Antigen-Specific Antibody Design and Optimization with Diffusion-Based Generative Models for Protein Structures

Shitong Luo[1*], **Yufeng Su**[2*], **Xingang Peng**[3], **Sheng Wang**[4], **Jian Peng**[1,2], **Jianzhu Ma**[1,5]
[1] Helixon Research
[2] University of Illinois Urbana-Champaign
[3] School of Intelligence Science and Technology, Peking University
[4] Paul G. Allen School of Computer Science, University of Washington
[5] Institute for AI Industry Research, Tsinghua University
luost@helixon.com,luost26@gmail.com
swang@cs.washington.edu,jianpeng@illinois.edu,majianzhu@tsinghua.edu.cn

Abstract

Antibodies are immune system proteins that protect the host by binding to specific antigens such as viruses and bacteria. The binding between antibodies and antigens is mainly determined by the complementarity-determining regions (CDR) of the antibodies. In this work, we develop a deep generative model that jointly models sequences and structures of CDRs based on diffusion probabilistic models and equivariant neural networks. Our method is the first deep learning-based method that generates antibodies explicitly targeting specific antigen structures and is one of the earliest diffusion probabilistic models for protein structures. The model is a "Swiss Army Knife" capable of sequence-structure co-design, sequence design for given backbone structures, and antibody optimization. We conduct extensive experiments to evaluate the quality of both sequences and structures of designed antibodies. We find that our model could yield competitive results in binding affinity measured by biophysical energy functions and other protein design metrics.

1 Introduction

Antibodies are important immune proteins generated during an immune response to recognize and neutralize the pathogen [Janeway et al., 2001]. As illustrated in Figure 1a, an antibody contains two heavy chains and two light chains, and their overall structure is similar. Six variable regions determine the specificity of an antibody to the antigens. They are called the Complementarity Determining Regions (CDRs), denoted as H1, H2, H3, L1, L2, and L3. Therefore, the most important step for developing effective therapeutic antibodies is to design CDRs that bind to the specific antigen [Presta, 1992, Akbar et al., 2022a].

Similar to other protein design tasks, the search space of CDRs is vast. A CDR sequence with L amino acids has up to 20^L possible protein sequences. It is not feasible to test all the possible sequences using experimental approaches, so computational methods are needed. Traditional computational approaches rely on sampling protein sequences and structures from complex biophysical energy functions [Pantazes and Maranas, 2010, Lapidoth et al., 2015, Adolf-Bryfogle et al., 2018, Warszawski et al., 2019]. They are generally time-consuming and are prone to get trapped in local optima. Recently, various deep generative models have been developed to design antibodies [Saka et al., 2021, Akbar et al., 2022b, Jin et al., 2022]. Compared to conventional algorithms, deep generative models

[*] Equal contribution.

Preprint.

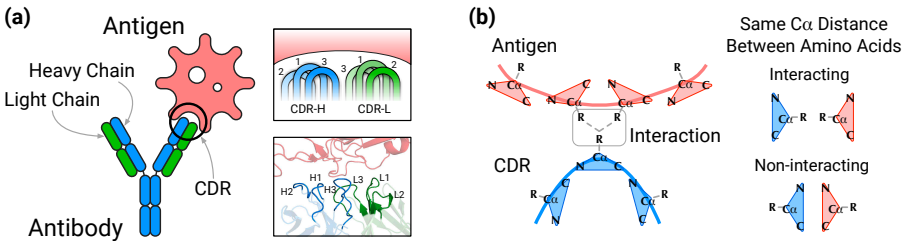

Figure 1: **(a)** Antibody-antigen complex structure and CDR structure. **(b)** The orientations of amino acids (represented by triangles) determine their side-chain orientations, which are key to inter-amino-acid interactions.

could directly capture higher-order interactions among amino acids on antibodies and antigens and generate antibodies more efficiently [Akbar et al., 2022a]. Recently, Jin et al. proposed a generative model for antibody structure-sequence co-design. Their model addresses two important computational challenges: First is how to model the intrinsic relation between CDR sequences and 3D structures, and second is how to model the distribution of CDRs conditional on the rest of the antibody sequence. However, there is still a large gap to fill before generative models become practical for antibody design.

Here, we identify another three challenges for antibody sequence-structure co-design. *First*, the model should be *explicitly conditional on the 3D structures of the antigen* and generate CDRs that fit the antigen structure in the 3D space. This is indispensable for the model to generalize to new antigens. *Second*, the interactions between amino acids are mainly determined by side-chains which are groups of atoms stretching out from the protein backbone (Figure 1b) [Liljas et al., 2016]. Therefore, the model should be able to consider both the *position* and *orientation* of amino acids. *Third*, in drug discovery, pharmacologists collect multiple initial antibodies either from humanized mice or patients [Presta, 1992, Barlow et al., 2018, Warszawski et al., 2019]. Therefore, instead of *de novo* design, the model should be applicable to another realistic scenario: optimizing a particular antibody to increase the binding affinity to the antigen. To the best of our knowledge, no previous machine learning model satisfies all of the above design principles.

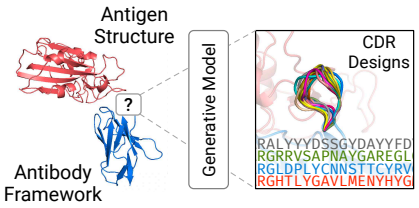

Figure 2: The task in this work is to design CDRs for a given antigen structure and an antibody framework.

To address these challenges, we propose a diffusion-based generative model [Sohl-Dickstein et al., 2015, Song and Ermon, 2019, Ho et al., 2020, Yang et al., 2022] capable of jointly sampling antibody CDR sequences and structures. More importantly, the joint distribution of a CDR sequence and its structure is *directly conditional on antigen structures*. Given a protein complex consisting of an *antigen* and an *antibody framework* as input[2] (as illustrated in Figure 2), we first initialize the CDR with an arbitrary sequence, positions, and orientations. The diffusion model first aggregates information from the antigen and the antibody framework. Then, it iteratively updates the amino acid type, position, and orientation of each amino acid on CDRs. In the last step, we reconstruct the CDR structure at the atom level using side-chain packing algorithms based on the predicted orientations [Alford et al., 2017]. From the perspective of model capability, one of the most important reasons for us to choose the diffusion-based model over other generative models such as generative adversarial networks [Goodfellow et al., 2014] and variational auto-encoders [Kingma and Welling, 2013] is that it generates CDR candidates iteratively in the sequence-structure space so that we can interfere and impose constraints on the sampling process to support a broader range of design tasks.

We summarize our contributions as follows:

- We propose the first deep learning models to perform antibody sequence-structure design by considering the 3D structures of the antigen.
- In our model, we not only design protein sequences and coordinates but also side-chain orientations (represented as $\mathrm{SO}(3)$ element) of each amino acid. It is the first deep learning

[2]The structure of the antigen-antibody framework can be obtained either from existing antigen-antibody structure or by docking an initial antibody to the target antigen.

model that could achieve atomic-resolution antibody design and is equivariant to rotation and translation.
- We show that our model can be applied to a wide range of antibody design tasks, including sequence-structure co-design, fix-backbone CDR design, and antibody optimization.

2 Related Work

Computational Antibody Design Conventional computational approaches are mainly based on sampling algorithms over hand-crafted and statistical energy functions and iteratively modify protein sequences and structures [Adolf-Bryfogle et al., 2018, Lapidoth et al., 2015, Warszawski et al., 2019, Pantazes and Maranas, 2010, Ruffolo et al., 2021]. These methods are inefficient and prone to getting stuck at local optima due to the rough energy landscape. In recent years, deep learning methods have shown potential in antibody design by using language models to generate protein sequences [Alley et al., 2019, Shin et al., 2021, Saka et al., 2021, Akbar et al., 2022b]. Although much more efficient, the sequence-based methods can only generate new antibodies based on previously observed antibodies but can hardly generate antibodies for specific antigen structures.

Jin et al. proposed the first CDR sequence-structure co-design deep generative model which focuses on designing antibodies to neutralize SARS-CoV-2. *It relies on an additional antigen-specific predictor to predict the neutralization of the designed antibodies, which is not generalizable to arbitrary antigens*. In comparison to their model, we explicitly model the 3D structure of an antigen, opening the door to generalizing the prediction to unseen antigens with solved 3D structures. Another advantage of our model is that we consider not only backbone atom coordinates but also the orientation of amino acids. The orientation is critical to protein-protein interactions as most of the atoms interacting between antibodies and antigens are in the side-chain [Liljas et al., 2016] (as illustrated in Figure 1b). Lastly, the model proposed by Jin et al. is not equivariant by construction, which is fundamental in molecular modeling.

Protein Structure Prediction Protein structure prediction algorithms take protein sequences and Multiple Sequence Alignments (MSAs) as input and translate them to 3D structures [Jumper et al., 2021, Baek et al., 2021, Yang et al., 2020]. Accurate protein structure prediction models predict not only the position of amino acids but also their orientation [Jumper et al., 2021, Yang et al., 2020]. The orientation of amino acids determines the direction in which its side chain stretches, so it is indispensable for reconstructing full-atom structures. AlphaFold2 [Jumper et al., 2021] predicts per-amino-acid orientations in an iterative fashion, similar to our proposed model. However, it is not generative, unable to efficiently sample diverse structures for protein design. Recently, based on prior protein structure prediction algorithms, methods for predicting antibody CDR structures have emerged [Ruffolo et al., 2022b,a], but they are not able to design CDR sequences.

Diffusion-Based Generative Models Diffusion probabilistic models learn to generate data via denoising samples from a prior distribution [Sohl-Dickstein et al., 2015, Song and Ermon, 2019, Ho et al., 2020]. Recently, progress has been made in developing equivariant diffusion models for molecular 3D structures [Shi et al., 2021, Hoogeboom et al., 2022, Jing et al., 2022]. Atoms in a molecule do not have natural orientations, so the generation process differs from generating protein structures. Diffusion models have also been extended to non-Euclidean data, such as data in the Riemannian manifolds [Leach et al., 2022, De Bortoli et al., 2022]. These models are relevant to modeling orientations which are represented by elements in $\mathrm{SO}(3)$. In addition, diffusion models can also be used to generate discrete categorical data [Hoogeboom et al., 2021, Austin et al., 2021].

3 Methods

This section is organized as follows: Section 3.1 introduces notations used throughout the paper and formally states the problem. Section 3.2 formulates the diffusion process for modeling antibodies. Section 3.3 introduces details about the neural network parameterization for the diffusion processes. Section 3.4 presents sampling algorithms for various antibody design tasks.

3.1 Definitions and Notations

An amino acid in a protein complex can be represented by its type, C_α atom coordinate, and the orientation, denoted as $s_i \in \{\texttt{ACDEFGHIKLMNPQRSTVWY}\}, \boldsymbol{x}_i \in \mathbb{R}^3, \boldsymbol{O}_i \in \mathrm{SO}(3)$, respectively. Here $i = 1 \dots N$, and N is the number of amino acids in the protein complex[3].

In this work, we assume the antigen structure and the antibody framework is given (Figure 2), and we focus on designing CDRs on the antibody framework. Assume the CDR to be generated has m amino acids with index from $l+1$ to $l+m$. They are denoted as $\mathcal{R} = \{(\mathrm{s}_j, \mathbf{x}_j, \mathbf{O}_j) \mid j = l+1, \dots, l+m\}$. Formally, our goal is to jointly model the distribution of $\mathcal{R}$ given the structure of the antibody-antigen complex $\mathcal{C} = \{(s_i, \boldsymbol{x}_i, \boldsymbol{O}_i) \mid i \in \{1 \dots N\} \backslash \{l+1, \dots, l+m\}\}$.

