A Latent Diffusion Model for Protein Structure Generation

Cong Fu[1]*, Keqiang Yan[1]*, Limei Wang[1], Wing Yee Au[2], Michael McThrow[2], Tao Komikado[3], Koji Maruhashi[3], Kanji Uchino[2], Xiaoning Qian[1], Shuiwang Ji[1]
[1] Texas A&M University, College Station, TX, USA [2] Fujitsu Research of America, Sunnyvale, CA, USA
[3] Fujitsu Research, Kanagawa, Japan
`{congfu, keqiangyan, limei, xqian, sji}@tamu.edu`
`{wau, mmcthrow, komikado.tao, maruhashi.koji, kanji}@fujitsu.com`

Abstract

Proteins are complex biomolecules that perform a variety of crucial functions within living organisms. Designing and generating novel proteins can pave the way for many future synthetic biology applications, including drug discovery. However, it remains a challenging computational task due to the large modeling space of protein structures. In this study, we propose a latent diffusion model that can reduce the complexity of protein modeling while flexibly capturing the distribution of natural protein structures in a condensed latent space. Specifically, we propose an equivariant protein autoencoder that embeds proteins into a latent space and then uses an equivariant diffusion model to learn the distribution of the latent protein representations. Experimental results demonstrate that our method can effectively generate novel protein backbone structures with high designability and efficiency. The code will be made publicly available at `https://github.com/divelab/AIRS/tree/main/OpenProt/LatentDiff`.

1 Introduction

Artificial intelligence has emerged as a promising approach that significantly enhances scientific research across various fields [1], such as physical simulation [2, 3], quantum mechanics [4, 5], materials [6, 7], and biology [8–13]. The discovery of novel proteins [14–19] is crucial in biomedicine. Recently, instead of generating novel protein sequences [20–27] and then predicting their corresponding structures, Trippe et al. [28] and Wu et al. [29] propose to directly generate protein structures using diffusion models, due to the impressive modeling power and generation quality of diffusion models [30–34] for images and small molecules. However, generating 3D protein structures is a more challenging task because of their complex geometric structures and vast exploration space. Additionally, as the modeling space increases, the cost of time and computational resources required to train and sample from diffusion models also increases significantly.

There are attempts to reduce the modeling space in the image and small molecule domain for diffusion models. Stable Diffusion [34] combines a pretrained image autoencoder and a latent diffusion model to reduce the modeling space for large images. However, there are currently no robust and powerful 3D graph autoencoders and latent diffusion models for 3D protein structures. Torsional Diffusion [33] only focuses on torsional angles and employs RDKit [35] predictions for bond lengths and bond angles, as the distributions of bond angles and lengths are highly confined in small molecules. But this assumption does not hold for protein structures.

In this paper, we reduce the diffusion modeling space of complex 3D protein structures by integrating a 3D graph autoencoder and a latent 3D diffusion model. To achieve this, the following challenges are addressed: (1) ensuring rotation equivariance in the autoencoder design, (2) accurately reconstructing intricate connection information in 3D graphs during decoding, and (3) developing a specialized latent diffusion process for 3D protein latent representations, including position and node latent representations. In the following sections, we first recap the background and related works for

*Equal contributions

protein backbone structure generation and diffusion models in Sec. 2, and then show in detail how we address the above challenges in Sec. 3. The efficiency and ability to generate novel protein backbone structures of our proposed method are demonstrated in Sec. 4.

2 Background and Related Work

2.1 Protein Backbone Structure Generation

Protein backbone generation aims to generate novel protein backbone structures by learning from real data distributions. To this end, a mapping between known distributions, such as a Gaussian, and the real data distribution, which is high dimensional and sparse, needs to be constructed. Since protein global geometric structures are mainly determined by backbones, the generation of protein structures can be simplified to the generation of backbones consisting of a sequence of amino acids and their corresponding positions. Following ProtDiff [28], we use the positions of alpha carbons to represent amino acid positions. The protein backbone structure is then represented by

$$\mathcal{S} = \{(\boldsymbol{x}_i, a_i)\}_{i=1}^{n}, \tag{1}$$

where $\boldsymbol{x}_i \in \mathbb{R}^3$ denotes the 3D position of alpha carbon in the i-th amino acid, and $a_i \in \{k | 1 \leq k \leq 20, k \in \mathbb{Z}\}$ denotes the corresponding amino acid type.

Instead of modeling amino acid types and alpha carbon positions together, previous studies [28] have shown that it is better to decompose the whole generation process into two stages as $p(\boldsymbol{X}, \boldsymbol{a}) = p(\boldsymbol{a}|\boldsymbol{X})p(\boldsymbol{X})$, where $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n]$, and $\boldsymbol{a} = [a_1, a_2, \cdots, a_n]^T$. Specifically, the positions of alpha carbons are first generated, and the corresponding amino acid types are predicted using pretrained inverse folding models such as ProteinMPNN [36].

2.2 Denoising Diffusion Probabilistic Models

As a powerful class of generative models [37–39], denoising diffusion probabilistic models (DDPM) [30] solve the Bayesian inverse problem of deriving the underlying data distribution $p_{\text{data}}(\boldsymbol{z})$ by establishing a bijective mapping between given prior distributions and $p_{\text{data}}(\boldsymbol{z})$. We review the background of DDPM here following the adopted conventions of ScoreSDE [31]. To enable faithful generation based on $p_{\text{data}}(\boldsymbol{z})$ by sampling simpler prior distributions, a discrete Markov chain is employed to gradually diffuse inputs as a map from given training data into random noise, for example, following multivariate normal (Gaussian) distributions. For every training sample $\boldsymbol{z}_0 \sim p_{\text{data}}(\boldsymbol{z})$, DDPMs consider a sequence of variance values $0 < \beta_1, \beta_2, \ldots, \beta_N < 1$ and construct a discrete Markov chain $\{\boldsymbol{z}_0, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_N\}$, where $p(\boldsymbol{z}_i|\boldsymbol{z}_{i-1}) = \mathcal{N}(\boldsymbol{z}_i; \sqrt{1-\beta_i}\boldsymbol{z}_{i-1}, \beta_i \mathbf{I})$. Based on this, we obtain $p(\boldsymbol{z}_i|\boldsymbol{z}_0) = \mathcal{N}(\boldsymbol{z}_i; \sqrt{\alpha_i}\boldsymbol{z}_0, (1-\alpha_i)\mathbf{I})$, where $\alpha_i = \prod_{t=0}^{i}(1-\beta_t)$. Hence, a sequence of noise scales can be predefined such that $\alpha_N \to 0$ and $\boldsymbol{z}_N$ is approximately distributed according to $\mathcal{N}(\mathbf{0}, \mathbf{I})$. For the reverse mapping from $\mathcal{N}(\mathbf{0}, \mathbf{I})$ to $p_{\text{data}}(\boldsymbol{z})$, a reverse Markov chain is parameterized as $p_\theta(\boldsymbol{z}_{i-1}|\boldsymbol{z}_i) = \mathcal{N}(\boldsymbol{z}_i; \mu_\theta(\boldsymbol{z}_i, i), \beta_i \mathbf{I})$, where $\mu_\theta(\boldsymbol{z}_i, i) = \frac{1}{\sqrt{1-\beta_i}}(\boldsymbol{z}_i - \frac{\beta_i}{\sqrt{1-\alpha_i}}\mathbf{s}_\theta(\boldsymbol{z}_i, i))$. The reverse diffusion model $\mathbf{s}_\theta$ is trained with a re-weighted evidence lower bound (ELBO) as below

$$\boldsymbol{\theta}^\star = \text{argmin}_{\boldsymbol{\theta}} \mathbb{E}_{t, \boldsymbol{z}_0, \boldsymbol{\sigma}}[\|\boldsymbol{\sigma} - \boldsymbol{s}_{\boldsymbol{\theta}}(\sqrt{\alpha_t}\boldsymbol{z}_0 + \sqrt{1-\alpha_t}\boldsymbol{\sigma}, t)\|^2], \tag{2}$$

where $\boldsymbol{\sigma} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. After $\mathbf{s}_\theta$ is trained, the reverse sampling process is conducted by first sampling from $\boldsymbol{z}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and then updating from time N to time 0 by the estimated reverse Markov chain

$$\boldsymbol{z}_{t-1} = \frac{1}{\sqrt{1-\beta_t}}(\boldsymbol{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{z}_t, t)) + \sqrt{\beta_t}\boldsymbol{\sigma}. \tag{3}$$

2.3 Related Work

Diffusion Models for Protein Structure Generation. Recent research [28, 29, 40–45] has been exploring the use of diffusion models to generate novel protein structures, building on the successes of diffusion models in other areas such as images [30, 31] and small molecules [32, 33, 46]. Among them, ProtDiff [28] focuses on generating protein backbone structures by determining the positions of alpha carbons, while FoldingDiff [29] represents protein backbone structures using bond and torsion angles and applies a sequence diffusion model to generate new backbone structures. Anand and Achim [40] attempts to generate the entire protein structure by using three separate diffusion models

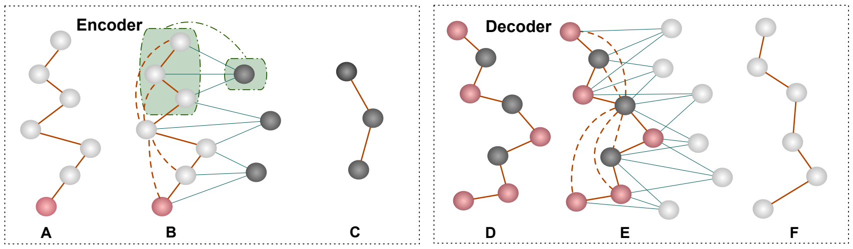


Figure 1: Autoencoder network structure for proteins. Step A, B, and C denote the Encoder network. A. Augmented input protein structure (white) with padding (red node), similar to image padding. B. (1) Edge building: create a fully connected graph (limited edges shown for simplicity) on the padded structure; (2) Graph Expansion: introduce new nodes (black) with specific connections according to the 1D-CNN convention. C. Compressed structure (in latent space). Steps D, E, and F denote the Decoder network. D. Padding latent structure for upsampling (similar to padding operation in image transpose convolution). E. Edge building and Graph Expansion are similar to B. F. Reconstructed protein chain.

to generate alpha carbon positions, amino acid types, and side chain rotation angles sequentially, but the joint modeling performance is relatively low. Additionally, Lee and Kim [41] proposes to diffuse 2D pairwise distances and angle matrices for amino acid residues, but further optimization using Rosseta minimization [47] is needed.

