodings, as detailed in section 3.3. Evaluation was performed using RMSD and TM-score, as described in section 4.3.

**Overall performance.** The summary statistics of overall performance is reported in table 2. From the results, the Transformer-based graph convolutional network with positional encodings achieves the best overall performance, with the lowest RMSD ( $16.39 \pm 4.29$  Å) and highest TM-score ( $0.272 \pm 0.041$ ). By contrast, the same architecture without positional encodings performs substantially worse, highlighting the importance of

incorporating sequential information in addition to geometric neighbourhoods. Interestingly, the UNet baseline outperforms the GCN without positional encodings, despite its simplistic design and reliance on padded sequences. This suggests that while graph representations capture local geometry, they require positional encodings to effectively model the directional structure of protein backbones.

Table 2: Quantitative evaluation of backbone reconstruction fidelity. Values reported are mean  $\pm$  standard deviation.

Model	RMSD ( $\text{\AA}$ )	TM-score
UNet baseline	$17.230 \pm 4.699$	$0.264 \pm 0.041$
GCN w/o positional encoding	$20.459 \pm 4.985$	$0.145 \pm 0.023$
GCN w/ positional encoding	$16.394 \pm 4.294$	$0.272 \pm 0.041$

**Distribution analysis.** In addition to summary statistics, we plot the distribution plot and box plot of RMSD and TM-scores across the dataset, figs. 5 and 6. These results reinforce the trends observed in table 2. The GCN model with positional encodings exhibits the most favourable distribution, with RMSD values concentrated toward lower errors and TM-scores shifted toward higher similarity. The UNet baseline, although simpler, produces a broader distribution with a heavier tail of poorly reconstructed examples. By contrast, the GCN without positional encodings consistently underperforms, with TM-scores clustered around low values and RMSD heavily shifted left, demonstrating that sequential information is critical for modelling backbone geometry.

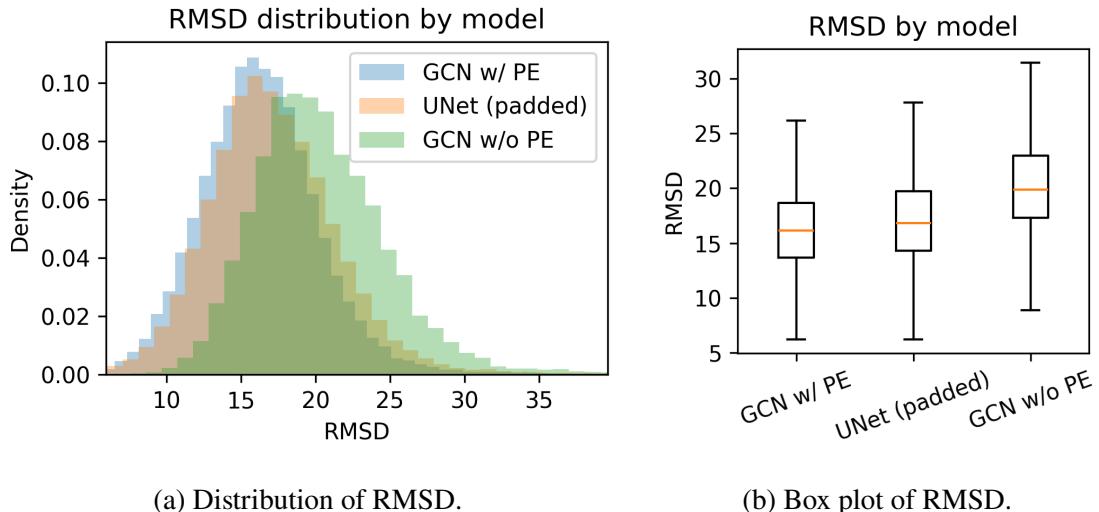


Figure 5: Distribution and box plot of RMSD values across test structures. Lower is better.