3.2 Diffusion Processes

A diffusion probabilistic model defines two Markov chains of diffusion processes. The forward diffusion process gradually adds noise to the data until the data distribution approximately reaches the prior distribution. The generative diffusion process starts from the prior distribution and iteratively transforms it to the desired distribution. Training the model relies on the forward diffusion process to simulate the noisy data. Let $\left(\mathrm{s}_j^t, \mathbf{x}_j^t, \mathbf{O}_j^t\right)$ denote the intermediate state of amino acid j at time step t. $\mathcal{R}^t = \{\mathrm{s}_j^t, \mathbf{x}_j^t, \mathbf{O}_j^t\}_{j=l+1}^{l+m}$ represents the sequence and structure sampled at step t. $t = 0$ represents the state of real data (observed sequences and structures of CDRs) and $t = T$ represents samples from the prior distribution. Forward diffusion goes from $t = 0$ to T, and generative diffusion proceeds in the opposite way. The diffusion processes for amino acid types s_j^t, coordinates $\mathbf{x}_j^t$, and orientations $\mathbf{O}_j^t$ are defined as follows:

Multinomial Diffusion for Amino Acid Types The forward diffusion process for amino acid types is based on the multinomial distribution defined as follows [Hoogeboom et al., 2021]:

$$q(\mathrm{s}_j^t | \mathrm{s}_j^{t-1}) = \mathrm{Multinomial}\left((1 - \beta_{\text{type}}^t) \cdot \texttt{onehot}(\mathrm{s}_j^{t-1}) + \beta_{\text{type}}^t \cdot \frac{1}{20} \cdot \mathbf{1}\right), \tag{1}$$

where `onehot` represents a function that converts amino acid type to a 20-dimensional one-hot vector and $\mathbf{1}$ is an all-one vector. β_{type}^t is the probability of resampling another amino acid over 20 types uniformly. When $t \to T$, β_{type}^t is set close to 1 and the distribution is closer to the uniform distribution. The following probability density provides an efficient way to perturb s_j^0 for timestep t during training [Hoogeboom et al., 2021]:

$$q(\mathrm{s}_j^t | \mathrm{s}_j^0) = \mathrm{Multinomial}\left(\bar{\alpha}_{\text{type}}^t \cdot \texttt{onehot}(\mathrm{s}_j^0) + (1 - \bar{\alpha}_{\text{type}}^t) \cdot \frac{1}{20} \cdot \mathbf{1}\right), \tag{2}$$

where $\bar{\alpha}_{\text{type}}^t = \prod_{\tau=1}^t (1 - \beta_{\text{type}}^\tau)$.

The generative diffusion process is defined as:

$$p(\mathrm{s}_j^{t-1} | \mathcal{R}^t, \mathcal{C}) = \mathrm{Multinomial}\left(F(\mathcal{R}^t, \mathcal{C})[j]\right), \tag{3}$$

where $F(\cdot)[j]$ is a neural network model taking the structure context (antigen and antibody framework) and the CDR state from the previous step as input and predicts the probability of the amino acid type for the j-th amino acid on the CDR. Note that, different from the forward diffusion process, the generative diffusion process must rely on the structure context $\mathcal{C}$ and the CDR state of the previous step including positions and orientations. The main difference between these two processes is that the forward diffusion process adds noise to data so it is irrelevant to data or contexts but the generative diffusion process depends on the given condition and full observation of the previous step. The generative diffusion process needs to approximate the posterior $q(\mathrm{s}_j^{t-1} | \mathrm{s}_j^t, \mathrm{s}_j^0)$ derived from Eq.1 and Eq.2 to denoise. Therefore, the objective of training the generative diffusion process for amino acid types is to minimize the expected KL divergence between Eq.3 and the posterior distribution:

$$L_{\text{type}}^t = \mathbb{E}_{\mathcal{R}^t \sim p}\left[\frac{1}{m} \sum\nolimits_j D_{\mathrm{KL}}\left(q(\mathrm{s}_j^{t-1} | \mathrm{s}_j^t, \mathrm{s}_j^0) \middle\| p(\mathrm{s}_j^{t-1} | \mathcal{R}^t, \mathcal{C})\right)\right]. \tag{4}$$

[3]Note that a protein complex contains more than one chain, so N is not the length of one protein but is the sum of the lengths of all chains in the complex.

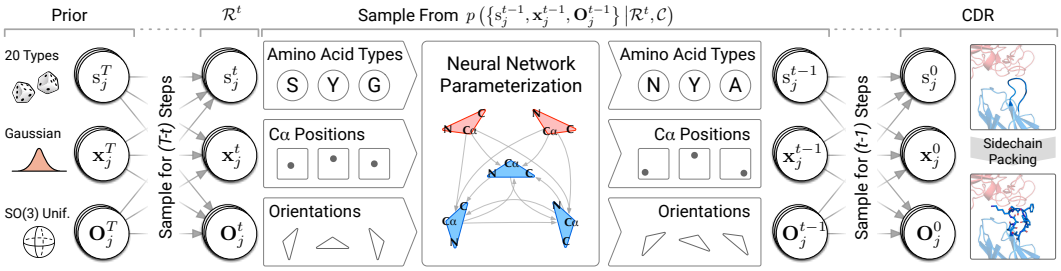

Figure 3: Illustration of the generative diffusion process. At each step, the network takes the current CDR state as input and parameterizes the distribution of the CDR's sequences, positions, and orientations for the next step. In the end, full-atom structures are constructed by the side-chain packing algorithm.

Diffusion for C_α Coordinates As the coordinate of an atom could be an arbitrary value, we scale and shift the coordinates of the whole structure such that the distribution of atom coordinates roughly match the standard normal distribution. We define the forward diffusion for the normalized C_α coordinate $\mathbf{x}_j$ as follows:

$$q\left(\mathbf{x}_j^t \mid \mathbf{x}_j^{t-1}\right) = \mathcal{N}\left(\mathbf{x}_j^t \middle| \sqrt{1-\beta_{\text{pos}}^t} \cdot \mathbf{x}_j^{t-1}, \beta_{\text{pos}}^t \boldsymbol{I}\right), \tag{5}$$

$$q\left(\mathbf{x}_j^t \mid \mathbf{x}_j^0\right) = \mathcal{N}\left(\mathbf{x}_j^t \middle| \sqrt{\bar{\alpha}_{\text{pos}}^0} \cdot \mathbf{x}_j^0, (1-\bar{\alpha}_{\text{pos}}^0) \boldsymbol{I}\right), \tag{6}$$

where β_{pos}^t controls the rate of diffusion and its value increases from 0 to 1 as time step goes from 0 to t, and $\bar{\alpha}_{\text{pos}}^t = \prod_{\tau=1}^{t}(1-\beta_{\text{pos}}^\tau)$. Using the reparameterization trick proposed by Ho et al., the generative diffusion process is defined as:

$$p\left(\mathbf{x}_j^{t-1} \middle| \mathcal{R}^t, \mathcal{C}\right) = \mathcal{N}\left(\mathbf{x}_j^{t-1} \middle| \boldsymbol{\mu}_p\left(\mathcal{R}^t, \mathcal{C}\right), \beta_{\text{pos}}^t \boldsymbol{I}\right), \tag{7}$$

$$\boldsymbol{\mu}_p\left(\mathcal{R}^t, \mathcal{C}\right) = \frac{1}{\sqrt{\alpha_{\text{pos}}^t}}\left(\mathbf{x}_j^t - \frac{\beta_{\text{pos}}^t}{\sqrt{1-\bar{\alpha}_{\text{pos}}^t}} G(\mathcal{R}^t, \mathcal{C})[j]\right). \tag{8}$$

Here, $G(\cdot)[j]$ is a neural network that predicts the standard Gaussian noise $\epsilon_j \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ added to $\sqrt{\bar{\alpha}_{\text{pos}}^0}\mathbf{x}_j^0$ (scaled coordinate of amino acid j) based on the reparameterization of Eq.6: $\mathbf{x}_j^t = \sqrt{\bar{\alpha}_{\text{pos}}^0}\mathbf{x}_j^0 + \sqrt{1-\bar{\alpha}_{\text{pos}}^0}\epsilon_j$. The objective function of training the generative process is the expected MSE between G and ϵ_j, which is simplified from aligning distribution p to the posterior $q(\mathbf{x}_j^{t-1} \mid \mathbf{x}_j^t, \mathbf{x}_j^0)$ [Ho et al., 2020]:

$$L_{\text{pos}}^t = \mathbb{E}\left[\frac{1}{m}\sum\nolimits_j \left\|\epsilon_j - G(\mathcal{R}^t, \mathcal{C})\right\|^2\right]. \tag{9}$$

SO(3) Denoising for Amino Acid Orientations We empirically formulate an iterative perturb-denoise scheme for learning and generating amino acid orientations represented by SO(3) elements [Leach et al., 2022]. Note that we do not use the term *diffusion* because the formulation does not strictly follow the framework of diffusion probabilistic models though the overall principle is the same. Similar to the typical diffusion process, the distribution of orientations perturbed for t steps is defined as, according to Leach et al. [2022]:

$$q\left(\mathbf{O}_j^t \mid \mathbf{O}_j^0\right) = \mathcal{IG}_{\text{SO(3)}}\left(\mathbf{O}_j^t \middle| \text{ScaleRot}\left(\sqrt{\bar{\alpha}_{\text{ori}}^t}, \mathbf{O}_j^0\right), 1-\bar{\alpha}_{\text{ori}}^t\right). \tag{10}$$

$\mathcal{IG}_{\text{SO(3)}}$ denotes the isotropic Gaussian distribution on $\text{SO}(3)$ parameterized by a mean rotation and a scalar variance [Leach et al., 2022, Matthies et al., 1970, Nikolayev and Savyolov, 1970]. ScaleRot modifies the rotation matrix by scaling its rotation angle with the rotation axis fixed [Gallier and Xu,

2003]. $\bar{\alpha}^t_{\text{ori}} = \prod_{\tau=1}^{t}(1-\beta^\tau_{\text{ori}})$, where β^t_{ori} is the variance increases with the step t. The conditional distribution used for the generation process of orientations is defined as:

$$p\left(\mathbf{O}_j^{t-1}\middle|\mathcal{R}^t,\mathcal{C}\right) = \mathcal{IG}_{\text{SO(3)}}\left(\mathbf{O}_j^{t-1}\middle|H(\mathcal{R}^t,\mathcal{C})[j],\beta^t_{\text{ori}}\right), \tag{11}$$

where $H(\cdot)[j]$ is a neural network that denoises the orientation and outputs the denoised orientation matrix of amino acid j. Training the conditional distribution requires aligning the predicted orientation from $H(\cdot)$ to the real orientation. Hence, we formulate the training object that minimizes the expected discrepancy measured by the inner product between the real and the predicted orientation matrices:

$$L^t_{\text{ori}} = \mathbb{E}\left[\frac{1}{m}\sum\nolimits_j \left\|\left(\mathbf{O}_j^0\right)^\intercal \widehat{\mathbf{O}}_j^{t-1} - \boldsymbol{I}\right\|_F^2\right], \tag{12}$$

where $\widehat{\mathbf{O}}_j^{t-1} = H(\cdot)[j]$ is the predicted orientation for amino acid j.

The Overall Training Objective By summing Eq.4, 9, and 12 and taking the expectation w.r.t. t, we obtain the final training objective function:

$$L = \mathbb{E}_{t\sim\text{Uniform}(1\ldots T)}\left[L^t_{\text{type}} + L^t_{\text{pos}} + L^t_{\text{ori}}\right]. \tag{13}$$

To train the model, we first sample a time step t and then sample noisy states $\{\mathrm{s}_j^t, \mathbf{x}_j^t, \mathbf{O}_j^t\}_{j=l+1}^{l+m} \sim p$ by adding noise to the training sample using the diffusion process defined by Eq.2, 6, and 10. We compute the loss using the noisy data and backpropagate the loss to update model parameters.