It is worth noting that, concurrent with the development of our method, several other works have emerged, capable of generating high-quality proteins. RFdiffusion [42] takes advantage of the powerful protein structure prediction model, RoseTTAFold [48], to achieve remarkable results on many generation tasks. RFdiffusion pretrains RoseTTAFold on the protein structure prediction task and then finetunes on generative tasks. But RFdiffusion only demonstrates the effectiveness of generating proteins when using pretrained weights. Chroma [43] uses a correlated diffusion process to transform protein structures into random collapsed polymers and encode the chain and radius of gyration constraints by a designed covariance model. In this way, Chroma can model the target distribution more efficiently by preserving some basic structures in proteins. Genie [44] and FrameDiff [45] adopt oriented reference frames to model residues. Genie only considers alpha carbon atoms so diffusion only needs to be applied to atom positions. FrameDiff generates full backbone atoms so diffusion on both frame position and orientation needs to be considered.

Despite the success of protein backbone structure generation [28, 29, 40–45], the modeling space of diffusion models is still vast, necessitating significant time and computational resources for both training and sampling from diffusion models.

Decreasing Modeling Space for Protein Structure. The modeling space for protein structure generation is reduced in several ways. ProtDiff [28] only considers the positions of alpha carbons, while FoldingDiff [29] represents protein backbone structures using bond and torsion angles and omits bond lengths to decrease the modeling space. Torsional Diffusion [33] uses RDKit-generated bond lengths and angles and only diffuses the torsional angles for the conformer generation of small molecules, but it is not applicable for protein structures.

Recently, the impressive generative capability of Stable Diffusion [34] in the image domain has attracted significant attention. By integrating a pre-trained image autoencoder with latent diffusion models, Stable Diffusion reduces the modeling space of large images and improves the generative power of image diffusion models. However, 3D geometric graphs for protein structures are different from images, no robust 3D equivariant protein autoencoders and 3D latent diffusion models for protein structures have been proposed yet.

3 Method

In this section, we introduce our LatentDiff for generating protein backbone structures. We describe the design of our equivariant protein autoencoder in Section 3.1, and next the latent space diffusion model in Section 3.2.

3.1 Equivariant Protein Autoencoder

We first introduce our equivariant autoencoder that helps reduce the protein design space. To design such an autoencoder, we identify some constraints and the uniqueness of protein backbones. First, C_α atoms in protein backbones have a fixed order due to the sequential nature of amino acid sequences. In general, downsampling or upsampling of sequence data can be achieved by 1D convolutional neural networks (CNNs). Also, since C_α atoms form a chain structure that could be preserved during upsampling, we don't need to reconstruct edge connections like traditional graph autoencoder. Second, despite the sequence representation of protein backbones, they also possess 3D geometries, which require equivariance during the downsampling and upsampling stages. Traditional CNN cannot meet this equivariant requirement, but graph neural networks (GNNs) are capable of dealing with this challenge. Based on these observations, we propose a novel equivariant protein autoencoder that considers both the amino acid sequence and 3D graph information of protein backbones.

Overview. In the equivariant protein autoencoder, we first downsample proteins to smaller sizes and upsample the latent graph to reconstruct the original protein. There are four steps within each downsampling and upsampling layer, namely **structure padding**, **edge building**, **graph expansion**, and **equivariant message passing**. The first three steps are used to construct a graph that contains the input nodes and initialized downsampling or upsampling nodes in the current layer. After the message passing, only updated downsampling or upsampling nodes will be kept as input in the next layer for further downsampling or upsampling operation. In the following, we describe the network input and details of one downsampling layer. The upsampling layer shares the exact same steps except for structure padding, which we will also introduce in the structure padding section.

Network Input. For a protein backbone structure $\mathcal{S}$, we move the structure to the zero centroid in order to make the model avoid capturing translational equivariance. Then we will augment the protein to a fixed length m to simplify the remaining operations in the network. So m is the maximum protein length that we can generate, and we choose m as 128 in this work. The augmented protein is shown as the white part in Figure 1.A. Specifically, we append $m-n$ extra nodes to the end of the protein structure. Each extra node is assigned a zero position and the same node type. And we denote the augmented protein structure as $\mathcal{S}_{\text{aug}} = (\boldsymbol{X}, \boldsymbol{H})$, where $\boldsymbol{X} \in \mathbb{R}^{3\times m}$ and $\boldsymbol{H} \in \mathbb{R}^{d\times m}$ are node positions and node feature vectors respectively. For $\boldsymbol{X}$, the first n columns $\{\boldsymbol{x}_i\}_{i=1}^{n}$ denote the positions of all C_α atoms in the original protein and the last $m-n$ columns $\{\boldsymbol{x}_i\}_{i=n+1}^{m}$ denote the zero positions of extra nodes. Each node feature vector $\boldsymbol{h}_i \in \mathbb{R}^d$ in $\boldsymbol{H}$ is a d-dimensional type embedding indicating the corresponding node type. Then the preprocessed $\mathcal{S}_{\text{aug}}$ is the input to the first downsampling layer.

Structure Padding. Similar to padding in image convolution, within each layer, we first need to pad the augmented protein structure $\mathcal{S}_{\text{aug}}$ before downsampling or upsampling the structure in order to obtain an output with the desired size. Let's assume that we have k nodes after structure padding. Denote the padded structure as $\mathcal{S}_{\text{pad}} = (\boldsymbol{X}_{\text{pad}}, \boldsymbol{H}_{\text{pad}})$, where $\boldsymbol{X}_{\text{pad}} \in \mathbb{R}^{3\times k}$ and $\boldsymbol{H}_{\text{pad}} \in \mathbb{R}^{d\times k}$. As shown in Figure 1.A and D, red nodes are padding nodes. For the downsampling, we pad the input structure on the boundary by adding nodes with the same node position and node features as the boundary node. For example, in Figure 1.A, the red node is the duplicate of the last white node. For the upsampling, we need both boundary padding and internal padding, similar to image padding in transpose convolution. The boundary padding is the same as that of downsampling. For an internal padding node, such as the second red node in Figure 1.D, it is initialized with the average value of the position and node features of its two nearest nodes on both sides.

Edge Building. After structure padding, we perform an edge-building step to construct a graph from a padded protein structure $\mathcal{S}_{\text{pad}}$. We could adopt fully connected graphs in order to capture interactions between all atom pairs. As shown in Figure 1.B, the edges in the constructed complete graph are in red. For simplicity, we only show the edge connections for one node. Note that ways of edge connections can be flexible in this step. Empirically we find that constructing a complete graph only over the non-padded structure during downsampling gives better reconstruction performance.

Graph Expansion. Then, for the graph expansion step, we need to first initialize downsampled nodes and connect them to the graph constructed in the edge-building step. We denote the expanded graph as $\mathcal{G}_{\text{exp}} = (\boldsymbol{X}_{\text{exp}}, \boldsymbol{H}_{\text{exp}}, \boldsymbol{A}_{\text{exp}})$, where $\boldsymbol{X}_{\text{exp}} = [\boldsymbol{X}_{\text{pad}}, \boldsymbol{X}_{\text{down}}] \in \mathbb{R}^{3\times(k+\frac{m}{2})}$, $\boldsymbol{H}_{\text{exp}} = [\boldsymbol{H}_{\text{pad}}, \boldsymbol{H}_{\text{down}}] \in \mathbb{R}^{d\times(k+\frac{m}{2})}$, and $\boldsymbol{A}_{\text{exp}} \in \mathbb{R}^{(k+\frac{m}{2})\times(k+\frac{m}{2})}$. Specifically, we create a set of new nodes with positions

$\boldsymbol{X}_{\text{down}} \in \mathbb{R}^{3\times\frac{m}{2}}$ and node feature vectors $\boldsymbol{H}_{\text{down}} \in \mathbb{R}^{d\times\frac{m}{2}}$ which represent the downsampled structure. The edge connections between downsampled structure and the augmented protein structure are created in a 1D CNN convention. Specifically, only nodes within a kernel-sized window will be connected to a new node. For example, as shown in Figure 1.B, the green area denotes a kernel of size 3, and the first black node connects to the first three white nodes in the green area. And each new node is initialized as the average of its connected nodes for both position and node feature.