We further analyse model performance as a function of backbone length  $L$ , plotting RMSD and TM-score versus  $L$  in fig. 7. Both metrics exhibit a clear dependence on sequence length: reconstruction error (RMSD) increases with  $L$ , while TM-score tends to decrease for longer proteins. This trend reflects the inherent difficulty of maintaining structural

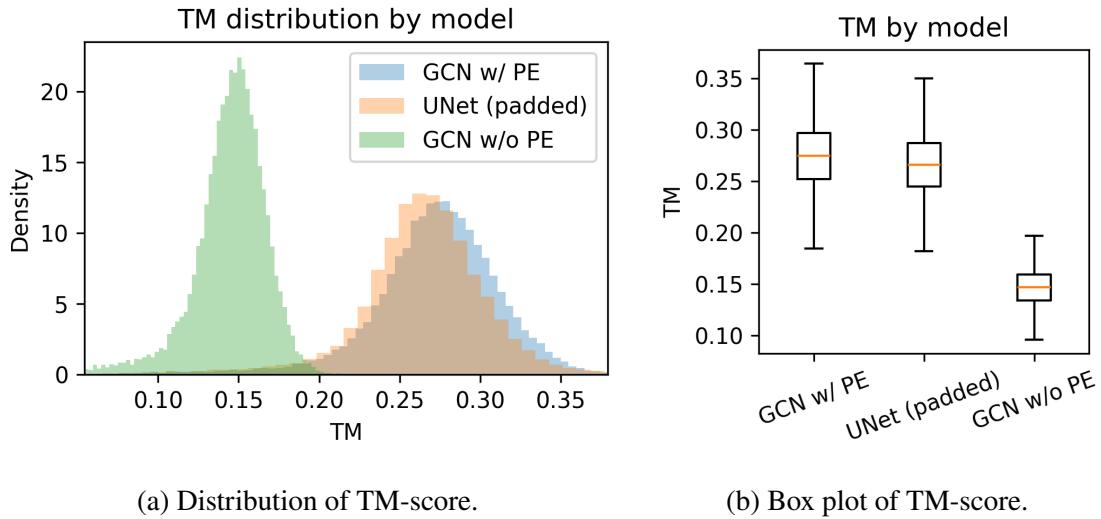


Figure 6: Distribution and box plot of TM-scores across test structures. Higher is better.

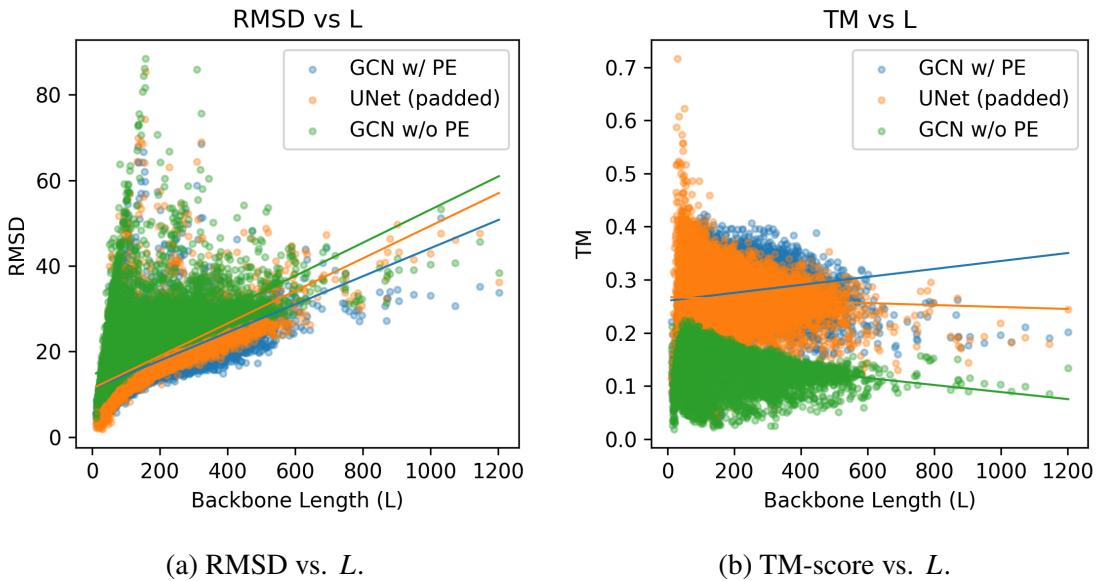


Figure 7: RMSD and TM-score versus backbone length  $L$ .

fidelity over extended chains, where deviations accumulate and long-range dependencies become harder to capture.