3.3 Parameterization with Neural Networks

In this section, we briefly introduce the neural network architectures used in different components of the diffusion process. The purpose of the networks is to encode the CDR state at a time step t along with the context structure: $\{\mathrm{s}_j^t, \mathbf{x}_j^t, \mathbf{O}_j^t\}_{j=l+1}^{l+m} \cup \{s_i^t, \boldsymbol{x}_i^t, \boldsymbol{O}_i^t\}_{i=\{1\ldots N\}\setminus\{l+1\ldots l+m\}}$, and then denoises the CDR amino acid types (F), positions (G), and orientations (H).

First, we adopt Multiple Layer Perceptrons (MLPs) to generate embeddings for single and pairs of amino acids. The single amino-acid embedding MLP creates vector $\boldsymbol{e}_i$ for amino acid i, which encodes the information of amino acid types, torsional angles, and 3D coordinates of all the heavy atoms. The pairwise embedding MLP encodes the Euclidean distances and dihedral angles between amino acid i and j to feature vectors $\boldsymbol{z}_{ij}$. We adopt IPA [Jumper et al., 2021], an orientation-aware roto-translation invariant network to transform $\boldsymbol{e}_i$ and $\boldsymbol{z}_{ij}$ into hidden representations $\boldsymbol{h}_i$, which aims to represent the amino acid itself and its environment. Next, the representations are fed to three different MLPs to denoise the amino acid types, 3D positions, and orientations of the CDR, respectively.

In particular, the MLP for denoising amino acid types outputs a 20-dimensional vector representing the posterior probabilities. The MLP for denoising C_α coordinates predicts the scaled change of the coordinate in terms of the current orientation of the amino acid. As the coordinate deviation is calculated in the local frame, we left-multiply it by the orientation matrix and transform it back to the global frame. Formally, this can be expressed as $\hat{\epsilon}_j = \mathbf{O}_j^t\,\text{MLP}_G\left(\boldsymbol{h}_j\right)$. Predicting coordinate deviations in the local frame and projecting it to the global frame ensures the equivariance of the prediction, as when the entire 3D structure rotates by a particular angle, the coordinate deviations also rotate by the same angle. The MLP for denoising orientations first predicts a $so(3)$ vector [Gallier and Xu, 2003]. The vector is converted to a rotation matrix $\boldsymbol{M}_j \in \text{SO(3)}$ right-multiplied to the orientation to produce a new mean orientation for the next generative step: $\widehat{\mathbf{O}}_j^{t-1} \leftarrow \mathbf{O}_j^t\boldsymbol{M}_j$. The proposed networks are equivariant to the rotation and translation of the overall structure:

Proposition 1. *For any proper rotation matrix $\boldsymbol{R} \in \text{SO(3)}$ and any 3D vector $\boldsymbol{r} \in \mathbb{R}^3$ (rigid transformation $(\boldsymbol{R},\boldsymbol{r}) \in \text{SE(3)}$), F, G and H satisfy the following equivariance properties:*

$$F(\boldsymbol{R}\mathcal{R}^t + \boldsymbol{r}, \boldsymbol{R}\mathcal{C} + \boldsymbol{r}) = F(\mathcal{R}^t,\mathcal{C}), \tag{14}$$

$$G(\boldsymbol{R}\mathcal{R}^t + \boldsymbol{r}, \boldsymbol{R}\mathcal{C} + \boldsymbol{r}) = \boldsymbol{R}G(\mathcal{R}^t,\mathcal{C}), \tag{15}$$

$$H(\boldsymbol{R}\mathcal{R}^t + \boldsymbol{r}, \boldsymbol{R}\mathcal{C} + \boldsymbol{r}) = \boldsymbol{R}H(\mathcal{R}^t,\mathcal{C}), \tag{16}$$

where $\boldsymbol{R}\mathcal{R}^t+\boldsymbol{r} := \{\mathrm{s}_j^t, \mathbf{x}_j^t+\boldsymbol{r}, \boldsymbol{R}\mathbf{O}_j^t\}_{j=l+1}^{l+m}$ and $\boldsymbol{R}\mathcal{C}+\boldsymbol{r} := \{s_i, \boldsymbol{x}_i+\boldsymbol{r}, \boldsymbol{R}\boldsymbol{O}_i\}_{i\in\{1\ldots N\}\setminus\{l+1,\ldots,l+m\}}$ denote the rotated and translated structure. Note that F, G, and H are not single MLPs. Each of them includes the shared encoder and a specific MLP.

3.4 Sampling Algorithms

The sampling algorithm first samples amino acid types from the uniform distribution over 20 classes: $\mathrm{s}_j^T \sim \mathrm{Uniform}(20)$, C_α positions from the standard normal distribution: $\mathbf{x}_j^T \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_3)$, and orientations from the uniform distribution over $\mathrm{SO}(3)$: $\mathbf{O}_j^T \sim \mathrm{Uniform}(\mathrm{SO}(3))$. Note that we normalize the coordinates of the structure in the same way as training such that C_α positions in the CDR roughly follow the standard normal distribution. Next, we iteratively sample sequences and structures from the generative diffusion kernel by denoising amino acid types, C_α coordinates, and orientations until $t = 0$. To build a full atom 3D structure, we construct the coordinates of N, C_α, C, O, and side-chain C_β (except glycine that does not have C_β) according to their ideal local coordinates relative to the C_α position and orientation of each amino acid [Engh and Huber, 2012]. Based on the five reconstructed atoms, the rest of the side-chain atoms are constructed using the side-chain packing function implemented in Rosetta [Alford et al., 2017]. In the end, we adopt the AMBER99 force field [Lindorff-Larsen et al., 2010] in OpenMM [Eastman et al., 2017] to refine the full atom structure.

In addition to the joint design of sequences and structures, we can constrain partial states for other design tasks. For example, by fixing the backbone structure (positions and orientations) and sampling only sequences, we can do **fix-backbone sequence design**. Another usage is to **optimize an existing antibody**. Specifically, we first add noise to the existing antibody for t steps and denoise the perturbed antibody sequence starting from the t-th step of the generative diffusion process.

4 Experiments

We present the application of our model, named **DiffAb**[4], in three antibody design tasks: sequence-structure co-design (Section 4.1), antibody sequence design based on antibody backbones (Section 4.2), and antibody optimization (Section 4.3). In Section 4.4, we show how to use our model without known antibody frameworks bound to the antigen.

4.1 Sequence-Structure Co-design

The dataset for training the model is derived from the SAbDab database[Dunbar et al., 2014]. We first remove structures whose resolution is worse than 4Å and discard antibodies targeting non-protein antigens. We cluster antibodies in the database according to CDR-H3 sequences at 50% sequence identity. We manually select five clusters as the test set, containing 19 antibody-antigen complexes in total. The test set includes antigens from several well-known pathogens including SARS-CoV-2, MERS, influenza, and so on. Structures in the remaining clusters are used for training.

To evaluate the performance, we remove the original CDR from the antibody-antigen complex in the test set and sample both the sequence and structure of the removed region. We set the length of the CDR to be identical to the length of the original CDR for simplicity. In practice, one can enumerate different lengths of CDRs. We compare our model to RosettaAntibodyDesign (RAbD) [Adolf-Bryfogle et al., 2018], an antibody design software based on Rosetta energy functions. For each model, we draw 100 samples for each CDR. Both the original structures and designed structures from different methods are refined by OpenMM and Rosetta.

We use the following metrics to evaluate designed antibodies: (1) IMP: is the percentage of designed CDRs with lower (better) binding energy (ΔG) than the original CDR. The binding energy is calculated by `InterfaceAnalyzer` in the Rosetta software package [Alford et al., 2017]. (2) RMSD: is the C_α root-mean-square deviation (RMSD) between the generated structure and the original structure with *only antibody frameworks aligned*. (3) AAR: is the amino acid recovery rate measured by the sequence identity between the reference CDR sequences and the generated sequences [Adolf-Bryfogle et al., 2018]. Note that different from Jin et al. [2022], we do not use neutralization prediction models because they are sequence-based and are specified to a limited class of antigens, which deviates from our goal of developing a general antibody design model.

Table 1 shows that our model (DiffAb) recovers CDR sequences more accurately than RAbD (higher AAR). The RMSDs of CDRs generated by DiffAb are higher in CDR-H3, which indicates that our generated samples are more diverse structurally. The IMP score of DiffAb is on par with RAbD in

[4]Code and data are available at `https://github.com/luost26/diffab`.

Table 1: Evaluation of the generated antibody CDRs (sequence-structure co-design) by RAbD and our DiffAb model.

CDR	Method	AAR	RMSD	IMP	CDR	Method	AAR	RMSD	IMP
H1	RAbD	22.85%	2.261Å	43.88%	L1	RAbD	34.27%	1.204Å	46.81%
	DiffAb	65.75%	1.188Å	53.63%		DiffAb	56.67%	1.388Å	45.58%
H2	RAbD	25.50%	1.641Å	53.50%	L2	RAbD	26.30%	1.767Å	56.94%
	DiffAb	49.31%	1.076Å	29.84%		DiffAb	59.32%	1.373Å	49.95%
H3	RAbD	22.14%	2.900Å	23.25%	L3	RAbD	20.73%	1.624Å	55.63%
	DiffAb	26.78%	3.597Å	23.63%		DiffAb	46.47%	1.627Å	47.32%

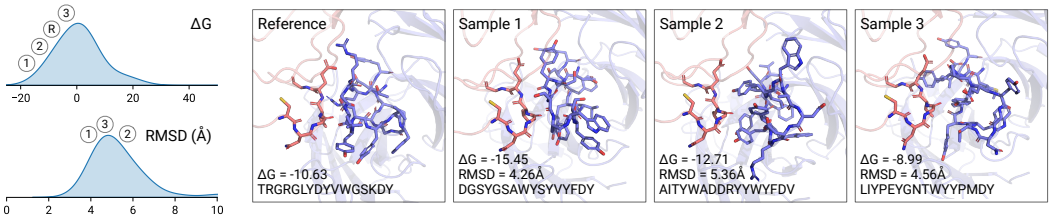

Figure 4: Examples of CDR-H3 designed by the sequence-structure co-design method and the distribution of their interaction energy and RMSD. The antigen-antibody template is derived from PDB:7chf, where the antigen is SARS-CoV-2 RBD. Sample 1 has better complementarity to the antigen while Sample 3 fits the antigen worse. This could explain their difference in the binding energy (ΔG).

CDR-H3, and lower in other CDRs. However, it should be noted that RAbD optimizes the Rosetta energy function, which is also used for evaluation. Our model achieves reasonably good binding energy without explicit supervision signal from Rosetta energy functions. Figure 4 presents three generated examples of CDR-H3 targeting SARS-CoV-2 RBD. Sample 1 has the lowest binding energy and it can be observed that it has better complementarity to the antigen. The binding energy of Sample 3 is higher than the original one and visually, the shape of the CDR does not fit the antigen well.

4.2 Fix-Backbone Sequence Design and Structure Prediction

In this setting, the backbone structure of CDRs is given and we only need to design the CDR sequence, which transforms the task into a constrained sampling problem. Fix-backbone design is a common setting in the area of protein design [Ingraham et al., 2019, Hsu et al., 2022, Anishchenko et al., 2021, Strokach et al., 2020, Tischer et al., 2020]. For this task, we consider **FixBB**, a Rosetta-based sequence design software given CDR backbone structure, as the baseline. We use the AAR metric introduced in Section 4.1 to evaluate the designed CDRs.