SE(3) Equivariant Message Passing. Proteins only contain right-handed alpha helices, so the network should not be equivariant to reflection. Thus, we use a SE(3) equivariant graph neural network to perform message passing on the expanded graph $\mathcal{G}_{\text{exp}}$ to update downsample nodes. We adapt the network architecture from Schneuing et al. [49], in which they modify the E(n) equivariant graph neural network (EGNN) [50] by adding an additional cross-product term in the coordinate update step. In this way, the network can be sensitive to reflection. Formally,

$$\hat{\boldsymbol{X}}_{\text{exp}}, \hat{\boldsymbol{H}}_{\text{exp}} = \text{EGNN}_{SE(3)}[\boldsymbol{X}_{\text{exp}}, \boldsymbol{H}_{\text{exp}}], \tag{4}$$

where $\hat{\boldsymbol{X}}_{\text{exp}} = [\hat{\boldsymbol{X}}, \hat{\boldsymbol{X}}_{\text{down}}]$ and $\hat{\boldsymbol{H}}_{\text{exp}} = [\hat{\boldsymbol{H}}, \hat{\boldsymbol{H}}_{\text{down}}]$. $\text{EGNN}_{SE(3)}$ contains L equivariant convolution layers (EGCL). Each layer performs a position and feature update, such that $\boldsymbol{x}_i^{l+1}, \boldsymbol{h}_i^{l+1} = \text{EGCL}[\boldsymbol{x}_i^l, \boldsymbol{h}_i^l]$, which is defined below:

$$\boldsymbol{m}_{ij} = \phi_e(\boldsymbol{h}_i^l, \boldsymbol{h}_j^l, d_{ij}^2, a_{ij}), \tag{5}$$

$$\boldsymbol{h}_i^{l+1} = \phi_h(\boldsymbol{h}_i^l, \sum_{j\neq i} \tilde{e}_{ij}\boldsymbol{m}_{ij}), \tag{6}$$

$$\boldsymbol{x}_i^{l+1} = \boldsymbol{x}_i^l + \sum_{j\neq i} \frac{\boldsymbol{x}_i^l - \boldsymbol{x}_j^l}{d_{ij}+1}\phi_x(\boldsymbol{m}_{ij}) + \frac{\left(\boldsymbol{x}_i^l - \overline{\boldsymbol{x}}^l\right) \times \left(\boldsymbol{x}_j^l - \overline{\boldsymbol{x}}^l\right)}{\left\|\left(\boldsymbol{x}_i^l - \overline{\boldsymbol{x}}^l\right) \times \left(\boldsymbol{x}_j^l - \overline{\boldsymbol{x}}^l\right)\right\| + 1}\phi_x^{\times}\left(\boldsymbol{m}_{ij}\right), \tag{7}$$

where $d_{ij} = \left\|\boldsymbol{x}_i^l - \boldsymbol{x}_j^l\right\|_2$ denotes the Euclidean distance between nodes i and j, and $a_{ij} = \text{MLP}([\boldsymbol{h}_i^l, \boldsymbol{h}_j^l])$ is the edge feature for edge (i, j). $\overline{\boldsymbol{x}}$ denotes the center of mass of all nodes. $d_{ij}+1$ can be optionally used to normalize the node distance to improve numerical stability. Following Hoogeboom et al. [46], we use an attention mechanism $\tilde{e}_{ij} = \phi_{inf}(\boldsymbol{m}_{ij})$ to infer a soft estimation of edges.

Then after the message passing, we will only keep the updated downsampled structure $(\hat{\boldsymbol{X}}_{\text{down}}, \hat{\boldsymbol{H}}_{\text{down}})$ as the input of next layer, as shown in Figure 1.C. During the upsampling stage in the decoder, we perform the same four steps as introduced above. After upsampling to the original size of the input augmented protein, we obtain a reconstructed structure with position and node embedding for each node. Then we use an MLP to process the final node embedding and predict whether a reconstructed node belongs to the augmented node type, as we describe in the following training loss section. We then use another MLP to predict the amino acid type of each node.

Training Loss. Reconstruction loss of autoencoder consists of six parts. First, we have a cross-entropy loss $\mathcal{L}_{\text{aug}}$ on a binary classification task to determine whether each reconstructed node is an augmented node that does not belong to the original protein. Next, we use another cross-entropy loss $\mathcal{L}_{\text{aa}}$ on the amino acid type prediction for each node. And then, we calculate the mean absolute error (MAE) of the position for each non-augmented node between the reconstructed protein and ground truth, and we denote it as $\mathcal{L}_{\text{pos}}$. Apart from these three losses, to further consider the secondary structure reconstruction for proteins, we also include edge distance loss $\mathcal{L}_{\text{dist}}$ and torsion angle loss $\mathcal{L}_{\text{tor}}$ calculated across the non-augmented nodes. Specifically, edge distance is calculated as the Euclidean distance between every two consecutive C_α atoms, and the torsion angle is the angle between two planes formed by four consecutive C_α atoms. To avoid latent node embeddings having an arbitrarily high variance, we use slight KL divergence loss $\mathcal{L}_{\text{reg}}$ to regularize latent node embeddings, which is similar to a variational autoencoder. So the total loss is the weighted sum of these individual losses. Formally,

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{aug}} + \mathcal{L}_{\text{aa}} + \mathcal{L}_{\text{pos}} + w_1 * \mathcal{L}_{\text{dist}} + w_2 * \mathcal{L}_{\text{tor}} + w_3 * \mathcal{L}_{\text{reg}}, \tag{8}$$

where w_1, w_2, and w_3 are relative weights to control the edge distance loss, torsion angle loss, and regularization loss, respectively. We want the network to optimize the absolute position of each node first and adjust edge distance and torsion angle later, so we set w_1 and w_2 as 0.5. Also, we want the autoencoder to have good reconstruction performance, so we only use very small regularization, and we set w_3 equal to $1e^{-4}$.

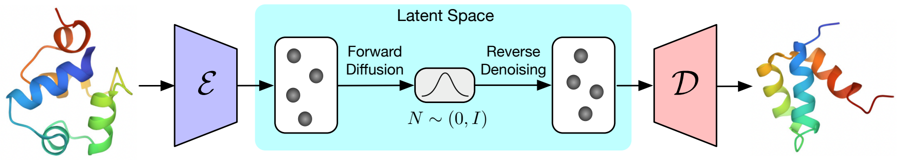


Figure 2: Pipeline of LatentDiff. Encoder $\mathcal{E}$ and decoder $\mathcal{D}$ are pretrained via equivariant protein autoencoder introduced in Section 3.1, and their parameters are fixed during training the latent diffusion. Protein structures are encoded into latent representations via the encoder $\mathcal{E}$. And latent representations are gradually perturbed into Gaussian noise. During generation, we first sample Gaussian noise and use the learned denoising network to generate protein representations in the latent space. And then, the decoder $\mathcal{D}$ decodes latent representations to protein structures.

3.2 Latent Diffusion

Modeling the extracted latent representations $(\boldsymbol{X}_{\text{down}}, \boldsymbol{H}_{\text{down}})$ of protein backbone structures poses unique challenges due to the fact that they consist of 3D Euclidean positions, which differ from images and texts. In this section, we first explain the desired distribution SE(3) invariance property and then provide a detailed description of the latent diffusion process that satisfies this property for the task of protein backbone generation. In this section, p_{data}, p_{model}, and p_θ denote the underlying data distribution, the output distribution of the whole model framework, and the latent distribution from the latent diffusion model, respectively.

Distribution SE(3) Invariance. For a given protein backbone structure $(\boldsymbol{X}, \boldsymbol{H})$, we would like the learned data distribution to be SE(3) invariant: $p_{\text{data}}(\boldsymbol{X}, \boldsymbol{H}) = p_{\text{data}}(R\boldsymbol{X} + b, \boldsymbol{H})$ as the geometric 3D structure remains unchanged after SE(3) transformations, where $R \in \mathbb{R}^{3\times3}$, $|R| = 1$ describing only the rotation transformations and $b \in \mathbb{R}^3$ for translation in 3D space. Because our protein autoencoder is translation invariant as described in Sec. 3.1, $p_{\text{model}}(\boldsymbol{X}, \boldsymbol{H}) = p_{\text{model}}(\boldsymbol{X} + b, \boldsymbol{H})$ holds naturally. Hence, distribution rotation invariance $p_{\text{model}}(\boldsymbol{X}, \boldsymbol{H}) = p_{\text{model}}(R\boldsymbol{X}, \boldsymbol{H})$ needs to be satisfied for the latent diffusion process.

In our approach, we propose to decompose the generation of protein backbone structures into two stages, including (1) protein latent representation generation and (2) latent representation decoding. The model distribution can be defined as $p_{\text{model}}(\boldsymbol{X}, \boldsymbol{H}) = p_{\text{decoder}}(\boldsymbol{X}, \boldsymbol{H} | \boldsymbol{X}_{\text{down}}, \boldsymbol{H}_{\text{down}}) p_\theta(\boldsymbol{X}_{\text{down}}, \boldsymbol{H}_{\text{down}})$ Given that the decoding process is SE(3) equivariant and deterministic, if the latent diffusion model $\mathbf{s}_\theta$ satisfies $p_\theta(\boldsymbol{X}_{\text{down}}, \boldsymbol{H}_{\text{down}}) = p_\theta(R\boldsymbol{X}_{\text{down}} + b, \boldsymbol{H}_{\text{down}})$, the distribution SE(3) invariance $p_{\text{model}}(\boldsymbol{X}, \boldsymbol{H}) = p_{\text{model}}(R\boldsymbol{X} + b, \boldsymbol{H})$ can be satisfied.