Across all lengths, the GCN model with positional encodings achieves the best performance, with lower RMSD growth rates and consistently higher TM-scores. The UNet baseline performs comparably on shorter backbones but degrades more quickly as length increases, consistent with its reliance on fixed-length padded representations. The GCN without positional encodings consistently underperforms, with rapidly increasing RMSD and TM-scores falling below 0.2 for most lengths, highlighting its inability to capture directional structure. These results emphasise that positional encodings are particularly critical for modelling long backbones, where sequential information provides essential constraints on geometry.

## 4.5 Qualitative Visualisation

We now return to the qualitative visualisations introduced earlier in fig. 3, which show generated backbones alongside their corresponding native structures, rendered in PyMOL (Schrödinger, LLC, 2015c,b,a). These examples provide an intuitive sense of how different architectures behave: they illustrate both successful reconstructions, where the generated backbone closely follows the native fold (low RMSD, high TM-score), and failure cases, where the model produces geometrically plausible but non-native conformations.

The Transformer-based GCN with positional encodings produces generations that resemble the overall topology of the reference backbone, though with local deviations. The UNet baseline yields broadly plausible structures but can introduce artifacts, particularly in regions affected by padding. The GCN without positional encodings performs noticeably worse in this example, producing a collapsed global fold. While these individual cases are not sufficient for firm conclusions, they are broadly consistent with the quantitative trends: positional information improves global fold recovery, and its absence leads to systematic degradation. Together, these visualisations complement the statistical results by illustrating the kinds of conformations the models tend to generate.

## 4.6 Summary

In summary, our experiments demonstrate that incorporating positional encodings into a Transformer-based graph convolutional network substantially improves backbone generation fidelity. Across quantitative metrics, the positional GCN achieved the lowest RMSD and highest TM-scores, outperforming both the UNet baseline and the GCN without positional information. Distributional analyses further showed that this model produces more consistent samples, while both quantitative results and qualitative visualisations highlighted that the absence of positional encodings often leads to collapsed or unrealistic folds.

At the same time, challenges remain. RMSD and TM-scores still indicate a gap between generated and native backbones, particularly for long chains where deviations accumulate. Individual qualitative examples provide intuition but do not fully capture the diversity of generated structures, suggesting the need for larger-scale structural assessments.

Overall, these results underscore the importance of combining geometric and sequential information when modelling proteins with diffusion processes. The following chapter discusses the broader implications of these findings, outlines limitations of the present work, and suggests directions for future research.

## 5 Discussion and Conclusion

### 5.1 Discussion of Results

The experiments in section 4 demonstrate that score-based diffusion models can generate protein backbones with meaningful structural similarity to native folds. Among the evaluated architectures, the Transformer-based GCN with positional embeddings consistently outperformed the UNet baseline and the GCN without positional encodings in both RMSD and TM-score. This finding underscores the importance of incorporating sequential order into graph-based architectures, complementing geometric edge features such as distances and orientations.

### 5.2 Limitations

While the results are encouraging, several limitations must be acknowledged:

- **Geometric constraints.** No explicit bond length or bond angle constraints were imposed during training or sampling. Although the model learns approximate geometry from data, explicit constraints would ensure chemical validity and prevent subtle violations of backbone stereochemistry (Branden and Tooze, 2012; Creighton, 1993; Watson et al., 2023).
- **Backbone-only representation.** Only  $C_\alpha$  atoms were modelled, ignoring side chains and residue types. This simplifies the state space but limits biological interpretability and applicability to sequence-conditioned design tasks.
- **Model scale.** The models used in this study are relatively small compared to state-of-the-art generative protein models (Wu et al., 2024; Watson et al., 2023). While we demonstrated feasibility even for backbones exceeding 1000 amino acids, scaling to larger networks with equivariant layers, global attention, or pretraining on larger structural databases may further improve results.
- **Evaluation metrics.** RMSD and TM-score measure fidelity to a specific native fold of length  $L$ . However, a generated backbone with high RMSD or low TM may still correspond to a physically plausible alternative fold. Thus, our evaluation measures reconstruction fidelity rather than unconditional structural validity.