As shown in Table 2, our model achieves better AAR in all the CDRs. This shows that our model is also powerful in modeling the conditional probability of sequences given backbone structures. Admittedly, the training data is clustered only by CDR-H3 sequences, so the model might have seen other CDRs in the test set during training, leading to even higher AAR. However, we believe this is not an issue as CDRs other than H3 are generally conserved and contribute less to the specificity Xu and Davis [2000].

Our model can predict CDR structures by fixing the sequence. Table 3 shows that it accurately predicts the structure of CDR H1, H2, L1, L2, and L3 (RMSD $\leq$ 1.5Å). The accuracy of CDR-H3 prediction is lower due to the high variability. Figure 5a separately shows the accuracy of different CDR-H3 lengths. The prediction is generally more accurate for shorter ones. When the CDR-H3 contains more than 10 amino acids, the prediction accuracy drops.

Table 2: Comparison of FixBB and DiffAb in terms of amino acid recovery (AAR) in the fix-backbone CDR design task. DiffAb achieves higher AAR. The AAR of DiffAb on CDR-H3 is lower than other CDRs since H3 is much more versatile.

CDR	Method	AAR	CDR	Method	AAR
H1	FixBB	37.14%	L1	FixBB	33.80%
	DiffAb	87.83%		DiffAb	86.63%
H2	FixBB	43.08%	L2	FixBB	28.54%
	DiffAb	79.70%		DiffAb	88.91%
H3	FixBB	30.74%	L3	FixBB	17.92%
	DiffAb	59.48%		DiffAb	78.69%

Table 3: The accuracy of CDR structures predicted by DiffAb in RMSD.

CDR	RMSD
H1	0.901Å
H2	1.044Å
H3	3.246Å
L1	1.365Å
L2	1.321Å
L3	1.492Å

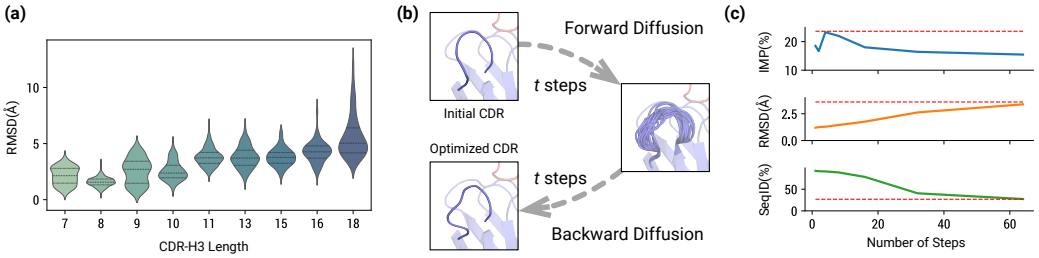

Figure 5: **(a)** RMSD of predicted CDR-H3 structures grouped by lengths. **(b)** The antibody optimization algorithm first perturbs the initial CDR for t steps using the forward diffusion process and then denoises it by the backward diffusion process into the optimized CDR. **(c)** IMP, RMSD, and SeqID of the CDRs optimized with different numbers of steps. Dashed lines represent the results of *de novo* design. When $t = 4$, the optimized CDRs reach an IMP score close to *de novo* CDRs but remain structurally similar to the original one.

4.3 Antibody Optimization

We use our model to optimize existing antibodies which is another common pharmaceutical application. To optimize an antibody, we *first perturb the CDR sequence and structure* for t steps using the forward diffusion process. Then, we denoise the sequences starting from the $(T - t)$-th step (t steps remaining) of the generative diffusion process and obtain a set of optimized antibodies. This process is illustrated in Figure 5b. We optimize CDR-H3 of the antibodies in the test set with various t values. For each antibody and t, we perturb the CDR independently 100 times and collect 100 optimized CDRs different from the original CDR. We report the percentage of optimized antibodies with improved binding energy (IMP), RMSD, and sequence identity (SeqID) of the optimized CDR in comparison to the original antibody. We also compare the optimized antibodies with the *de novo* ($t = T = 100$) designed antibodies introduced in Section 4.1. As shown in Table 4 and Figure 5c, the optimization method could produce antibodies with improved binding energy measured by the Rosetta energy function. In contrast to redesigning CDRs, optimization improves binding energy while keeping the optimized CDR similar to the original one, which is desired in many practical applications.

Table 4: Evaluation of optimized CDR-H3s with different numbers of optimization steps. In contrast to redesigning the CDR, the optimization method can improve binding energy while *keeping the optimized CDR similar to the original one*. Figure 5c shows the line plot of the results.

t	IMP	RMSD	SeqID
1	18.52%	1.194Å	92.42%
2	16.67%	1.252Å	91.61%
4	23.29%	1.290Å	91.16%
8	22.01%	1.447Å	88.78%
16	18.02%	1.759Å	78.43%
32	16.43%	2.623Å	40.58%
64	15.47%	3.380Å	27.30%
T	23.63%	3.597Å	26.78%

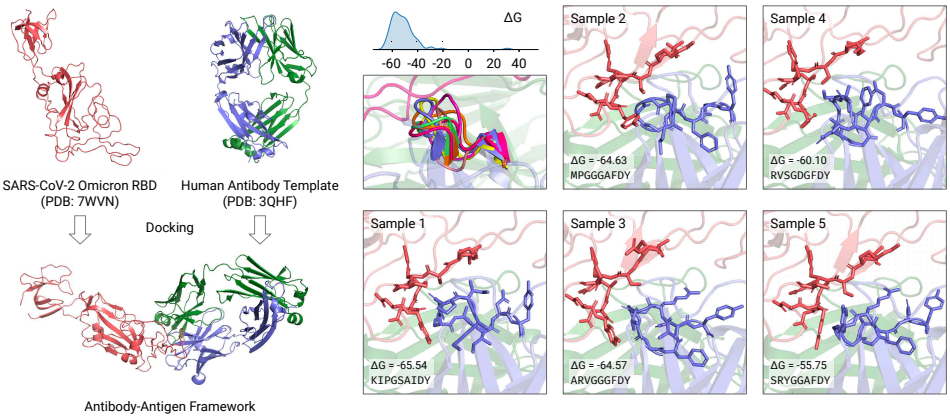

Figure 6: A human antibody framework docked to SARS-CoV-2 Omicron RBD using HDOCK. CDR-H3s are designed based on the docking structure.

4.4 Design Without Bound Antibody Frameworks

In the last experiment, we consider designing antibodies without a known binding pose against the antigen, a more general and challenging setting. We show that this challenging task could be achieved with docking software. Specifically, we create an *antibody template* from an existing antibody structure by removing its CDR-H3. This is because CDR-H3 is the most variable one and accounts for most of the specificity, while other CDRs are much more conserved [Xu and Davis, 2000]. Next, we use HDOCK [Yan et al., 2017] to dock the antibody template to the target antigen to produce the antibody-antigen complex. In this way, the problem reduces to the original problem so we can adapt our model to design the CDR-H3 sequence and structure and re-design other CDRs. We demonstrate using this method to design antibodies for the SARS-CoV-2 Omicron RDB structure (PDB: `7wvn`, residue A322-A590, the structure is not bound to any antibodies). The antibody template is derived from a human antibody against influenza (PDB: `3qhf`). Figure 6 shows the docking structure, five designed CDR-H3s, and the binding energy distribution. It is hard to confidently conclude that the generated antibodies are effective without a reference antibody. However, according to the binding energy distribution, we can still say the generated antibodies are at least reasonable.

5 Conclusions and Limitations

In this work, we propose a diffusion-based generative model for antibody design. Our model is capable of a wide range of antibody design tasks and can achieve competitive performance. One main limitation of this work is that it relies on an antibody framework bound to the target antigen. Therefore, we leave it for future work to design an effective model for generating antibodies without bound structures. Another limitation is that it remains unclear whether the generated antibodies can be produced in the wet lab and actually binds to the target. More efforts are needed to design a biologically effective antibody.

Acknowledgments and Disclosure of Funding

Supported by National Key R&D Program of China No. 2021YFF1201600.

References

Jared Adolf-Bryfogle, Oleks Kalyuzhniy, Michael Kubitz, Brian D Weitzner, Xiaozhen Hu, Yumiko Adachi, William R Schief, and Roland L Dunbrack Jr. Rosettaantibodydesign (rabd): A general framework for computational antibody design. *PLoS computational biology*, 14(4):e1006112, 2018.

Rahmad Akbar, Habib Bashour, Puneet Rawat, Philippe A Robert, Eva Smorodina, Tudor-Stefan Cotet, Karine Flem-Karlsen, Robert Frank, Brij Bhushan Mehta, Mai Ha Vu, et al. Progress and

challenges for the machine learning-based design of fit-for-purpose monoclonal antibodies. In *Mabs*, volume 14, page 2008790. Taylor & Francis, 2022a.

Rahmad Akbar, Philippe A Robert, Cédric R Weber, Michael Widrich, Robert Frank, Milena Pavlović, Lonneke Scheffer, Maria Chernigovskaya, Igor Snapkov, Andrei Slabodkin, et al. In silico proof of principle of machine learning-based antibody design at unconstrained scale. In *Mabs*, volume 14, page 2031482. Taylor & Francis, 2022b.

Rebecca F Alford, Andrew Leaver-Fay, Jeliazko R Jeliazkov, Matthew J O'Meara, Frank P DiMaio, Hahnbeom Park, Maxim V Shapovalov, P Douglas Renfrew, Vikram K Mulligan, Kalli Kappel, et al. The rosetta all-atom energy function for macromolecular modeling and design. *Journal of chemical theory and computation*, 13(6):3031–3048, 2017.

Ethan C Alley, Grigory Khimulya, Surojit Biswas, Mohammed AlQuraishi, and George M Church. Unified rational protein engineering with sequence-based deep representation learning. *Nature methods*, 16(12):1315–1322, 2019.

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.

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

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.

Kyle A Barlow, Shane O Conchuir, Samuel Thompson, Pooja Suresh, James E Lucas, Markus Heinonen, and Tanja Kortemme. Flex ddg: Rosetta ensemble-based estimation of changes in protein–protein binding affinity upon mutation. *The Journal of Physical Chemistry B*, 122(21): 5389–5399, 2018.

Sidhartha Chaudhury, Sergey Lyskov, and Jeffrey J Gray. Pyrosetta: a script-based interface for implementing molecular modeling algorithms using rosetta. *Bioinformatics*, 26(5):689–691, 2010.

Valentin De Bortoli, Emile Mathieu, Michael Hutchinson, James Thornton, Yee Whye Teh, and Arnaud Doucet. Riemannian score-based generative modeling. *arXiv preprint arXiv:2202.02763*, 2022.

James Dunbar, Konrad Krawczyk, Jinwoo Leem, Terry Baker, Angelika Fuchs, Guy Georges, Jiye Shi, and Charlotte M Deane. Sabdab: the structural antibody database. *Nucleic acids research*, 42 (D1):D1140–D1146, 2014.

Peter Eastman, Jason Swails, John D Chodera, Robert T McGibbon, Yutong Zhao, Kyle A Beauchamp, Lee-Ping Wang, Andrew C Simmonett, Matthew P Harrigan, Chaya D Stern, et al. Openmm 7: Rapid development of high performance algorithms for molecular dynamics. *PLoS computational biology*, 13(7):e1005659, 2017.

RA Engh and R Huber. Structure quality and target parameters. 2012.

Jean Gallier and Dianna Xu. Computing exponentials of skew-symmetric matrices and logarithms of orthogonal matrices. *International Journal of Robotics and Automation*, 18(1):10–20, 2003.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

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

Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. *Advances in Neural Information Processing Systems*, 34, 2021.