The challenge of $p_\theta(\boldsymbol{X}_{\text{down}}, \boldsymbol{H}_{\text{down}}) = p_\theta(R\boldsymbol{X}_{\text{down}} + b, \boldsymbol{H}_{\text{down}})$ can be addressed by (1) modeling zero-mean geometric distribution for $\boldsymbol{X}$, (2) using a high-dimensional Gaussian distribution as the prior distribution, and (3) employing rotation equivariant reverse diffusion process [32, 46]. Specifically, the influence of translation transformations in 3D space is omitted by reducing the central position of $\boldsymbol{X}$. Additionally, by using an isotropic high dimensional Gaussian prior, we have $p_\theta(\boldsymbol{X}_T, \boldsymbol{H}_T) = p_\theta(R\boldsymbol{X}_T, \boldsymbol{H}_T)$. The rotation equivariant reverse diffusion process further guarantees that $p_\theta(\boldsymbol{X}_t, \boldsymbol{H}_t) = p_\theta(R\boldsymbol{X}_t, \boldsymbol{H}_t)$ for any time t and the proof is provided in Appendix. A.1.

Rotational Distribution Invariant Latent Diffusion. Due to the aforementioned considerations, we propose the rotation distribution invariant latent forward and reverse diffusion processes for the extracted protein backbone latent features $(\boldsymbol{X}_{\text{down}}, \boldsymbol{H}_{\text{down}})$. The implementation is based on EDM [46] with adjustments to support the latent diffusion process. Specifically, we generate latent 3D points with position and latent node features, so we do not need to decode the node type at the last step of reverse diffusion. Additionally, since protein structures possess natural order, we add sinusoidal positional encoding features to provide sequence order information. Most importantly, similar to Section 3.1, we also modified the message passing in EDM to be SE(3) equivariant. The pipeline of our protein latent diffusion is shown in Figure 2. During the forward process, the input latent representations $(\boldsymbol{X}_{\text{down}}, \boldsymbol{H}_{\text{down}})$ are diffused slowly into random noise by a sequence of noise scales $0 < \beta_1, \beta_2, \ldots, \beta_N < 1$ as follows

$$\boldsymbol{X}_i = \sqrt{1-\beta_i}\boldsymbol{X}_{i-1} + \sqrt{\beta_i}\boldsymbol{\sigma}_{\boldsymbol{X}}, \quad (9) \qquad \boldsymbol{H}_i = \sqrt{1-\beta_i}\boldsymbol{H}_{i-1} + \sqrt{\beta_i}\boldsymbol{\sigma}_{\boldsymbol{H}}, \quad (10)$$

where $\boldsymbol{\sigma_H} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and $\boldsymbol{\sigma_X}$ is first sampled from $\mathcal{N}(\mathbf{0}, \mathbf{I})$ and then reduced based on the corresponding central position following Hoogeboom et al. [46]. And the closed-form forward process can be written as

$$\boldsymbol{X}_t = \sqrt{\alpha_t}\boldsymbol{X}_{\text{down}} + \sqrt{1-\alpha_t}\boldsymbol{\sigma_X}, \tag{11}$$

$$\boldsymbol{H}_t = \sqrt{\alpha_t}\boldsymbol{H}_{\text{down}} + \sqrt{1-\alpha_t}\boldsymbol{\sigma_H}, \tag{12}$$

where $\alpha_t = \prod_{i=0}^{t}(1-\beta_i)$. Since α_t is a scalar value, we have $p_t(\boldsymbol{X}_t, \boldsymbol{H}_t) = p(\boldsymbol{X}_{\text{down}}, \boldsymbol{H}_{\text{down}})p(\boldsymbol{\sigma_X}, \boldsymbol{\sigma_H})$ where p_t is the data distribution at time t and $p(\boldsymbol{\sigma_X}, \boldsymbol{\sigma_H}) = p(\boldsymbol{\sigma_X})p(\boldsymbol{\sigma_H})$ denotes the corresponding multivariate Gaussian distributions. It can be seen that $p_t(\boldsymbol{X}_t, \boldsymbol{H}_t) = p_t(R\boldsymbol{X}_t, \boldsymbol{H}_t)$ because $p(\boldsymbol{X}_{\text{down}}, \boldsymbol{H}_{\text{down}})p(\boldsymbol{\sigma_X}, \boldsymbol{\sigma_H}) = p(R\boldsymbol{X}_{\text{down}}, \boldsymbol{H}_{\text{down}})p(\boldsymbol{\sigma_X}, \boldsymbol{\sigma_H})$. Hence, the forward diffusion process satisfies rotation distribution invariance.

For the reverse diffusion process, a reverse Markov chain is formed as below

$$(\boldsymbol{X}_{t-1}, \boldsymbol{H}_{t-1}) = \frac{1}{\sqrt{1-\beta_t}}\boldsymbol{\mu_t} + \sqrt{\beta_t}(\boldsymbol{\sigma_X}, \boldsymbol{\sigma_H}), \tag{13}$$

$$\boldsymbol{\mu_t} = (\boldsymbol{X}_t, \boldsymbol{H}_t) - \frac{\beta_t}{\sqrt{1-\alpha_t}}\boldsymbol{s_\theta}(\boldsymbol{X}_t, \boldsymbol{H}_t, t), \tag{14}$$

where $\boldsymbol{s_\theta}$ is a rotation equivariant network implemented based on the SE(3) version of EGNN [49, 50].

Training Loss. The reverse diffusion model $\boldsymbol{s_\theta}$ is trained with a re-weighted evidence lower bound (ELBO) following ProtDiff [28] and DDPM [30] as below

$$\boldsymbol{\theta}^\star = \text{argmin}_{\boldsymbol{\theta}}\mathbb{E}_{t,(\boldsymbol{X}_{\text{down}}, \boldsymbol{H}_{\text{down}}),\boldsymbol{\sigma}}[\|\boldsymbol{\delta}\|^2], \tag{15}$$

$$\boldsymbol{\delta} = \boldsymbol{\sigma} - \boldsymbol{s_\theta}(\sqrt{\alpha_t}(\boldsymbol{X}_{\text{down}}, \boldsymbol{H}_{\text{down}}) + \sqrt{1-\alpha_t}\boldsymbol{\sigma}, t), \tag{16}$$

where $\boldsymbol{\sigma} = (\boldsymbol{\sigma_X}, \boldsymbol{\sigma_H})$.

4 Experiments

We empirically demonstrate the effectiveness and efficiency of our method for generating protein backbone structures. The overall generation process can be found in Appendix A.4. In Section 4.1, we first introduce the dataset we curated from existing protein databases and the baseline models. In Section 4.2–Section 4.4, we show the reconstruction performance of the pre-trained autoencoder, the designability of generated proteins, and the parallel sampling efficiency of LatentDiff. We also provide additional experiments about secondary structures, diversity, structural distribution of generated proteins, and structure-sequence co-design in Appendix A.5, A.6, A.9, and Appendix A.8, respectively. In Appendix A.3, we describe the training details of the autoencoder and latent diffusion model.

4.1 Experimental Setting

Dataset. We curate the dataset from Protein Data Bank (PDB) and Swiss-Prot data in AlphaFold Protein Structure Database (AlphaFold DB) [51, 52]. Details of the dataset can be found in Appendix A.2.

Baselines. To evaluate our proposed methods, we compare with three protein generation methods, ProtDiff [28], FoldingDiff [29], and FrameDiff [45]. The first two works appeared before we started developing our methods whereas FrameDiff is a more recent method of protein backbone generation.

4.2 Autoencoder Reconstruction

In this section, we show the reconstruction performance of the protein autoencoder. We compare autoencoders with different downsampling factors $f = \{2, 4, 8\}$, which we denote as $auto-f$.

Metrics. First, we evaluate the classification accuracy of augmented and non-augmented nodes (Augment Acc), and the accuracy of amino acid type classification (Residue Acc). And we have the following three geometric evaluations. We use root mean square deviation (RMSD) to compare

Table 1: Performance of autoencoder with different downsampling factors. $\uparrow$ ($\downarrow$) represents that a higher (lower) value indicates better performance.

Factor	RMSD (Å)$^{\downarrow}$	Augment Acc (%)$^{\uparrow}$	Residue Acc (%)$^{\uparrow}$	Edge Stable (%)$^{\uparrow}$	Torsion MAE (rad)$^{\downarrow}$
2	0.5280	100	99	95.29	0.4361
4	1.2755	100	98	70.99	0.8951
8	2.2772	100	45	59.97	1.1903

the absolute position error between reconstructed C_α atoms and ground truth. Additionally, we measure edge stability, which counts the proportion of $C_\alpha - C_\alpha$ distance that resides with range $[3.65\text{Å}, 3.95\text{Å}]$. The reason for choosing this range is that 99% $C_\alpha - C_\alpha$ distances in ground truth are within this range. We also calculate the mean absolute error (MAE) of the torsion angle. Note that all the geometric evaluations are performed on the original protein backbones without considering augmented nodes.