### 5.3 Related Work and Future Directions

Recent years have seen rapid progress in generative modelling of protein structures. Diffusion-based methods in particular have achieved state-of-the-art results: Watson et al. (2023) demonstrated that RoseTTAFold-based diffusion models can generate novel proteins with designed functions, while FoldingDiff (Wu et al., 2024) applied score-based diffusion directly to backbone coordinates. At the architectural level, SE(3)-equivariant neural networks (Satorras et al., 2021; Geiger and Smidt, 2022) have been proposed to improve geometric fidelity and long-range modelling, building on earlier work in equivariant tensor field networks (Thomas et al., 2018; Weiler et al., 2018; Kondor et al., 2018).

Inspired by these developments, as well as the limitations identified in this dissertation, several promising directions follow from this study.

**Constraint integration.** One natural extension is to incorporate stereochemical constraints directly into the generative process. In the present work, bond lengths, bond angles, and other relevant structural constraints were only learned approximately from data. Future models could embed these constraints within the diffusion dynamics, ensuring that generated samples lie on a physically valid manifold. Recent advances in Riemannian score-based generative modelling (De Bortoli et al., 2022; Huang et al., 2022; Chung et al., 2022) provide a principled framework for defining diffusion processes and training objectives on constrained manifolds, which could be adapted to enforce protein backbone geometry explicitly.

**Advanced architectures.** Another promising direction is to adopt architectures with built-in geometric equivariance. While the graph neural networks capture local geometry via edge features, it is not strictly equivariant under rotations or reflections. By contrast, SE(3)-equivariant GNNs (Thomas et al., 2018; Weiler et al., 2018; Kondor et al., 2018; Geiger and Smidt, 2022; Geiger et al., 2025) encode these symmetries by construction, ensuring that predictions transform consistently with input coordinates. This property is particularly attractive for protein backbones, where physical laws are invariant to global orientation. Recent works (Yim et al., 2024) have begun exploring this idea in the context of protein diffusion models.

## 5.4 Conclusion

This dissertation has explored the use of score-based diffusion models for protein backbone generation. We outlined the theoretical background of diffusion models, developed a graph-based representation of protein backbones, and implemented several neural architectures for score estimation. Experiments on the CATH S40 dataset showed that a residual Transformer-based GCN with positional embeddings achieved improved performance over both a UNet baseline and a graph model without positional information.

These results suggest that diffusion models combined with graph-based architectures offer a viable approach to coarse-grained protein structure generation. Although the models used here are relatively small in scale, the findings highlight the value of incorporating sequential information into graph representations and point toward directions for improvement, including constraint integration, larger architectures, and sequence conditioning.