Emiel Hoogeboom, Victor Garcia Satorras, Clément Vignac, and Max Welling. Equivariant diffusion for molecule generation in 3d. *arXiv preprint arXiv:2203.17003*, 2022.

Chloe Hsu, Robert Verkuil, Jason Liu, Zeming Lin, Brian Hie, Tom Sercu, Adam Lerer, and Alexander Rives. Learning inverse folding from millions of predicted structures. *bioRxiv*, 2022.

John Ingraham, Vikas Garg, Regina Barzilay, and Tommi Jaakkola. Generative models for graph-based protein design. *Advances in Neural Information Processing Systems*, 32, 2019.

Charles A Janeway, Paul Travers, Mark Walport, and Donald J Capra. *Immunobiology*. Taylor & Francis Group UK: Garland Science, 2001.

Wengong Jin, Jeremy Wohlwend, Regina Barzilay, and Tommi S. Jaakkola. Iterative refinement graph neural network for antibody sequence-structure co-design. In *International Conference on Learning Representations*, 2022.

Bowen Jing, Gabriele Corso, Jeffrey Chang, Regina Barzilay, and Tommi Jaakkola. Torsional diffusion for molecular conformer generation. *arXiv preprint arXiv:2206.01729*, 2022.

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.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Gideon D Lapidoth, Dror Baran, Gabriele M Pszolla, Christoffer Norn, Assaf Alon, Michael D Tyka, and Sarel J Fleishman. Abdesign: A n algorithm for combinatorial backbone design guided by natural conformations and sequences. *Proteins: Structure, Function, and Bioinformatics*, 83(8): 1385–1406, 2015.

Adam Leach, Sebastian M Schmon, Matteo T Degiacomi, and Chris G Willcocks. Denoising diffusion probabilistic models on so (3) for rotational alignment. In *ICLR 2022 Workshop on Geometrical and Topological Representation Learning*, 2022.

Anders Liljas, Lars Liljas, Goran Lindblom, Poul Nissen, Morten Kjeldgaard, and Miriam-rose Ash. *Textbook of structural biology*, volume 8. World Scientific, 2016.

Kresten Lindorff-Larsen, Stefano Piana, Kim Palmo, Paul Maragakis, John L Klepeis, Ron O Dror, and David E Shaw. Improved side-chain torsion potentials for the amber ff99sb protein force field. *Proteins: Structure, Function, and Bioinformatics*, 78(8):1950–1958, 2010.

S Matthies, J Muller, and GW Vinel. On the normal distribution in the orientation space. *Textures and Microstructures*, 10, 1970.

Dmitry I Nikolayev and Tatjana I Savyolov. Normal distribution on the rotation group so (3). *Textures and Microstructures*, 29, 1970.

RJ Pantazes and Costas D Maranas. Optcdr: a general computational method for the design of antibody complementarity determining regions for targeted epitope binding. *Protein Engineering, Design & Selection*, 23(11):849–858, 2010.

Leonard G Presta. Antibody engineering. *Current Opinion in Structural Biology*, 2(4):593–596, 1992.

Jeffrey A Ruffolo, Jeffrey J Gray, and Jeremias Sulam. Deciphering antibody affinity maturation with language models and weakly supervised learning. *arXiv*, 2021.

Jeffrey A Ruffolo, Lee-Shin Chu, Sai Pooja Mahajan, and Jeffrey J Gray. Fast, accurate antibody structure prediction from deep learning on massive set of natural antibodies. *bioRxiv*, 2022a.

Jeffrey A Ruffolo, Jeremias Sulam, and Jeffrey J Gray. Antibody structure prediction using interpretable deep learning. *Patterns*, 3(2):100406, 2022b.

Koichiro Saka, Taro Kakuzaki, Shoichi Metsugi, Daiki Kashiwagi, Kenji Yoshida, Manabu Wada, Hiroyuki Tsunoda, and Reiji Teramoto. Antibody design using lstm based deep generative model from phage display library for affinity maturation. *Scientific reports*, 11(1):1–13, 2021.

Maxim V Shapovalov and Roland L Dunbrack Jr. A smoothed backbone-dependent rotamer library for proteins derived from adaptive kernel density estimates and regressions. *Structure*, 19(6): 844–858, 2011.

Chence Shi, Shitong Luo, Minkai Xu, and Jian Tang. Learning gradient fields for molecular conformation generation. In *International Conference on Machine Learning*, pages 9558–9568. PMLR, 2021.

Jung-Eun Shin, Adam J Riesselman, Aaron W Kollasch, Conor McMahon, Elana Simon, Chris Sander, Aashish Manglik, Andrew C Kruse, and Debora S Marks. Protein design and variant prediction using autoregressive generative models. *Nature communications*, 12(1):1–11, 2021.

Ken Shoemake. Uniform random rotations. In *Graphics Gems III (IBM Version)*, pages 124–132. Elsevier, 1992.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015.

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

Alexey Strokach, David Becerra, Carles Corbi-Verge, Albert Perez-Riba, and Philip M Kim. Fast and flexible protein design using deep graph neural networks. *Cell systems*, 11(4):402–411, 2020.

Doug Tischer, Sidney Lisanza, Jue Wang, Runze Dong, Ivan Anishchenko, Lukas F Milles, Sergey Ovchinnikov, and David Baker. Design of proteins presenting discontinuous functional sites using deep learning. *Biorxiv*, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Shira Warszawski, Aliza Borenstein Katz, Rosalie Lipsh, Lev Khmelnitsky, Gili Ben Nissan, Gabriel Javitt, Orly Dym, Tamar Unger, Orli Knop, Shira Albeck, et al. Optimizing antibody affinity and stability by the automated design of the variable light-heavy chain interfaces. *PLoS computational biology*, 15(8):e1007207, 2019.

John L Xu and Mark M Davis. Diversity in the cdr3 region of vh is sufficient for most antibody specificities. *Immunity*, 13(1):37–45, 2000.

Yumeng Yan, Di Zhang, Pei Zhou, Botong Li, and Sheng-You Huang. Hdock: a web server for protein–protein and protein–dna/rna docking based on a hybrid strategy. *Nucleic acids research*, 45(W1):W365–W373, 2017.

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.

Ling Yang, Zhilong Zhang, and Shenda Hong. Diffusion models: A comprehensive survey of methods and applications. *arXiv preprint arXiv:2209.00796*, 2022.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] Our method might be used to design malicious proteins.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Appendix E
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Single A100 GPU.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

A Diffusion Processes

A.1 Posteriors

Posterior of Amino Acid Types The generative diffusion kernel for amino acid types $p(\mathrm{s}_j^{t-1}|\mathcal{R}^t, \mathcal{C})$ (Eq.3) should align to the posterior $q(\mathrm{s}_j^{t-1}|\mathrm{s}_j^t, \mathrm{s}_j^0)$. It can be derived from Eq.1 and Eq.2 [Hoogeboom et al., 2021]:

$$q(\mathrm{s}_j^{t-1}|\mathrm{s}_j^t, \mathrm{s}_j^0) = \mathrm{Multinomial}\left(\left[\alpha_{\text{type}}^t \cdot \texttt{onehot}(\mathrm{s}_j^t) + (1-\alpha_{\text{type}}^t)\cdot\frac{1}{20}\cdot\mathbf{1}\right]\odot \left[\bar{\alpha}_{\text{type}}^{t-1} \cdot \texttt{onehot}(\mathrm{s}_j^0) + (1-\bar{\alpha}_{\text{type}}^{t-1})\cdot\frac{1}{20}\cdot\mathbf{1}\right]\right). \quad (17)$$

The vector inside $\mathrm{Multinomial}(\cdot)$ might not sum to one. In this case, the probability of a class is the ratio of the value in the sum of the vector.

Posterior of C_α Coordinates The generative diffusion kernel $p\left(\mathbf{x}_j^{t-1}\middle|\mathcal{R}^t, \mathcal{C}\right)$ (Eq.7) should align to the posterior obtained from Eq.5 and Eq.6 [Ho et al., 2020]:

$$q(\mathbf{x}_j^{t-1} \mid \mathbf{x}_j^t, \mathbf{x}_j^0) = \mathcal{N}\left(\mathbf{x}_j^{t-1}\middle|\boldsymbol{\mu}_q\left(\mathbf{x}_j^t, \mathbf{x}_j^0\right), \frac{(1-\bar{\alpha}_{\text{pos}}^{t-1})\beta_{\text{pos}}^t}{1-\bar{\alpha}_{\text{pos}}^t}\boldsymbol{I}\right), \quad (18)$$

$$\text{where}\quad \boldsymbol{\mu}_q(\cdots) = \frac{\sqrt{\bar{\alpha}_{\text{pos}}^{t-1}}\beta_{\text{pos}}^t}{1-\bar{\alpha}_{\text{pos}}^{t-1}}\mathbf{x}_j^0 + \frac{\sqrt{\alpha_{\text{pos}}^t}(1-\bar{\alpha}_{\text{pos}}^{t-1})}{1-\bar{\alpha}_{\text{pos}}^t}\mathbf{x}_j^t. \quad (19)$$

A.2 Amino Acid C_α Position Normalization

As amino acid C_α positions could be arbitrary in the 3D space. We need to normalize them to use the standard normal distribution with zero mean and unit variance as the prior distribution. First, we need to derive the statistics of CDR positions. For each CDR in the SAbDab dataset, we shift the overall structure such that the center point of the two CDR anchors is located in origin. Then, we aggregate C_α positions in the shifted CDRs. Finally, we calculate the mean and standard deviation of them. Before training and inference, we shift the whole structure according to their CDR anchors and further shift and scale the structure according to the pre-calculated mean and standard deviation to obtain the normalized coordinates.

B Distributions on SO(3)

B.1 Preliminary: Axis-Angle Representation of Rotations

Conventionally, a rotation is usually represented by 3 Euler angles (α, β, γ), which can be interpreted as the composition of counter-clockwise rotations by α, β, γ about x, y, z axes. However, the Euler representation is unsuitable for defining useful operations and distributions w.r.t. rotations considered in this work. Alternatively, we introduce another rotation representation called *axis-angle representations*. This representation parameterized a rotation with an rotational axis $\boldsymbol{u}$ ($\|\boldsymbol{u}\|_2 = 1$) and an angle θ ($\theta \in \mathbb{R}$).

B.2 Logarithm of Rotation Matrices and Exponential of Skew-Symmetric Matrices

Logarithm of Rotation Matrices Derived from the definition of matrix logarithm, the logarithm of a rotation matrix $\boldsymbol{R}$ is a **skew-symmetric matrix** [Gallier and Xu, 2003], which can be represented as:

$$\boldsymbol{S} := \log \boldsymbol{R} = \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix}. \quad (20)$$

It can be proven that $\boldsymbol{v} = [v_x, v_y, v_z]$ is the rotational axis of $\boldsymbol{R}$, and $\|\boldsymbol{v}\|_2$ is the rotational angle. For brevity, we can use the vector notation $\boldsymbol{v}$ to represent a rotation in the logarithm space. The space is also known as $so(3)$ (different from the rotation group $\mathrm{SO}(3)$, the symbol is in lowercase).

To efficiently compute the logarithm of a rotation matrix without computing matrix logarithm or solving rotational axis-angle, we can use the following formula [Gallier and Xu, 2003]:

$$\log \boldsymbol{R} = \frac{\theta}{2\sin\theta}(\boldsymbol{R} - \boldsymbol{R}^\intercal), \tag{21}$$

where θ can be obtained from $\theta = \cos^{-1}\left(\frac{\mathrm{Tr}\,\boldsymbol{R}-1}{2}\right)$ by the fact that $\mathrm{Tr}(\boldsymbol{R}) = 1 + 2\cos\theta$. Specially, when $\theta = 0$ (or $\boldsymbol{R} = \boldsymbol{I}$), $\log \boldsymbol{R} = [\boldsymbol{0}, \boldsymbol{0}, \boldsymbol{0}]$.