In Table 1, we summarize the results with respect to these five metrics for protein autoencoders with different downsampling factors. In order to reduce the modeling space of proteins and make it easier for the diffusion model to learn the latent distribution, larger downsampling factors are preferred; but meanwhile, it will become more difficult to achieve good reconstruction results. We can see that $auto-8$ has the worst reconstruction performance because the autoencoder compresses information too much. Although $auto-2$ performs the best among the three settings, the number of nodes in the latent space is still relatively large. So in order to achieve a balance between computation and reconstruction performance, we finally choose $auto-4$ as the pre-trained model for generating latent space data and decoding protein backbones.

4.3 *In-silico* Evaluation

For generated protein structures, we need to evaluate the designability, which means whether we can build amino acid sequences that can fold into desired backbone structures. The most faithful and desirable evaluation is to check through a wet-lab experiment, but this is often resource demanding and not feasible. Here we use *in silico* evaluations as an alternative.

Table 2: Percentage of generated proteins with scTM score > 0.5. Following FoldingDiff and ProtDiff, results are shown within short (50–70) and long (70–128) categories.

Method	50–70	70–128	50–128
ProtDiff	17.1%	8.9%	11.8%
FoldingDiff	27.1%	9.4%	14.2%
FrameDiff	86.6%	87.7%	87.4%
LatentDiff	64.7%	82.8%	66.9%

Specifically, for a generated backbone structure, we first use an inverse folding model, ProteinMPNN [36], to predict eight amino acid sequences that could possibly fold into that backbone structure. OmegaFold [53] is then used to predict folding structures for each amino acid sequence. Next, we adopt TMalign [54] to compute the similarity between the generated backbone structure and each OmegaFold-predicted backbone structure and calculate a TM score to quantify the similarity. The maximum TM-score among these eight scores is referred to as the self-consistency TM-score (scTM). If a scTM score is larger than 0.5, two backbone structures are considered with the same fold and that generated backbone structure is designable.

Similar to previous works [28, 29], we generate 780 backbone structures with various lengths between 50 and 128 and evaluate them by the scTM score, for which the sampling temperature in ProteinMPNN is 0.1. The comparison with FoldingDiff, ProtDiff, and FrameDiff is shown in Table 2. Following ProtDiff, generated proteins are further split into short (50-70) and long (70-128) categories. For our LatentDiff, 66.9% generated structures have their scTM scores > 0.5, which is significantly better than FoldingDiff (14.2%) and ProtDiff (11.8%). Compared with more recent work such as FrameDiff, even though LatentDiff has worse performance in designability, our sampling efficiency is still an advantage, as shown in Table 3. Details about efficiency comparison can be found in Section 4.4. We also visualize some exemplar backbones and OmegaFold-predicted backbone structures using PyMOL [55] in Figure 3.

4.4 Parallel Sampling Efficiency Comparison

In this section, we demonstrate the parallel sampling efficiency of our method. Diffusion models usually need to perform thousands of reverse steps to generate a single data point, and the

Table 3: Sampling efficiency comparison between diffusion models in latent and protein space.[1] LatentDiff-P denotes that no protein autoencoder is used and diffusion is performed directly in the protein space. 1000 proteins are sampled to calculate sampling time.

Method	Parameters	Protein Length	Latent Nodes	Diffusion Steps	Time (hrs)	Speed (sec/sample)
ProtDiff	1.9M	128	N/A	1000	1.9	6.85
FrameDiff	17.4M	128	N/A	500	16.6	60
RFdiffusion	59.8M	128	N/A	200	46.6	168
LatentDiff-P	2.9M	128	N/A	1000	3.9	14.15
LatentDiff	2.9M	128	32	1000	0.25	0.93
LatentDiff	2.9M	128	32	2000	0.51	1.84
LatentDiff	2.9M	256	64	1000	0.95	3.42

data size must be the same during every reverse step. So the generation process is very time-consuming and computationally expensive, especially when the modeling space of diffusion models is large. So this prohibits efficient parallel sampling with limited computing resources. Generation in latent space can reduce memory usage and computational complexity as the latent space is much smaller than the protein space, thereby improving the generation throughput. The reason we compare efficiency in terms of parallel sampling is that a large number of proteins need to be sampled in the screening procedure and high throughput sampling is desired. In this sense, sampling in latent space demonstrates significant efficiency improvement. For the efficiency comparison, we sample 1000 proteins on a single NVIDIA 2080Ti GPU and summarize the result in Table 3. To rule out factors other than different modeling spaces, we also compare with LatentDiff without downsampling (named LatentDiff-P). For our model, the processing time of the decoder is orders of magnitude less than that of our latent diffusion model, so we do not take the decoder time into account. From the result, we can see that the generation time of 1000 protein structures in the protein space is about 3.9 hours, while it only takes about 15 minutes to generate in the latent space and then map to the protein space. Additionally, we also compare the efficiency with FrameDiff and RFdiffusion and we can achieve about 64× and 180× faster generation speed, respectively.[1] Even though the performance still needs to be further improved to compare with recent state-of-the-art methods, the idea of performing diffusion on reduced modeling space already demonstrates potential usefulness in practice. The sampling time of LatentDiff scales linearly with the number of diffusion steps because diffusion steps are performed sequentially. Moreover, since we use a fully connected graph for the diffusion model, increasing latent nodes will quadratically increase memory consumption and computational complexity. Consequently, the sampling throughput will decrease and is contingent upon the GPU memory and computational capacity, with the throughput being constrained by whichever resource reaches its limit first.

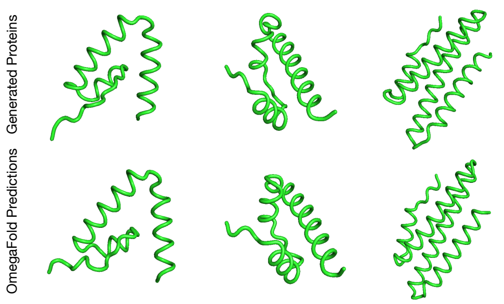


Figure 3: Some samples of generated structures with scTM > 0.5. The top row shows our generated backbones and the second row shows the backbone structures predicted by the OmegaFold from the predicted amino acid sequences.

5 Conclusions

In this work, we have proposed LatentDiff, a 3D latent diffusion framework for protein backbone structure generation. To reduce the modeling space of protein structures, LatentDiff uses a pre-trained equivariant 3D autoencoder to transform protein backbones into a more compact latent space, and models the latent distribution with an equivariant latent diffusion model. LatentDiff is shown to be effective and efficient in generating designable protein backbone structures by comprehensive experimental results.

[1]Note that FrameDiff and RFdiffusion generate full backbone atoms whereas ProtDiff and LatentDiff generate C_α atoms.

References

[1] Xuan Zhang, Limei Wang, Jacob Helwig, Youzhi Luo, Cong Fu, Yaochen Xie, Meng Liu, Yuchao Lin, Zhao Xu, Keqiang Yan, Keir Adams, Maurice Weiler, Xiner Li, Tianfan Fu, Yucheng Wang, Haiyang Yu, YuQing Xie, Xiang Fu, Alex Strasser, Shenglong Xu, Yi Liu, Yuanqi Du, Alexandra Saxton, Hongyi Ling, Hannah Lawrence, Hannes Stärk, Shurui Gui, Carl Edwards, Nicholas Gao, Adriana Ladera, Tailin Wu, Elyssa F. Hofgard, Aria Mansouri Tehrani, Rui Wang, Ameya Daigavane, Montgomery Bohde, Jerry Kurtin, Qian Huang, Tuong Phung, Minkai Xu, Chaitanya K. Joshi, Simon V. Mathis, Kamyar Azizzadenesheli, Ada Fang, Alán Aspuru-Guzik, Erik Bekkers, Michael Bronstein, Marinka Zitnik, Anima Anandkumar, Stefano Ermon, Pietro Liò, Rose Yu, Stephan Günnemann, Jure Leskovec, Heng Ji, Jimeng Sun, Regina Barzilay, Tommi Jaakkola, Connor W. Coley, Xiaoning Qian, Xiaofeng Qian, Tess Smidt, and Shuiwang Ji. Artificial intelligence for science in quantum, atomistic, and continuum systems, 2023.

[2] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International conference on machine learning*, pages 8459–8468. PMLR, 2020.

[3] Jacob Helwig, Xuan Zhang, Cong Fu, Jerry Kurtin, Stephan Wojtowytsch, and Shuiwang Ji. Group equivariant fourier neural operators for partial differential equations. *arXiv preprint arXiv:2306.05697*, 2023.

[4] Dmitrii Kochkov, Tobias Pfaff, Alvaro Sanchez-Gonzalez, Peter Battaglia, and Bryan K Clark. Learning ground states of quantum hamiltonians with graph networks. *arXiv preprint arXiv:2110.06390*, 2021.

[5] Cong Fu, Xuan Zhang, Huixin Zhang, Hongyi Ling, Shenglong Xu, and Shuiwang Ji. Lattice convolutional networks for learning ground states of quantum many-body systems. *arXiv preprint arXiv:2206.07370*, 2022.

[6] Janet R McMillan, Oliver G Hayes, Peter H Winegar, and Chad A Mirkin. Protein materials engineering with dna. *Accounts of chemical research*, 52(7):1939–1948, 2019.

[7] Keqiang Yan, Yi Liu, Yuchao Lin, and Shuiwang Ji. Periodic graph transformers for crystal material property prediction. In *The 36th Annual Conference on Neural Information Processing Systems*, 2022.