## A Code Appendix

This appendix provides excerpts from the implementation of the graph neural network (GNN) model described in the main text (section 3.3.2). The code is written in Python and makes use of standard machine learning libraries, including PyTorch and PyTorch Geometric (PyG) (Fey and Lenssen, 2019; Fey et al., 2025). For readability, only the core components of the model are shown here. The complete source code used in this dissertation, including data preprocessing, training and sampling scripts, and relevant utilities, is available in the accompanying GitHub repository: [TODO:MakePublicRepository](#).

```
1  class GraphTransformerScoreModel(nn.Module):
2      def __init__(self, hidden_dim=128, pos_embed_dim=128,
3          t_embed_dim=128, t_edge_proj_dim=16, num_layers=5, heads=8,
4          max_num_neighbors=500, radius=5, num_basis=32):
5          super().__init__()
6          self.max_num_neighbors = max_num_neighbors
7          self.radius = radius
8          self.num_basis = num_basis
9          self.heads = heads
10
11         # position embedding
12         self.pos_embed = nn.Sequential(
13             SinusoidalEncoding(embed_dim=pos_embed_dim),
14             nn.Linear(pos_embed_dim, pos_embed_dim)
15         )
16
17         # position MLPs per layer
18         self.pos_mlps = nn.ModuleList([
19             nn.Linear(pos_embed_dim, pos_embed_dim) for _ in
20             range(num_layers)
21         ])
22
23         # Time embedding
24         self.t_embed = nn.Sequential(
25             GaussianFourierProjection(t_embed_dim, scale=16.),
26             nn.Linear(t_embed_dim, t_embed_dim),
27         )
28
29         self.t_edge_proj = nn.Linear(t_embed_dim,
30             t_edge_proj_dim)
31
32         # Time MLPs per layer
33         self.t_mlps = nn.ModuleList([
34             nn.Linear(t_embed_dim, t_embed_dim) for _ in
35             range(num_layers)
36         ])
```

```

31     ])
32
33     # TransformerConv layers
34     in_channels = hidden_dim + pos_embed_dim + t_embed_dim
35     self.convs = nn.ModuleList([
36         TransformerConv(
37             in_channels=in_channels,
38             out_channels=hidden_dim // heads,
39             heads=heads,
40             concat=True,
41             beta=True,
42             edge_dim=self.num_basis + 3 + t_edge_proj_dim
43         )
44         for _ in range(num_layers)
45     ])
46
47     # Norm layers
48     self.norms = nn.ModuleList([
49         GraphNorm(hidden_dim) for _ in range(num_layers)
50     ])
51
52     # Input/output projection
53     self.input_proj = nn.Linear(3, hidden_dim)
54     self.output_proj = nn.Sequential(
55         nn.Linear(hidden_dim, hidden_dim),
56         nn.SiLU(),
57         nn.Linear(hidden_dim, 3)
58     )
59
60     self.act = nn.SiLU()
61
62     def forward(self, coords, batch, t):
63         # Positional encoding (based on position index in the
64         # chain)
65         num_graphs = batch.max().item() + 1
66         pos_index = torch.empty_like(batch, dtype=torch.float)
67         for i in range(num_graphs):
68             mask = (batch == i)
69             count = mask.sum()
70             pos_index[mask] = torch.linspace(0, 1, steps=count,
71                                         device=coords.device)
72         pos_feat = self.act(self.pos_embed(pos_index[:, None]))
73         # [N, pos_embed_dim]
74
75         # Time embedding

```

```

73     t_feat = self.act(self.t_embed(t[:, None])) # [B,
74     t_embed_dim]
75
76     # Node features
77     h = self.input_proj(coords) # [N, hidden_dim]
78
79     # Build radius graph to get the edge index
80     edge_index = radius_graph(coords, r=self.radius,
81                               batch=batch, max_num_neighbors=self.max_num_neighbors,
82                               loop=True, flow='target_to_source')
83
84     # edge attributes
85     edge_vec = coords[edge_index[0]] - coords[edge_index[1]]
86     edge_dir = F.normalize(edge_vec, dim=-1)
87     edge_length = edge_vec.norm(dim=-1)
88     edge_scalar = soft_one_hot_linspace(
89         edge_length,
90         start=0.0,
91         end=self.radius,
92         number=self.num_basis,
93         basis='gaussian',
94         cutoff=True
95     ) # [num_edges, num_basis]
96     t_edge = t_feat[batch][edge_index[0]] # [num_edges,
97     t_embed_dim]
98     t_edge = self.t_edge_proj(t_edge) # [num_edges,
99     t_edge_proj_dim]
100    # Combine with direction vectors
101    edge_attr = torch.cat([edge_scalar, edge_dir, t_edge],
102                          dim=-1) # [num_edges, num_basis + 3 + t_edge_proj_dim]
103
104    for conv, norm, pos_mlp, t_mlp in zip(self.convs,
105                                           self.norms, self.pos_mlps, self.t_mlps):
106        t_per_node = t_mlp(t_feat)[batch] # [N, t_embed_dim]
107        pos = pos_mlp(pos_feat)
108
109        # Residual
110        h_res = h
111
112        # Concat time + feature
113        h = conv(torch.cat([h, pos, t_per_node], dim=-1),
114                 edge_index=edge_index, edge_attr=edge_attr)
115
116        # Normalization + residual
117        h = self.act(norm(h, batch) + h_res)