Exponential of Skew-Symmetric Matrices The inversion of the rotation matrix logarithm is the exponential of skew-symmetric matrices. Derived from the definition of matrix exponential, the conversion formula is [Gallier and Xu, 2003]:

$$\exp \boldsymbol{S} = \boldsymbol{I} + \frac{\sin\|\boldsymbol{v}\|_2}{\|\boldsymbol{v}\|_2}\boldsymbol{S} + \frac{1-\cos\|\boldsymbol{v}\|_2}{\|\boldsymbol{v}\|_2^2}\boldsymbol{S}^2, \tag{22}$$

where $\boldsymbol{S}$ is a skew-symmetric matrix parameterized by three values $\boldsymbol{v} = [v_x, v_y, v_z]$, identical to the definition in Eq.20.

Remarks The logarithm and exponential defined above provide an easy way to create and manipulate rotations in the axis-angle parameterization space. For example, when we would like to create a rotation matrix with an axis and an angle, we can first create a vector $\boldsymbol{v}$ whose direction is the same as the given axis and whose length equals the angle. Then, we rewrite the vector $\boldsymbol{v}$ into a skew-symmetric matrix $\boldsymbol{S}$, and finally convert it to a rotation matrix by Eq.22. We can also manipulate a rotation matrix, for example, changing its rotational angle, by mapping it to the logarithm space, modifying the skew-symmetric matrix, and finally converting it back to a rotation matrix using the exponential formula.

B.3 $\mathrm{ScaleRot}$: Rotation Scaling Function

When we parameterize a rotation matrix with an axis and an angle, it is natural to define the rotation scaling function $\mathrm{ScaleRot}$ as scaling the rotational angle. Formally, the definition is:

$$\mathrm{ScaleRot}(k, \boldsymbol{R}) := \exp\left(k \log \boldsymbol{R}\right), \tag{23}$$

where k is the scaling factor and $\boldsymbol{R}$ is a rotation matrix. Specially, $\mathrm{ScaleRot}(0, \boldsymbol{R}) = \boldsymbol{I}$ for all rotation matrix $\boldsymbol{R}$. Intuitively, scaling a rotation matrix by 0 cancels its effect, leading to identity transformation.

B.4 $\mathcal{IG}_{\mathrm{SO}(3)}$: Isotropic Gaussian Distribution on SO(3)

The isotropic Gaussian distribution on $\mathrm{SO}(3)$, denoted as $\mathcal{IG}_{\mathrm{SO}(3)}$, is defined on the axis-angle space of rotation: $\mathbb{S}^2 \times [0, \pi]$, where $\mathbb{S}^2 = \{\|\boldsymbol{x}\|_2 = 1 | \boldsymbol{x} \in \mathbb{R}^3\}$ is the unit sphere in $\mathbb{R}^3$. $\mathcal{IG}_{\mathrm{SO}(3)}$ is parameterized by a mean rotation $\boldsymbol{M}$ and a scalar variance σ^2. Let $\mathbf{u} \in \mathbb{S}^2$ and θ denote the rotational axis and angle random variables respectively. We first consider $\mathcal{IG}_{\mathrm{SO}(3)}$ with the identity matrix as its mean: $\mathcal{IG}_{\mathrm{SO}(3)}(\mathbf{u}, \theta | \boldsymbol{I}, \sigma^2)$. Its p.d.f. is defined by the product of the uniform distribution on $\mathbb{S}^2$ and a special angular distribution [Matthies et al., 1970, Nikolayev and Savyolov, 1970, Leach et al., 2022]:

$$p_{\mathcal{IG}_{\mathrm{SO}(3)}}(\mathbf{u}, \theta | \boldsymbol{I}, \sigma^2) = p_{\text{uniform}(\mathbb{S}^2)}(\mathbf{u}) p_{\text{angular}}(\theta | \sigma^2), \tag{24}$$

$$\text{where} \quad p_{\text{uniform}(\mathbb{S}^2)}(\mathbf{u}) = \frac{1}{4\pi}\delta\left(\|\mathbf{u}\|_2 - 1\right), \qquad (\mathbf{u} \in \mathbb{S}^2) \tag{25}$$

$$\text{and} \quad p_{\text{angular}}(\theta | \sigma^2) = \frac{1-\cos\theta}{\pi}\sum_{l=0}^{\infty}(2l+1)e^{-l(l+1)\sigma^2}\frac{\sin\left(\left(l+\frac{1}{2}\right)\theta\right)}{\sin(\frac{\theta}{2})}. \qquad (\theta \in [0, \pi]) \tag{26}$$

When the mean is other than $\boldsymbol{I}$, to sample from the distribution, we can first sample an rotation $\boldsymbol{E}$ from $\mathcal{IG}_{\mathrm{SO}(3)}(\mathbf{u}, \theta | \boldsymbol{I}, \sigma^2)$. Then, we left-multiply $\boldsymbol{R}$ to $\boldsymbol{E}$ to obtain the desired random value $\boldsymbol{RE}$.

Sampling The algorithm for drawing samples from $\mathcal{IG}_{\mathrm{SO}(3)}(\boldsymbol{I}, \sigma^2)$ (here, the mean rotation is identity) can be broken down into two steps.

The first step is to draw a unit vector $\boldsymbol{u}$ from the uniform distribution on $\mathbb{S}^2$, $p_{\text{uniform}(\mathbb{S}^2)}(\mathbf{u})$. This can be done efficiently by sampling from the 3D standard Gaussian distribution and then normalizing the sampled vector to unit length.

The second part is drawing samples from $p_{\text{angular}}(\theta|\sigma^2)$, which could be more tricky. We empirically use two different proximate sampling strategies depending on the variance σ^2. When σ is larger than 0.1, the series (Eq.26) converges fast. In such cases, we use histograms to approximate the distribution. In specific, we evenly partition $[0, \pi]$ into 8192 bins, and use the probability density $p_{\text{angular}}(\theta|\sigma^2)$ at the center of each bin as the bin weight. We randomly select a bin according to the weights to draw samples from the discretized distribution. Then, we sample from the uniform distribution spanning from the lower bound to the upper bound of the bin. The discretization process is time-consuming. However, since the variances in the diffusion processes are predetermined, we pre-compute and cache the bins and weights to draw samples efficiently. When σ is smaller than 0.1, we approximate the distribution using the truncated Gaussian distribution whose mean is 2σ and the standard deviation is σ. Empirically, we find that the above proximate sampling algorithm is sufficient for training and sampling from our diffusion model.

To sample from $\mathcal{IG}_{\mathrm{SO}(3)}$ with an arbitrary mean rotation $\boldsymbol{R}$, we first draw a rotation from $\mathcal{IG}_{\mathrm{SO}(3)}(\boldsymbol{I}, \sigma^2)$, denoted as $\boldsymbol{E}$. Finally, we left-multiply $\boldsymbol{R}$ to $\boldsymbol{E}$ to get the desired sample.

B.5 Uniform Distribution on SO(3)

The uniform distribution on $\mathrm{SO}(3)$ is equivalent to the uniform distribution of normalized quaternions on $\mathbb{S}^3$ [Shoemake, 1992]. To sample a random rotation uniformly, we first sample a random vector from the 4D standard normal distribution. Next, we normalize the vector and treat it as a quaternion. Finally, we convert the quaternion to a rotation matrix which can be regarded as a sample from the uniform distribution on $\mathrm{SO}(3)$.

C Neural Network Parameterization

C.1 Computing Residue Orientations

The orientation of a residue is determined by the coordinate of its three backbone atoms: C_α, C, and N. Let $\boldsymbol{x}_i^\alpha$, $\boldsymbol{x}_i^{\mathrm{C}}$, and $\boldsymbol{x}_i^{\mathrm{N}}$ denote the 3D coordinates of the three backbone atoms of the i-th residue respectively. The orientation of the residue, denoted by $\boldsymbol{O}_i$, can be constructed using the following Gram-Schmidt-based algorithm:

$$\boldsymbol{v}_1 \leftarrow \boldsymbol{x}_i^{\mathrm{C}} - \boldsymbol{x}_i^\alpha, \tag{27}$$

$$\boldsymbol{e}_1 \leftarrow \frac{\boldsymbol{v}_1}{\|\boldsymbol{v}_1\|}, \tag{28}$$

$$\boldsymbol{v}_2 \leftarrow \boldsymbol{x}_i^{\mathrm{N}} - \boldsymbol{x}_i^\alpha, \tag{29}$$

$$\boldsymbol{u}_2 \leftarrow \boldsymbol{v}_2 - \langle \boldsymbol{e}_1, \boldsymbol{v}_2 \rangle \boldsymbol{e}_1, \tag{30}$$

$$\boldsymbol{e}_2 \leftarrow \frac{\boldsymbol{u}_2}{\|\boldsymbol{u}_2\|}, \tag{31}$$

$$\boldsymbol{e}_3 \leftarrow \boldsymbol{e}_1 \times \boldsymbol{e}_2, \tag{32}$$

$$\boldsymbol{O}_i \leftarrow [\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3]\,. \tag{33}$$

C.2 Architectures

Amino Acid Embedding Layer The embedding layer for each amino acid takes into account the following information:

- **Amino acid type**: Each of the 20 amino acid types is represented by an embedding vector denoted by $\boldsymbol{e}_i^{\text{type}}$.
- **Heavy atom local coordinates**: The coordinate of each heavy atom in an amino acid is projected to the local coordinate frame using the rule $\boldsymbol{x}_i^{\text{local}} = \boldsymbol{O}_i^\intercal(\boldsymbol{x}_i^{\text{atom}} - \boldsymbol{x}_i^\alpha)$. All

of the local coordinates are concatenated into a single vector denoted by $\boldsymbol{e}_i^{\text{coord}}$. If some heavy atoms are missing, their local coordinates are filled with zeros. Note that the local coordinates are invariant to global rotation and translation thanks to the projection rule.

- **Backbone dihedral angles**: The backbone dihedrals of amino acid, including ϕ, ψ, and ω [Liljas et al., 2016, Ingraham et al., 2019], are transformed using a series of sine and cosine functions with different frequencies, which are then concatenated into a single vector $\boldsymbol{e}_i^{\text{dihed}}$.
- **CDR flags and anchor flags**: Amino acids on the CDR or by the two ends of the CDR (anchors) are differentiated from other amino acids by special 0-1 flags denoted as $\boldsymbol{e}_i^{\text{flag}}$.

All of the vectors above are concatenated and fed to an MLP to produce the final embedding vector for each residue.