[8] Meng Liu, Cong Fu, Xuan Zhang, Limei Wang, Yaochen Xie, Hao Yuan, Youzhi Luo, Zhao Xu, Shenglong Xu, and Shuiwang Ji. Fast quantum property prediction via deeper 2d and 3d graph networks. *arXiv preprint arXiv:2106.08551*, 2021.

[9] Meng Liu, Youzhi Luo, Limei Wang, Yaochen Xie, Hao Yuan, Shurui Gui, Haiyang Yu, Zhao Xu, Jingtun Zhang, Yi Liu, Keqiang Yan, Haoran Liu, Cong Fu, Bora M Oztekin, Xuan Zhang, and Shuiwang Ji. DIG: A turnkey library for diving into graph deep learning research. *Journal of Machine Learning Research*, 22(240):1–9, 2021.

[10] Meng Liu, Youzhi Luo, Kanji Uchino, Koji Maruhashi, and Shuiwang Ji. Generating 3D molecules for target protein binding. In *Proceedings of The 39th International Conference on Machine Learning*, pages 13912–13924, 2022.

[11] Yi Liu, Limei Wang, Meng Liu, Yuchao Lin, Xuan Zhang, Bora Oztekin, and Shuiwang Ji. Spherical message passing for 3D molecular graphs. In *International Conference on Learning Representations*, 2022.

[12] Limei Wang, Yi Liu, Yuchao Lin, Haoran Liu, and Shuiwang Ji. ComENet: Towards complete and efficient message passing for 3D molecular graphs. In *The 36th Annual Conference on Neural Information Processing Systems*, 2022.

[13] Zhengyang Wang, Meng Liu, Youzhi Luo, Zhao Xu, Yaochen Xie, Limei Wang, Lei Cai, Qi Qi, Zhuoning Yuan, Tianbao Yang, and Shuiwang Ji. Advanced graph and sequence neural networks for molecular property prediction and drug discovery. *Bioinformatics*, 38(9):2579–2586, 2022.

[14] Namrata Anand and Possu Huang. Generative modeling for protein structures. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[15] Raphael R Eguchi, Christian A Choe, and Po-Ssu Huang. Ig-VAE: Generative modeling of protein structure by direct 3d coordinate generation. *PLoS computational biology*, 18(6): e1010271, 2022.

[16] Namrata Anand, Raphael Eguchi, and Po-Ssu Huang. Fully differentiable full-atom protein backbone generation, 2019.

[17] Sari Sabban and Mikhail Markovsky. RamaNet: Computational *de novo* helical protein backbone design using a long short-term memory generative neural network. *bioRxiv*, page 671552, 2020.

[18] Shitong Luo, Yufeng Su, Xingang Peng, Sheng Wang, Jian Peng, and Jianzhu Ma. Antigen-specific antibody design and optimization with diffusion-based generative models. *bioRxiv*, 2022.

[19] Chence Shi, Chuanrui Wang, Jiarui Lu, Bozitao Zhong, and Jian Tang. Protein sequence and structure co-design with equivariant translation. *arXiv preprint arXiv:2210.08761*, 2022.

[20] Zachary Wu, Kadina E Johnston, Frances H Arnold, and Kevin K Yang. Protein sequence design with deep generative models. *Current opinion in chemical biology*, 65:18–27, 2021.

[21] Ivan Anishchenko, Samuel J Pellock, Tamuka M Chidyausiku, Theresa A Ramelot, Sergey Ovchinnikov, Jingzhou Hao, Khushboo Bafna, Christoffer Norn, Alex Kang, Asim K Bera, et al. *De novo* protein design by deep network hallucination. *Nature*, 600(7889):547–552, 2021.

[22] Noelia Ferruz, Steffen Schmidt, and Birte Höcker. ProtGPT2 is a deep unsupervised language model for protein design. *Nature communications*, 13(1):4348, 2022.

[23] Donatas Repecka, Vykintas Jauniskis, Laurynas Karpus, Elzbieta Rembeza, Irmantas Rokaitis, Jan Zrimec, Simona Poviloniene, Audrius Laurynenas, Sandra Viknander, Wissam Abuajwa, et al. Expanding functional protein sequence spaces using generative adversarial networks. *Nature Machine Intelligence*, 3(4):324–333, 2021.

[24] Alex Hawkins-Hooker, Florence Depardieu, Sebastien Baur, Guillaume Couairon, Arthur Chen, and David Bikard. Generating functional protein variants with variational autoencoders. *PLoS computational biology*, 17(2):e1008736, 2021.

[25] Ali Madani, Bryan McCann, Nikhil Naik, Nitish Shirish Keskar, Namrata Anand, Raphael R Eguchi, Po-Ssu Huang, and Richard Socher. ProGen: Language modeling for protein generation. *arXiv preprint arXiv:2004.03497*, 2020.

[26] Erik Nijkamp, Jeffrey Ruffolo, Eli N Weinstein, Nikhil Naik, and Ali Madani. ProGen2: exploring the boundaries of protein language models. *arXiv preprint arXiv:2206.13517*, 2022.

[27] Mostafa Karimi, Shaowen Zhu, Yue Cao, and Yang Shen. De novo protein design for novel folds using guided conditional Wasserstein generative adversarial networks. *Journal of chemical information and modeling*, 60(12):5667–5681, 2020.

[28] Brian L Trippe, Jason Yim, Doug Tischer, Tamara Broderick, David Baker, Regina Barzilay, and Tommi Jaakkola. Diffusion probabilistic modeling of protein backbones in 3d for the motif-scaffolding problem. *arXiv preprint arXiv:2206.04119*, 2022.

[29] Kevin E Wu, Kevin K Yang, Rianne van den Berg, James Y Zou, Alex X Lu, and Ava P Amini. Protein structure generation via folding diffusion. *arXiv preprint arXiv:2209.15611*, 2022.

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

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

[32] Minkai Xu, Lantao Yu, Yang Song, Chence Shi, Stefano Ermon, and Jian Tang. GeoDiff: A geometric diffusion model for molecular conformation generation. In *International Conference on Learning Representations*, 2021.

[33] Bowen Jing, Gabriele Corso, Regina Barzilay, and Tommi S Jaakkola. Torsional diffusion for molecular conformer generation. In *ICLR2022 Machine Learning for Drug Discovery*, 2022.

[34] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.

[35] Greg Landrum et al. RDKit: A software suite for cheminformatics, computational chemistry, and predictive modeling.

[36] Justas Dauparas, Ivan Anishchenko, Nathaniel Bennett, Hua Bai, Robert J Ragotte, Lukas F Milles, Basile IM Wicky, Alexis Courbet, Rob J de Haas, Neville Bethel, et al. Robust deep learning–based protein sequence design using ProteinMPNN. *Science*, 378(6615):49–56, 2022.

[37] Youzhi Luo, Keqiang Yan, and Shuiwang Ji. GraphDF: A discrete flow model for molecular graph generation. In *Proceedings of The 38th International Conference on Machine Learning*, pages 7192–7203, 2021.

[38] Meng Liu, Keqiang Yan, Bora Oztekin, and Shuiwang Ji. GraphEBM: Molecular graph generation with energy-based models. In *Energy Based Models Workshop-ICLR 2021*, 2021.

[39] Youzhi Luo and Shuiwang Ji. An autoregressive flow model for 3D molecular geometry generation from scratch. In *International Conference on Learning Representations*, 2022.

[40] Namrata Anand and Tudor Achim. Protein structure and sequence generation with equivariant denoising diffusion probabilistic models. *arXiv preprint arXiv:2205.15019*, 2022.

[41] Jin Sub Lee and Philip M Kim. ProteinSGM: Score-based generative modeling for *de novo* protein design. *bioRxiv*, 2022.

[42] Joseph L Watson, David Juergens, Nathaniel R Bennett, Brian L Trippe, Jason Yim, Helen E Eisenach, Woody Ahern, Andrew J Borst, Robert J Ragotte, Lukas F Milles, et al. Broadly applicable and accurate protein design by integrating structure prediction networks and diffusion generative models. *bioRxiv*, pages 2022–12, 2022.

[43] John Ingraham, Max Baranov, Zak Costello, Vincent Frappier, Ahmed Ismail, Shan Tie, Wujie Wang, Vincent Xue, Fritz Obermeyer, Andrew Beam, et al. Illuminating protein space with a programmable generative model. *bioRxiv*, pages 2022–12, 2022.

[44] Lin Yeqing and AlQuraishi Mohammed. Generating novel, designable, and diverse protein structures by equivariantly diffusing oriented residue clouds. 2023. doi: 10.48550. *arXiv preprint arXiv.2301.12485*.

[45] Jason Yim, Brian L Trippe, Valentin De Bortoli, Emile Mathieu, Arnaud Doucet, Regina Barzilay, and Tommi Jaakkola. Se (3) diffusion model with application to protein backbone generation. *arXiv preprint arXiv:2302.02277*, 2023.

[46] Emiel Hoogeboom, Vıctor Garcia Satorras, Clément Vignac, and Max Welling. Equivariant diffusion for molecule generation in 3d. In *International Conference on Machine Learning*, pages 8867–8887. PMLR, 2022.

[47] Jianyi Yang, Ivan Anishchenko, Hahnbeom Park, Zhenling Peng, Sergey Ovchinnikov, and David Baker. Improved protein structure prediction using predicted interresidue orientations. *Proceedings of the National Academy of Sciences*, 117(3):1496–1503, 2020.