```

```

110
111     return self.output_proj(h)
112
113
114 class SinusoidalEncoding(nn.Module):
115     def __init__(self, embed_dim=128):
116         super().__init__()
117         self.embed_dim = embed_dim
118
119     def forward(self, pos_index):
120         i = torch.arange(self.embed_dim, device=pos_index.device)
121         angle_rates = 1 / torch.pow(10000, (2 * (i // 2)) /
122                                     self.embed_dim)
123         angle_rads = pos_index * angle_rates
124         angle_rads[:, 0::2] = torch.sin(angle_rads[:, 0::2])
125         angle_rads[:, 1::2] = torch.cos(angle_rads[:, 1::2])
126         return angle_rads
127
128
129 class GaussianFourierProjection(nn.Module):
130     def __init__(self, embed_dim, scale=30.):    # embed_dim must
131         be divisible by 2
132         super().__init__()
133         self.W = nn.Parameter(torch.randn(embed_dim // 2) *
134                             scale, requires_grad=False)
135
136     def forward(self, t):
137         x_proj = 2 * torch.pi * t * self.W  # [B, D/2]
138         return torch.cat([torch.sin(x_proj), torch.cos(x_proj)], dim=-1)

```

## References

- Anderson, B. D. O. (1982). Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 12(3):313–326.
- Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., and Pascanu, R. (2018). Relational inductive biases, deep learning, and graph networks.
- Branden, C. I. and Tooze, J. (2012). *Introduction to Protein Structure*. Garland Science, New York, 2 edition.
- Cai, T., Luo, S., Xu, K., He, D., Liu, T.-Y., and Wang, L. (2021). GraphNorm: A Principled Approach to Accelerating Graph Neural Network Training. In *Proceedings of the 38th International Conference on Machine Learning*, pages 1204–1215. PMLR.
- Chung, H., Sim, B., Ryu, D., and Ye, J. C. (2022). Improving Diffusion Models for Inverse Problems using Manifold Constraints. *Advances in Neural Information Processing Systems*, 35:25683–25696.
- Creighton, T. E. (1993). *Proteins: structures and molecular properties*. Macmillan.
- De Bortoli, V., Mathieu, E., Hutchinson, M., Thornton, J., Teh, Y. W., and Doucet, A. (2022). Riemannian Score-Based Generative Modelling. *Advances in Neural Information Processing Systems*, 35:2406–2422.
- Dhariwal, P. and Nichol, A. (2021). Diffusion Models Beat GANs on Image Synthesis. In *Advances in Neural Information Processing Systems*, volume 34, pages 8780–8794. Curran Associates, Inc.
- Fey, M. and Lenssen, J. E. (2019). Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.
- Fey, M., Sunil, J., Nitta, A., Puri, R., Shah, M., Stojanović, B., Bendias, R., Alexandria, B., Kocijan, V., Zhang, Z., He, X., Lenssen, J. E., and Leskovec, J. (2025). PyG 2.0: Scalable learning on real world graphs. In *Temporal Graph Learning Workshop @ KDD*.
- Geiger, M. and Smidt, T. (2022). E3nn: Euclidean Neural Networks.
- Geiger, M., Smidt, T., M, A., Miller, B. K., Boomsma, W., Dice, B., Lapcheyvskyi, K., Kotak, M., Tyszkiewicz, M., Uhrin, M., Batzner, S., Madisetti, D., Frellsen, J., Maheshkar, S., Jung, N., Fomitchev, B., Wen, M., jkh, Sanborn, S., Barnes, R., Kohler, C., Morehead, A., Tan, C. W., Rackers, J., LFu, Rød, M., Bailey, M., Brocidiaco, M., eszter137, and McConkey, R. (2025). Euclidean neural networks: E3nn. Zenodo.
- Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural Message Passing for Quantum Chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1263–1272. PMLR.

- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2020). Generative adversarial networks. *Commun. ACM*, 63(11):139–144.
- He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.
- Ho, J., Jain, A., and Abbeel, P. (2020). Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc.
- Huang, C.-W., Aghajohari, M., Bose, J., Panangaden, P., and Courville, A. C. (2022). Riemannian Diffusion Models. *Advances in Neural Information Processing Systems*, 35:2750–2761.
- Hyvärinen, A. (2005). Estimation of Non-Normalized Statistical Models by Score Matching. *Journal of Machine Learning Research*, 6(24):695–709.
- Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard, A. J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., Back, T., Petersen, S., Reiman, D., Clancy, E., Zielinski, M., Steinegger, M., Pacholska, M., Berghammer, T., Bodenstein, S., Silver, D., Vinyals, O., Senior, A. W., Kavukcuoglu, K., Kohli, P., and Hassabis, D. (2021). Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589.
- Kingma, D. P. and Ba, J. (2017). Adam: A Method for Stochastic Optimization.
- Kingma, D. P. and Welling, M. (2022). Auto-Encoding Variational Bayes.
- Kipf, T. N. and Welling, M. (2017). Semi-Supervised Classification with Graph Convolutional Networks.
- Kondor, R., Lin, Z., and Trivedi, S. (2018). Clebsch-Gordan Nets: A Fully Fourier Space Spherical Convolutional Neural Network.
- Lewis, T. E., Sillitoe, I., Dawson, N., Lam, S. D., Clarke, T., Lee, D., Orengo, C., and Lees, J. (2018). Gene3D: Extensive prediction of globular domains in proteins. *Nucleic Acids Research*, 46(D1):D435–D439.
- Li, G., Muller, M., Thabet, A., and Ghanem, B. (2019). DeepGCNs: Can GCNs Go As Deep As CNNs? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9267–9276.
- Li, Q., Han, Z., and Wu, X.-m. (2018). Deeper Insights Into Graph Convolutional Networks for Semi-Supervised Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1).
- Nichol, A. and Dhariwal, P. (2021). Improved Denoising Diffusion Probabilistic Models.
- Orengo, C. A., Michie, A. D., Jones, S., Jones, D. T., Swindells, M. B., and Thornton,

- J. M. (1997). CATH – a hierachic classification of protein domain structures. *Structure*, 5(8):1093–1109.
- Ronneberger, O., Fischer, P., and Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In Navab, N., Hornegger, J., Wells, W. M., and Frangi, A. F., editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham. Springer International Publishing.
- Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E. L., Ghasemipour, K., Gontijo Lopes, R., Karagol Ayan, B., Salimans, T., Ho, J., Fleet, D. J., and Norouzi, M. (2022). Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding. *Advances in Neural Information Processing Systems*, 35:36479–36494.
- Satorras, V. G., Hoogeboom, E., and Welling, M. (2021). E(n) Equivariant Graph Neural Networks. In *Proceedings of the 38th International Conference on Machine Learning*, pages 9323–9332. PMLR.
- Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2009). The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1):61–80.
- Schrödinger, LLC (2015a). The AxPyMOL molecular graphics plugin for Microsoft PowerPoint, version 1.8. Website: <https://pymol.org/axpymol>.
- Schrödinger, LLC (2015b). The JyMOL molecular graphics development component, version 1.8. Website: <https://pymol.org/axpymol>.
- Schrödinger, LLC (2015c). The PyMOL molecular graphics system, version 1.8. Website: <https://pymol.org/axpymol>.
- Senior, A. W., Evans, R., Jumper, J., Kirkpatrick, J., Sifre, L., Green, T., Qin, C., Žídek, A., Nelson, A. W. R., Bridgland, A., Penedones, H., Petersen, S., Simonyan, K., Crossan, S., Kohli, P., Jones, D. T., Silver, D., Kavukcuoglu, K., and Hassabis, D. (2020). Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792):706–710.
- Shi, Y., Huang, Z., Feng, S., Zhong, H., Wang, W., and Sun, Y. (2021). Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification.
- Sillitoe, I., Bordin, N., Dawson, N., Waman, V. P., Ashford, P., Scholes, H. M., Pang, C. S. M., Woodridge, L., Rauer, C., Sen, N., Abbasian, M., Le Cornu, S., Lam, S. D., Berka, K., Varekova, I. H., Svobodova, R., Lees, J., and Orengo, C. A. (2021). CATH: Increased structural coverage of functional space. *Nucleic Acids Research*, 49(D1):D266–D273.
- Song, Y., Durkan, C., Murray, I., and Ermon, S. (2021a). Maximum Likelihood Training of Score-Based Diffusion Models. In *Advances in Neural Information Processing Systems*, volume 34, pages 1415–1428. Curran Associates, Inc.
- Song, Y. and Ermon, S. (2019). Generative Modeling by Estimating Gradients of the Data Distribution. *Advances in Neural Information Processing Systems*, 32.
- Song, Y. and Ermon, S. (2020). Improved Techniques for Training Score-Based Generative Models. In *Advances in Neural Information Processing Systems*, volume 33, pages 12438–12448. Curran Associates, Inc.

- Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. (2021b). Score-Based Generative Modeling through Stochastic Differential Equations. In *International Conference on Learning Representations*.
- Tancik, M., Srinivasan, P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J., and Ng, R. (2020). Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains. In *Advances in Neural Information Processing Systems*, volume 33, pages 7537–7547. Curran Associates, Inc.
- Thomas, N., Smidt, T., Kearnes, S., Yang, L., Li, L., Kohlhoff, K., and Riley, P. (2018). Tensor field networks: Rotation- and translation-equivariant neural networks for 3D point clouds.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ukasz Kaiser, Ł., and Polosukhin, I. (2017). Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.
- Vincent, P. (2011). A Connection Between Score Matching and Denoising Autoencoders. *Neural Computation*, 23(7):1661–1674.
- Waman, V. P., Bordin, N., Alcraft, R., Vickerstaff, R., Rauer, C., Chan, Q., Sillitoe, I., Yamamori, H., and Orengo, C. (2024). CATH 2024: CATH-AlphaFlow Doubles the Number of Structures in CATH and Reveals Nearly 200 New Folds. *Journal of Molecular Biology*, 436(17):168551.
- Watson, J. L., Juergens, D., Bennett, N. R., Trippe, B. L., Yim, J., Eisenach, H. E., Ahern, W., Borst, A. J., Ragotte, R. J., Milles, L. F., Wicky, B. I. M., Hanikel, N., Pellock, S. J., Courbet, A., Sheffler, W., Wang, J., Venkatesh, P., Sappington, I., Torres, S. V., Lauko, A., De Bortoli, V., Mathieu, E., Ovchinnikov, S., Barzilay, R., Jaakkola, T. S., DiMaio, F., Baek, M., and Baker, D. (2023). De novo design of protein structure and function with RFdiffusion. *Nature*, 620(7976):1089–1100.
- Weiler, M., Geiger, M., Welling, M., Boomsma, W., and Cohen, T. (2018). 3D Steerable CNNs: Learning Rotationally Equivariant Features in Volumetric Data.
- Wu, K. E., Yang, K. K., van den Berg, R., Alamdari, S., Zou, J. Y., Lu, A. X., and Amini, A. P. (2024). Protein structure generation via folding diffusion. *Nature Communications*, 15(1):1059.
- Xu, J. and Zhang, Y. (2010). How significant is a protein structure similarity with TM-score = 0.5? *Bioinformatics*, 26(7):889–895.
- Yim, J., Stärk, H., Corso, G., Jing, B., Barzilay, R., and Jaakkola, T. S. (2024). Diffusion models in protein structure and docking. *WIREs Computational Molecular Science*, 14(2):e1711.
- Zhang, Y. and Skolnick, J. (2004). Scoring function for automated assessment of protein structure template quality. *Proteins: Structure, Function, and Bioinformatics*, 57(4):702–710.
- Zhang, Y. and Skolnick, J. (2005). TM-align: A protein structure alignment algorithm based on the TM-score. *Nucleic Acids Research*, 33(7):2302–2309.