Pairwise Embedding Layer Pairwise embeddings include information about the relationship between two residues. The pairwise embedding for residue i and j involves the following information:

- **Amino acid types of both amino acids**: There are $20 \times 20 = 400$ combinations of two amino acid types. We represent each of them using an embedding vector denoted by $\boldsymbol{z}_{ij}^{\text{type}}$.
- **Sequential relative position**: If two residues are on the same chain and their distance on the sequence is less than or equal to 32 ($d_{ij}^{\text{seq}} \in \{-32 \dots 32\}$), the distance is represented by an embedding vector $\boldsymbol{z}_{ij}^{\text{seq}}$. Otherwise, the distance embedding is filled with zeros.
- **Pairwise distances**: The distances between all pairs of atoms are flattened into a vector and transformed by $e^{-c\boldsymbol{d}_{ij}}$ (c is a learnable coefficient) into the spatial distance embedding $\boldsymbol{z}_{ij}^{\text{dist}}$. Missing pairs are filled with zeros.
- **Pairwise backbone dihedrals**: The backbone dihedrals between any two amino acids i and j are defined as $\phi_{ij} = \text{Dihedral}(\boldsymbol{x}_i^{\text{C}}, \boldsymbol{x}_j^{\text{N}}, \boldsymbol{x}_j^{\alpha}, \boldsymbol{x}_j^{\text{C}})$ and $\psi_{ij} = \text{Dihedral}(\boldsymbol{x}_i^{\text{N}}, \boldsymbol{x}_i^{\alpha}, \boldsymbol{x}_i^{\text{C}}, \boldsymbol{x}_j^{\text{N}})$. These two dihedrals are transformed by a series of sine and cosine functions into pairwise dihedral embeddings $\boldsymbol{z}_{ij}^{\text{dihed}}$.

We concatenate the above vectors and feed them into an MLP to get the final pairwise embeddings for each pair of amino acids $\boldsymbol{z}_{ij}$.

Encoder The encoder for encoding the current diffusion state consists of a stack of orientation-aware invariant 3D attention layers. Its aim is to capture relationships between amino acids and provide high-level representations for each residue to denoise.

Let $\boldsymbol{h}_i^{\ell}$ denote the hidden representation output from the last layer (when $\ell = 0$, the representation is the initial residue embedding). The formulas for computing the logit of attention weight between residue i (query) and j (key) is:

$$a_{ij} = \left\langle \boldsymbol{q}\left(\boldsymbol{h}_i^{\ell}\right), \boldsymbol{k}\left(\boldsymbol{h}_j^{\ell}\right) \right\rangle + f\left(\boldsymbol{z}_{ij}\right) + g\left(\left\{\boldsymbol{O}_i{}^{\intercal}(\boldsymbol{x}_j^{\text{atom}} - \boldsymbol{x}_i^{\alpha})\right\}_{\text{atom}}\right), \tag{34}$$

where $\boldsymbol{q}(\cdot)$, $\boldsymbol{k}(\cdot)$, $f(\cdot)$, and $g(\cdot)$ are MLP subnetworks. The attention weights can be obtained by taking softmax: $w_{ij} = \text{softmax}_{j=1}^{N}(a_{ij})$. Note that, for simplicity, we do not consider attention heads in the formula, but in practice we use multiple attention heads and different heads can be combined easily via concatenation.

The formula for computing the value passed from residue j to i is:

$$\boldsymbol{v}_{ij} = \boldsymbol{v}\left(\boldsymbol{h}_j^{\ell}, \boldsymbol{z}_{ij}, \left\{\boldsymbol{O}_i{}^{\intercal}(\boldsymbol{x}_j^{\text{atom}} - \boldsymbol{x}_i^{\alpha})\right\}_{\text{atom}}\right), \tag{35}$$

where $\boldsymbol{v}(\cdot)$ is a network consisting of MLPs. Finally, the values along with attention weights are used to update the amino acid representations with residual connection and layer normalization, same as the standard transformer [Vaswani et al., 2017].

C.3 Notes on the Notations of the Denoising Networks F, G, and H

The notations F, G, and H do *not only* denote the MLPs following the encoder that outputs denoising results. It refers to the embedding layers, the encoder, and the specific output MLP (for example, F includes the MLP for denoising amino acid types). Therefore, the input to F, G, and H is the diffusion state (sequence and structure) rather than hidden representations. Treating the three sections as a whole allows us to neatly express the equivariance property of the model.

C.4 Proof of Equivariance

Lemma 1. *The Euclidean distance function between two points is invariant to rotations and translations, i.e.* $d(\boldsymbol{R}\boldsymbol{x}_1 + \boldsymbol{r}, \boldsymbol{R}\boldsymbol{x}_2 + \boldsymbol{r}) = d(\boldsymbol{x}_1, \boldsymbol{x}_2)$, $\forall \boldsymbol{R} \in \mathrm{SO}(3), \boldsymbol{r} \in \mathbb{R}^3$.

Proof.

$$\begin{aligned} d(\boldsymbol{R}\boldsymbol{x}_1 + \boldsymbol{r}, \boldsymbol{R}\boldsymbol{x}_2 + \boldsymbol{r}) &= \|(\boldsymbol{R}\boldsymbol{x}_1 + \boldsymbol{r}) - (\boldsymbol{R}\boldsymbol{x}_2 + \boldsymbol{r})\|_2 \\ &= \|\boldsymbol{R}(\boldsymbol{x}_1 - \boldsymbol{x}_2)\|_2 \\ &= (\boldsymbol{x}_1 - \boldsymbol{x}_2)^\intercal \boldsymbol{R}^\intercal \boldsymbol{R}(\boldsymbol{x}_1 - \boldsymbol{x}_2) \\ &= \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|_2 \\ &= d(\boldsymbol{x}_1, \boldsymbol{x}_2). \end{aligned}$$

□

Lemma 2. *The dihedral function for four points is invariant to rotations and translations, i.e.* $\mathrm{Dihedral}(\boldsymbol{R}\boldsymbol{x}_1 + \boldsymbol{r}, \boldsymbol{R}\boldsymbol{x}_2 + \boldsymbol{r}, \boldsymbol{R}\boldsymbol{x}_3 + \boldsymbol{r}, \boldsymbol{R}\boldsymbol{x}_4 + \boldsymbol{r}) = \mathrm{Dihedral}(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4)$, $\forall \boldsymbol{R} \in \mathrm{SO}(3), \boldsymbol{r} \in \mathbb{R}^3$. *Here,* $\mathrm{Dihedral}(\cdots)$ *is defined as:*

$$\mathrm{Dihedral}(\boldsymbol{x}_1 \dots \boldsymbol{x}_4) = \mathrm{atan2}(\boldsymbol{v}_2 \cdot ((\boldsymbol{v}_1 \times \boldsymbol{v}_2) \times (\boldsymbol{v}_2 \times \boldsymbol{v}_3)), \|\boldsymbol{v}_2\|(\boldsymbol{v}_1 \times \boldsymbol{v}_2) \cdot (\boldsymbol{v}_2 \times \boldsymbol{v}_3)), \quad (36)$$

where $\boldsymbol{v}_i = \boldsymbol{x}_{i+1} - \boldsymbol{x}_i$ *(*$i = 1, 2, 3$*).*

Proof. First, we note that:

$$(\boldsymbol{R}\boldsymbol{x}_{i+1} + \boldsymbol{r}) - (\boldsymbol{R}\boldsymbol{x}_i + \boldsymbol{r}) = \boldsymbol{R}(\boldsymbol{x}_{i+1} - \boldsymbol{x}_i) = \boldsymbol{R}\boldsymbol{v}_i.$$

By the equivariance of cross product ($\boldsymbol{R}\boldsymbol{a} \times \boldsymbol{R}\boldsymbol{b} = \boldsymbol{R}(\boldsymbol{a} \times \boldsymbol{b})$) and the invariance of inner product ($\boldsymbol{R}\boldsymbol{a} \cdot \boldsymbol{R}\boldsymbol{b} = \boldsymbol{a} \cdot \boldsymbol{b}$), we have:

$$\begin{aligned} \mathrm{Dihedral}(\boldsymbol{R}\boldsymbol{x}_i + \boldsymbol{r} | i = 1 \dots 4) &= \mathrm{atan2}(\boldsymbol{R}\boldsymbol{v}_2 \cdot (\boldsymbol{R}(\boldsymbol{v}_1 \times \boldsymbol{v}_2) \times \boldsymbol{R}(\boldsymbol{v}_2 \times \boldsymbol{v}_3)), \\ &\qquad \|\boldsymbol{R}\boldsymbol{v}_2\|\boldsymbol{R}(\boldsymbol{v}_1 \times \boldsymbol{v}_2) \cdot \boldsymbol{R}(\boldsymbol{v}_2 \times \boldsymbol{v}_3)) \\ &= \mathrm{atan2}(\boldsymbol{R}\boldsymbol{v}_2 \cdot \boldsymbol{R}((\boldsymbol{v}_1 \times \boldsymbol{v}_2) \times (\boldsymbol{v}_2 \times \boldsymbol{v}_3)), \\ &\qquad \|\boldsymbol{v}_2\|(\boldsymbol{v}_1 \times \boldsymbol{v}_2) \cdot (\boldsymbol{v}_2 \times \boldsymbol{v}_3)) \\ &= \mathrm{atan2}(\boldsymbol{v}_2 \cdot ((\boldsymbol{v}_1 \times \boldsymbol{v}_2) \times (\boldsymbol{v}_2 \times \boldsymbol{v}_3)), \\ &\qquad \|\boldsymbol{v}_2\|(\boldsymbol{v}_1 \times \boldsymbol{v}_2) \cdot (\boldsymbol{v}_2 \times \boldsymbol{v}_3)) \\ &= \mathrm{Dihedral}(\boldsymbol{x}_i | i = 1 \dots 4) \end{aligned}$$

□

Lemma 3. *The per-amino-acid orientation* $\boldsymbol{O}_i$ *is equivariant to rotations and translations, i.e.,* $\boldsymbol{O}(\boldsymbol{R}\boldsymbol{x}_i^\alpha + \boldsymbol{r}, \boldsymbol{R}\boldsymbol{x}_i^\mathrm{C} + \boldsymbol{r}, \boldsymbol{R}\boldsymbol{x}_i^\mathrm{N} + \boldsymbol{r}) = \boldsymbol{R}\boldsymbol{O}(\boldsymbol{x}_i^\alpha, \boldsymbol{x}_i^\mathrm{C}, \boldsymbol{x}_i^\mathrm{N})$

Proof. First, we show that the first two basis vectors $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ are equivariant:

$$\begin{aligned} \boldsymbol{e}_1(\boldsymbol{R}\boldsymbol{x}_i^\alpha + \boldsymbol{r}, \boldsymbol{R}\boldsymbol{x}_i^\mathrm{C} + \boldsymbol{r}) &= \frac{(\boldsymbol{R}\boldsymbol{x}_i^\mathrm{C} + \boldsymbol{r}) - (\boldsymbol{R}\boldsymbol{x}_i^\alpha + \boldsymbol{r})}{\|(\boldsymbol{R}\boldsymbol{x}_i^\mathrm{C} + \boldsymbol{r}) - (\boldsymbol{R}\boldsymbol{x}_i^\alpha + \boldsymbol{r})\|} \\ &= \boldsymbol{R}\frac{\boldsymbol{x}_i^\mathrm{C} - \boldsymbol{x}_i^\alpha}{\|v\boldsymbol{x}_i^\mathrm{C} - \boldsymbol{x}_i^\alpha\|} \\ &= \boldsymbol{R}\boldsymbol{e}_1(\boldsymbol{x}_i^\alpha, \boldsymbol{x}_i^\mathrm{C}). \end{aligned}$$