[48] Minkyung Baek, Frank DiMaio, Ivan Anishchenko, Justas Dauparas, Sergey Ovchinnikov, Gyu Rie Lee, Jue Wang, Qian Cong, Lisa N Kinch, R Dustin Schaeffer, et al. Accurate prediction of protein structures and interactions using a three-track neural network. *Science*, 373(6557):871–876, 2021.

[49] Arne Schneuing, Yuanqi Du, Charles Harris, Arian Jamasb, Ilia Igashov, Weitao Du, Tom Blundell, Pietro Lió, Carla Gomes, Max Welling, et al. Structure-based drug design with equivariant diffusion models. *arXiv preprint arXiv:2210.13695*, 2022.

[50] Vıctor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E(n) equivariant graph neural networks. In *International conference on machine learning*, pages 9323–9332. PMLR, 2021.

[51] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, 2021.

[52] Mihaly Varadi, Stephen Anyango, Mandar Deshpande, Sreenath Nair, Cindy Natassia, Galabina Yordanova, David Yuan, Oana Stroe, Gemma Wood, Agata Laydon, et al. AlphaFold protein structure database: massively expanding the structural coverage of protein-sequence space with high-accuracy models. *Nucleic acids research*, 50(D1):D439–D444, 2022.

[53] Ruidong Wu, Fan Ding, Rui Wang, Rui Shen, Xiwen Zhang, Shitong Luo, Chenpeng Su, Zuofan Wu, Qi Xie, Bonnie Berger, et al. High-resolution *de novo* structure prediction from primary sequence. *BioRxiv*, 2022.

[54] Yang Zhang and Jeffrey Skolnick. TM-align: a protein structure alignment algorithm based on the TM-score. *Nucleic acids research*, 33(7):2302–2309, 2005.

[55] Warren L DeLano. The PyMOL molecular graphics system. *http://www. pymol. org*, 2002.

[56] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

[57] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of Adam and beyond. In *International Conference on Learning Representations*, 2018.

[58] Gilles Labesse, N Colloc'h, Joël Pothier, and J-P Mornon. P-SEA: a new efficient assignment of secondary structure from cα trace of proteins. *Bioinformatics*, 13(3):291–295, 1997.

A Appendix

A.1 Distribution Rotation Invariant Reverse Diffusion Process

In this section, we provide proof that by (1) using a high-dimensional Gaussian distribution as the prior distribution, and (2) employing rotation equivariant reverse diffusion model $\boldsymbol{s_\theta}$ [32, 46, 49], the challenge of $p_\theta(\boldsymbol{X}_{\text{down}}, \boldsymbol{H}_{\text{down}}) = p_\theta(R\boldsymbol{X}_{\text{down}}, \boldsymbol{H}_{\text{down}})$ can be addressed. The proof process borrows ideas from Xu et al. [32] and Hoogeboom et al. [46].

First, because $p_\theta(\boldsymbol{X}_T, \boldsymbol{H}_T) = \mathcal{N}(\mathbf{0}, \mathbf{I})$, and $\mathcal{N}(\mathbf{0}, \mathbf{I})$ is isotropic, we have $p_\theta(\boldsymbol{X}_T, \boldsymbol{H}_T) = p_\theta(R\boldsymbol{X}_T, \boldsymbol{H}_T)$, where $R \in \mathbb{R}^{3\times 3}$, $|R| = 1$ describes the rotation transformations in 3D space.

Second, because $\boldsymbol{s_\theta}$ is rotation equivariant for $\boldsymbol{X}_t$ and rotation invariant for $\boldsymbol{H}_t$, and

$$\boldsymbol{X}_{t-1} = \frac{1}{\sqrt{1-\beta_t}}(\boldsymbol{X}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\boldsymbol{s_\theta}(\boldsymbol{X}_t, \boldsymbol{H}_t, t)_{\boldsymbol{X}}) + \sqrt{\beta_t}\boldsymbol{\sigma_X}, \tag{17}$$

$$\boldsymbol{H}_{t-1} = \frac{1}{\sqrt{1-\beta_t}}(\boldsymbol{H}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\boldsymbol{s_\theta}(\boldsymbol{X}_t, \boldsymbol{H}_t, t)_{\boldsymbol{H}}) + \sqrt{\beta_t}\boldsymbol{\sigma_H}, \tag{18}$$

where $\boldsymbol{s_\theta}(\boldsymbol{X}_t, \boldsymbol{H}_t, t)_{\boldsymbol{X}}$ and $\boldsymbol{s_\theta}(\boldsymbol{X}_t, \boldsymbol{H}_t, t)_{\boldsymbol{H}}$ denote the network predictions to update $\boldsymbol{X}$ and $\boldsymbol{H}$, correspondingly. When we apply transformation $R \in \mathbb{R}^{3\times 3}$, $|R| = 1$ to $\boldsymbol{X}_{t-1}$, we will have

$$R\boldsymbol{X}_{t-1} = \frac{1}{\sqrt{1-\beta_t}}R(\boldsymbol{X}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\boldsymbol{s_\theta}(\boldsymbol{X}_t, \boldsymbol{H}_t, t)_{\boldsymbol{X}}) + \sqrt{\beta_t}R\boldsymbol{\sigma_X} \tag{19}$$

$$= \frac{1}{\sqrt{1-\beta_t}}(R\boldsymbol{X}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}R\boldsymbol{s_\theta}(\boldsymbol{X}_t, \boldsymbol{H}_t, t)_{\boldsymbol{X}}) + \sqrt{\beta_t}R\boldsymbol{\sigma_X} \tag{20}$$

$$= \frac{1}{\sqrt{1-\beta_t}}(R\boldsymbol{X}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\boldsymbol{s_\theta}(R\boldsymbol{X}_t, \boldsymbol{H}_t, t)_{\boldsymbol{X}}) + \sqrt{\beta_t}R\boldsymbol{\sigma_X}, \tag{21}$$

and we can have the following

$$\begin{aligned} p_\theta(\boldsymbol{X}_{t-1}, \boldsymbol{H}_{t-1}|\boldsymbol{X}_t, \boldsymbol{H}_t) = p_\theta(\boldsymbol{X}_t, \boldsymbol{H}_t)p(\boldsymbol{\sigma_X}, \boldsymbol{\sigma_H}) &= p_\theta(R\boldsymbol{X}_t, \boldsymbol{H}_t)p(R\boldsymbol{\sigma_X}, \boldsymbol{\sigma_H}) \\ &= p_\theta(R\boldsymbol{X}_{t-1}, \boldsymbol{H}_{t-1}|R\boldsymbol{X}_t, \boldsymbol{H}_t). \end{aligned} \tag{22}$$

Beyond this, for the reverse diffusion time $t \in \{T, T-1, \cdots, 1\}$, assume $p_\theta(\boldsymbol{X}_t, \boldsymbol{H}_t)$ satisfies $p_\theta(\boldsymbol{X}_t, \boldsymbol{H}_t) = p_\theta(R\boldsymbol{X}_t, \boldsymbol{H}_t)$, where $R \in \mathbb{R}^{3\times 3}$, $|R| = 1$ describes the rotation transformations in 3D space. Then we have:

$$\begin{aligned} p_\theta(R\boldsymbol{X}_{t-1}, \boldsymbol{H}_{t-1}) &= \int_{(\boldsymbol{X}_t, \boldsymbol{H}_t)} p_\theta(R\boldsymbol{X}_{t-1}, \boldsymbol{H}_{t-1}|\boldsymbol{X}_t, \boldsymbol{H}_t)p_\theta(\boldsymbol{X}_t, \boldsymbol{H}_t) \\ &= \int_{(\boldsymbol{X}_t, \boldsymbol{H}_t)} p_\theta(R\boldsymbol{X}_{t-1}, \boldsymbol{H}_{t-1}|RR^{-1}\boldsymbol{X}_t, \boldsymbol{H}_t)p_\theta(RR^{-1}\boldsymbol{X}_t, \boldsymbol{H}_t) \\ &= \int_{(\boldsymbol{X}_t, \boldsymbol{H}_t)} p_\theta(\boldsymbol{X}_{t-1}, \boldsymbol{H}_{t-1}|R^{-1}\boldsymbol{X}_t, \boldsymbol{H}_t)p_\theta(R^{-1}\boldsymbol{X}_t, \boldsymbol{H}_t), \end{aligned}$$

let $\boldsymbol{X}' = R^{-1}\boldsymbol{X}_t$, we have $\det R = 1$ and

$$p_\theta(R\boldsymbol{X}_{t-1}, \boldsymbol{H}_{t-1}) == \int_{(\boldsymbol{X}', \boldsymbol{H}_t)} p_\theta(\boldsymbol{X}_{t-1}, \boldsymbol{H}_{t-1}|\boldsymbol{X}', \boldsymbol{H}_t)p_\theta(\boldsymbol{X}', \boldsymbol{H}_t) * \det R = p_\theta(\boldsymbol{X}_{t-1}, \boldsymbol{H}_{t-1}), \tag{23}$$

and $p_\theta(\boldsymbol{X}_{t-1}, \boldsymbol{H}_{t-1})$ is invariant. By induction, $p_\theta(\boldsymbol{X}_{T-1}, \boldsymbol{H}_{T-1}), \ldots, p_\theta(\boldsymbol{X}_0, \boldsymbol{H}_0)$ are all invariant and the proof is complete.

A.2 Datasets

We curate the dataset from Protein Data Bank (PDB) and Swiss-Prot data in AlphaFold Protein Structure Database (AlphaFold DB) [51, 52]. We filter all the single-chain protein data from PDB with $C_\alpha - C_\alpha$ distance less than 5Å and sequence length between 40 and 128 residues, resulting in 4460 protein sequences. We randomly split the data according to 80/10/10 train/validation/test split. In order to include more training data, we further curate protein data from two resources and add them to the current training set. The first part of augmented training data comes AlphaFold DB.

Specifically, we filter single-chain proteins in Swiss-Prot with lengths between 40 and 128 and add these proteins to the training data. The second part of augmented training data comes from PDB, where we curate data from those single-chain proteins with $C_\alpha - C_\alpha$ distance larger than 5Å and sequence lengths longer than 40. Specifically, we split these proteins at the position where $C_\alpha - C_\alpha$ distance is larger than 5Å to obtain protein fragments. Then we add these fragments with lengths between 50 and 128 to the training data. For these fragments with lengths longer than 256, we uniformly cut them into lengths between 50 and 128, and add them to the training data. After this data augmentation process, we can finally obtain about $100k$ training data.

A.3 Experimental Details

For training of the autoencoder, we have used all the available training data. We then use the trained encoder to embed all the training protein data and use their latent representations to train the latent diffusion model. We have trained the autoencoder for 200 epochs with batch size 128, by Adam optimizer [56] with learning rate $1e^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and weight delay $2e^{-4}$. The latent diffusion model has been trained for $16k$ epochs with batch size 2048, by Amsgrad optimizer [57] with learning rate $5e^{-5}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and weight delay $1e^{-12}$. We use 1000 diffusion steps and the same noise scheduler used in Hoogeboom et al. [46]. We implement all the models in PyTorch. The protein autoencoder was trained on a single NVIDIA A100 GPU for 6 days. The latent diffusion model was trained on four NVIDIA A100 GPUs for 7 days.

A.4 Overall Generation Process

To generate a novel protein backbone structure, we first sample multivariate Gaussian noise and use the learned latent diffusion model to generate 3D positions and node embeddings in the latent space. We use low-temperature sampling [43] in the reverse process of the diffusion model. And then we use the pre-trained decoder to generate backbone structures in the protein space. Note that the output of the decoder has a pre-defined fixed size. In order to generate proteins of various lengths, each node in the decoder output is predicted to be an augmented node or not. We simply find the first node that is classified as an augmented node and drop the remaining nodes in the generated protein backbone structure. Note that we do not use reconstructed amino acid types for the corresponding node. Instead, we use the inverse folding model ProteinMPNN [36] to predict protein amino acid sequences from generated backbone structures.

A.5 Secondary Structures

We use P-SEA [58] to count the number of two types of secondary structures in the generated proteins. Specifically, we calculate the percentage of generated proteins that contain only α-helix, only β-sheet, and both α-helix and β-sheet, respectively. The results are shown in Table 4. As seen, more than half of the generated proteins include α-helix, and a large portion of generated proteins contain β-sheet. This proves that our method can successfully generate various secondary structures in natural proteins.

Table 4: Percentage of generated proteins that contain only α-helix, only β-sheet, and both α-helix and β-sheet, respectively.

α-helix only	β-sheet only	α-helix + β-sheet
73.4%	2.2%	23.8%

A.6 Diversity

We also evaluate the diversity of generated proteins with scTM > 0.5 (designable), as shown in Table 5. Specifically, we calculate the TM scores with all other designable proteins for each designable protein and choose the maximum TM score to measure its similarity with the generated proteins. Then, we calculate the average of maximum TM scores over all designable proteins to assess the diversity of the generated proteins (lower is better).

Table 5: Diversity of generated designable proteins (scTM > 0.5). ↓ represents that a lower value indicates better performance.

Method	Diversity$^{\downarrow}$
ProtDiff	0.836±0.1648
FoldingDiff	0.585±0.1276
FrameDiff	0.611±0.1544
LatentDiff	0.634±0.0919

A.7 Latent Space Interpolation

Usually, it is natural to visualize the latent space and perform latent code interpolation to test if the latent space is well-structured. However, a protein in our latent space is not represented by a single latent feature vector, but rather, it is a set of nodes associated with 3D coordinates and node features. As such, it is difficult to use dimension reduction techniques like t-SNE to visualize the latent space. In addition, we did not add a KL-divergence loss on coordinates since it would break equivariance. Even for invariant node features, we only add a minimal KL-divergence penalty to control the variance of the latent space, as we aim to maintain high reconstruction accuracy for the autoencoder. Therefore, in our case, the latent space does not necessarily need to be well-structured, and arbitrary interpolation may not guarantee valid protein structures upon decoding.

To show this, we pick two generated proteins with scTM>0.5 (designable), and their corresponding latent space data are (X^s_{emb}, H^s_{emb}) and (X^t_{emb}, H^t_{emb}). Then we interpolate these two latent space data as $(X^{interp}_{emb}, H^{interp}_{emb}) = (X^s_{emb} * (1-\lambda) + X^t_{emb} * \lambda, H^s_{emb} * (1-\lambda) + H^t_{emb} * \lambda)$. We choose different values of λ and decode the interpolated latent space data into proteins and calculate the scTM score, as shown in Table 6. We can see that if λ is close to 0 or 1, generated proteins are still designable. However, if λ is near 0.5, generated proteins are not valid, just as we analyzed above.

Table 6: The scTM score of proteins decoded from the interpolation of two latent protein representations. λ is the interpolation weights. TM-left means the TM score with the start protein, and TM-right means the TM score with the end protein.

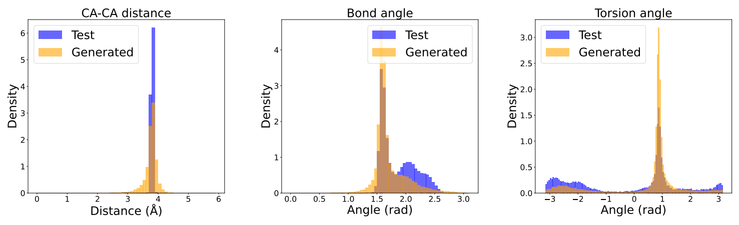
λ	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
scTM	0.86	0.61	0.49	0.48	0.32	0.33	0.27	0.30	0.35	0.62	0.78
TM-left	1.0	0.74	0.57	0.48	0.36	0.29	0.31	0.35	0.40	0.43	0.48
TM-right	0.49	0.51	0.46	0.41	0.37	0.36	0.32	0.39	0.56	0.75	1.0

A.8 Structure and Sequence Co-Design

Since the decoder of the protein autoencoder can predict amino acid types, LatentDiff also possesses the capability to perform structure and sequence co-design, which is a key difference from other protein generation methods. Specifically, we can use decoded sequences as the generated protein sequences instead of predicting sequences from decoded structures using inverse folding methods. The designability result is shown in Table 7. We can see that inverse folding predicted sequences have better alignment with the generated structures than generated sequences. This is because conditionally predicting sequences is easier than jointly generating both structures and sequences. However, even using generated sequences, LatentDiff can achieve similar results with earlier protein generation methods such as ProtDiff.

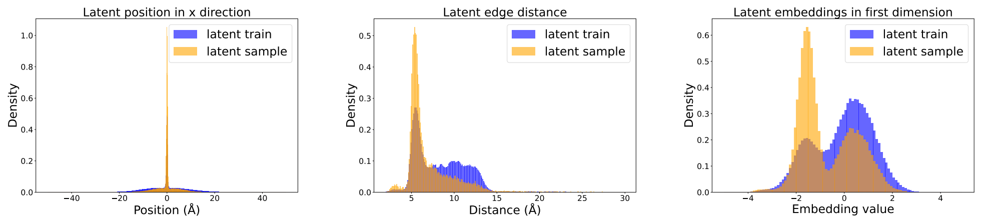
Table 7: Percentage of generated proteins with scTM score > 0.5. Results are shown within short (50–70) and long (70–128) categories.

Method	50–70	70–128	50–128
ProtDiff (use inverse folding sequence)	17.1%	8.9%	11.8%
LatentDiff (use generated sequence)	14.7%	9.7%	14.1%
LatentDiff (use inverse folding sequence)	64.7%	82.8%	66.9%



(a) **(b)** **(c)**

Figure 4: Distribution comparison between generated backbone structures and test set protein backbones. (a) Edge distance between any two consecutive C_α atoms along a protein chain. (b) Bond angle formed by any three consecutive C_α atoms along a protein chain. (c) Torsion angle formed by any four consecutive C_α atoms along a protein chain.



(a) **(b)** **(c)**

Figure 5: Distribution comparison between training data and generated samples in the latent space. (a) Position of latent node in the x direction. (b) Edge distance between any two consecutive nodes in the latent space. (c) First dimension of latent node embeddings.

A.9 Structure Distribution Analysis

Besides showing the success of *in silico* tests, we illustrate the distributions of generated samples in both the original protein space and the latent space. First, we show the edge distance, bond angle, and torsion angle distributions of generated backbones and test set backbones. As shown in Figure 4, the distributions of generated samples are similar to the test distributions. We further investigate the distributions in the latent space. Specifically, we show the distributions of node positions, edge distances, and node embeddings in the latent space. For simplicity, we only show the x coordinate of the latent node position and the first dimension of latent node embeddings. As shown in Figure 5, these distributions of generated latent samples almost recover the latent training data distributions.