Let $\boldsymbol{v}_2 = \boldsymbol{x}_i^\mathrm{N} - \boldsymbol{x}_i^\alpha$. We have $(\boldsymbol{R}\boldsymbol{x}_i^\mathrm{N} + \boldsymbol{r}) - (\boldsymbol{R}\boldsymbol{x}_i^\alpha + \boldsymbol{r}) = \boldsymbol{R}\boldsymbol{v}_2$. Then, we can prove the equivariance of $\boldsymbol{e}_2$:

$$\begin{aligned} \boldsymbol{e}_2(\boldsymbol{R}\boldsymbol{x}_i^\alpha + \boldsymbol{r}, \boldsymbol{R}\boldsymbol{x}_i^\mathrm{C} + \boldsymbol{r}, \boldsymbol{R}\boldsymbol{x}_i^\mathrm{N} + \boldsymbol{r}) &= \boldsymbol{R}\boldsymbol{v}_2 - \langle \boldsymbol{R}\boldsymbol{e}_1, \boldsymbol{R}\boldsymbol{v}_2 \rangle \boldsymbol{R}\boldsymbol{e}_1 \\ &= \boldsymbol{R}\boldsymbol{v}_2 - \langle \boldsymbol{e}_1, \boldsymbol{v}_2 \rangle \boldsymbol{R}\boldsymbol{e}_1 \\ &= \boldsymbol{R}(\boldsymbol{v}_2 - \langle \boldsymbol{e}_1, \boldsymbol{v}_2 \rangle \boldsymbol{e}_1) \\ &= \boldsymbol{R}\boldsymbol{e}_2(\boldsymbol{x}_i^\alpha, \boldsymbol{x}_i^\mathrm{C}, \boldsymbol{x}_i^\mathrm{N}) \end{aligned}$$

By the equivariance of cross product, it is straightforward to show that $\boldsymbol{e}_3$ is also equivariant. Finally, combining the equivariance of $\boldsymbol{e}_1$, $\boldsymbol{e}_1$, and $\boldsymbol{e}_3$, we prove the equivariance of the orientation matrix:

$$\begin{aligned} \boldsymbol{O}(\boldsymbol{R}\boldsymbol{x}_i^\alpha + \boldsymbol{r}, \boldsymbol{R}\boldsymbol{x}_i^\mathrm{C} + \boldsymbol{r}, \boldsymbol{R}\boldsymbol{x}_i^\mathrm{N} + \boldsymbol{r}) &= [\boldsymbol{R}\boldsymbol{e}_1, \boldsymbol{R}\boldsymbol{e}_2, \boldsymbol{R}\boldsymbol{e}_3] \\ &= \boldsymbol{R}\boldsymbol{O}(\boldsymbol{x}_i^\alpha, \boldsymbol{x}_i^\mathrm{C}, \boldsymbol{x}_i^\mathrm{N}). \end{aligned}$$

□

Lemma 4. *The per-amino-acid and pairwise embedding layers are invariant to rotations and translations of the input structure. i.e.*

$$\boldsymbol{e}(s_i, \{\boldsymbol{x}_i^{atom}\}_{atom}, \phi_i, \psi_i, \omega_i, \boldsymbol{e}_i^{flag}) = \boldsymbol{e}(s_i, \{\boldsymbol{R}\boldsymbol{x}_i^{atom} + \boldsymbol{r}\}_{atom}, \phi_i, \psi_i, \omega_i, \boldsymbol{e}_i^{flag}), \quad \textit{and}$$
$$\boldsymbol{z}(\{d(\boldsymbol{x}_i^{atom1}, \boldsymbol{x}_j^{atom2})\}_{atom1,\, atom2}, \cdots) = \boldsymbol{z}(\{d(\boldsymbol{R}\boldsymbol{x}_i^{atom1} + \boldsymbol{r}, \boldsymbol{R}\boldsymbol{x}_j^{atom2} + \boldsymbol{r})\}_{atom1,\, atom2}, \cdots).$$

Proof. Before embedding atom positions for an amino acid, the network first projects the positions using the orientation by the rule:

$$\boldsymbol{x}_i^{\text{local}} = \boldsymbol{O}_i^{\intercal}(\boldsymbol{x}_i^{\text{atom}} - \boldsymbol{x}_i^{\alpha})$$

The projection operation is invariant to rotations and translations, using Lemma 3:

$$\begin{aligned}\boldsymbol{x}_i^{\text{local}}(\boldsymbol{R}\boldsymbol{x}_i^{\text{atom}} + \boldsymbol{r}, \boldsymbol{R}\boldsymbol{x}_i^{\alpha} + \boldsymbol{r}) &= (\boldsymbol{R}\boldsymbol{O}_i)^{\intercal}((\boldsymbol{R}\boldsymbol{x}_i^{\text{atom}} + \boldsymbol{r}) - (\boldsymbol{R}\boldsymbol{x}_i^{\alpha} + \boldsymbol{r})) \\ &= \boldsymbol{O}_i^{\intercal}\boldsymbol{R}^{\intercal}\boldsymbol{R}(\boldsymbol{x}_i^{\text{atom}} - \boldsymbol{x}_i^{\alpha}) \\ &= \boldsymbol{x}_i^{\text{local}}(\boldsymbol{x}_i^{\text{atom}}, \boldsymbol{x}_i^{\alpha}).\end{aligned}$$

The formulas for computing dihedral angles $(\phi_i, \psi_i, \omega_i)$ are also invariant by Lemma 2 Other variables (amino acid types and CDR flags) are independent of the 3D structure and hence they are invariant.

So far, we have shown that all the components of embedding layers are invariant to rotations and translations of the overall 3D structure. Therefore, the embedding layer is invariant.

Pairwise embedding layers involve distances between residues, which are invariant by Lemma 2. Other variables are irrelevant to 3D structures. Hence, the pairwise embedding layer is invariant. □

Lemma 5. *The orientation-aware attention layer is invariant to rotations and translations if the input hidden representations $\boldsymbol{h}_i, \boldsymbol{z}_{ij}(i, j = 1 \dots N)$ come from invariant functions.*

Proof. First, we show that projecting atoms on the j-th amino acid to the orientation of the i-th amino acid is invariant to rotations and translations by Lemma 3:

$$(\boldsymbol{R}\boldsymbol{O}_i)^{\intercal}((\boldsymbol{R}\boldsymbol{x}_j^{\text{atom}} + \boldsymbol{r}) - (\boldsymbol{R}\boldsymbol{x}_i^{\alpha} + \boldsymbol{r})) = \boldsymbol{O}_i^{\intercal}\boldsymbol{R}^{\intercal}\boldsymbol{R}(\boldsymbol{x}_j^{\text{atom}} - \boldsymbol{x}_i^{\alpha}).$$

As other inputs to the attention layer $(\boldsymbol{h}_i, \boldsymbol{z}_{ij}(i, j = 1 \dots N))$ are invariant to rigid transforms on the structure, the networks for computing attention weights and values are invariant. Hence, the attention layer is invariant.

In the case where we stack multiple attention layers, each layer outputs invariant representations for its next layer. Therefore, the network consisting of multiple attention layers is invariant. □

Proposition 2. *For any proper rotation matrix $\boldsymbol{R} \in \mathrm{SO}(3)$ and any 3D vector $\boldsymbol{r} \in \mathbb{R}^3$ (rigid transformation $(\boldsymbol{R}, \boldsymbol{r}) \in \mathrm{SE}(3)$), F, G and H satisfy the following equivariance properties:*

$$F(\boldsymbol{R}\mathcal{R}^t + \boldsymbol{r}, \boldsymbol{R}\mathcal{C} + \boldsymbol{r}) = F(\mathcal{R}^t, \mathcal{C}), \tag{37}$$
$$G(\boldsymbol{R}\mathcal{R}^t + \boldsymbol{r}, \boldsymbol{R}\mathcal{C} + \boldsymbol{r}) = \boldsymbol{R}G(\mathcal{R}^t, \mathcal{C}), \tag{38}$$
$$H(\boldsymbol{R}\mathcal{R}^t + \boldsymbol{r}, \boldsymbol{R}\mathcal{C} + \boldsymbol{r}) = \boldsymbol{R}H(\mathcal{R}^t, \mathcal{C}), \tag{39}$$

where $\boldsymbol{R}\mathcal{R}^t + \boldsymbol{r} := \{\mathrm{s}_j^t, \mathbf{x}_j^t + \boldsymbol{r}, \boldsymbol{R}\mathbf{O}_j^t\}_{j=l+1}^{l+m}$ and $\boldsymbol{R}\mathcal{C} + \boldsymbol{r} := \{s_i, \boldsymbol{x}_i + \boldsymbol{r}, \boldsymbol{R}\boldsymbol{O}_i\}_{i \in \{1 \dots N\} \setminus \{l+1, \dots, l+m\}}$ denote the transformed structure.

Proof. By Lemma 5, we know that the encoder network produces invariant representations. Therefore, the MLP for predicting amino acid types that transforms the invariant representations into a probability over 20 categories is invariant, so F is invariant.

The MLP for predicting local coordinate changes $\mathrm{MLP}_G(\boldsymbol{h}_i)$ is invariant. The local coordinate change is converted to the global coordinate change using the following rule:

$$\hat{\epsilon}_j = \mathbf{O}_j^t \,\mathrm{MLP}_G\left(\boldsymbol{h}_j\right).$$

By Lemma 3, the above rule is equivariant to rotations, and hence G is equivariant to rotations.

Similarly, the MLP for predicting changes in orientation $\mathrm{MLP}_H(\boldsymbol{h}_i)$ is invariant. The changes is applied to the original orientation by:

$$\widehat{\mathbf{O}}_j^{t-1} = \mathbf{O}_j^t \boldsymbol{M}_j,$$

which is equivariant to rotations according to Lemma 3. Therefore, H is equivariant to rotations. □

D Sampling Algorithms

D.1 Backbone Atoms and Sidechain C_β Construction

The coordinates of backbone atoms (N, C_α, C, O) and sidechain C_β can be determined by the orientation and the C_α position of an amino acid because the geometry of these atoms is inflexible [Liljas et al., 2016]. To construct the position of N, C_α, C, and C_β for the i-th amino acid, we use the following formula:

$$\boldsymbol{x}_i^{\text{atom}} = \boldsymbol{O}_i \boldsymbol{c}^{\text{atom}} + \boldsymbol{x}_i, \quad (\text{atom} \in \{\text{N}, \text{C}_\alpha, \text{C}, \text{C}_\beta\}) \tag{40}$$

where $\boldsymbol{O}_i$ and $\boldsymbol{x}_i$ is the model-predicted amino acid orientation and C_α position. $\boldsymbol{c}^{\text{atom}}$ is the local coordinate derived from experimental data relative to the orientation and the C_α position, as shown in the following table.

Atom	c_x	c_y	c_z
N	-0.526	1.361	0.000
C_α	0.000	0.000	0.000
C	1.525	0.000	0.000
C_β	-0.500	-0.733	-1.154

The position of O depends on the ψ angle of the amino acid, which relies on the next amino acid in the sequence. Therefore, after constructing backbone atoms, we calculate the ψ angle for each amino acid ($\psi_i = \text{Dihedral}(\text{N}^i, \text{C}_\alpha^i, \text{C}^i, \text{N}^{i+1})$), and use the following rule to construct O coordinates:

$$\boldsymbol{x}_i^{\text{O}} = \boldsymbol{O}_i \boldsymbol{c}^{\text{O}}(\psi_i) + \boldsymbol{x}_i, \tag{41}$$

where

$$\boldsymbol{c}^{\text{O}}(\psi_i) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi_i & -\sin\psi_i \\ 0 & \sin\psi_i & \cos\psi_i \end{bmatrix} \begin{bmatrix} 2.151 \\ -1.062 \\ 0.000 \end{bmatrix}. \tag{42}$$

D.2 Sidechain Packing and Full Atom Refinement

We use `PackRotamersMover` in PyRosetta [Chaudhury et al., 2010] to pack sidechains only for amino acids on the generated CDR. The packing program is based on the Dunbrack 2010 rotamer library [Shapovalov and Dunbrack Jr, 2011] and the `REF2015` energy function [Alford et al., 2017].

After packing sidechains, we refine the structure with OpenMM [Eastman et al., 2017]. Specifically, we first use PDBFixer to prepare the structure for refinement. We minimize the potential energy of the structure. The potential energy is AMBER99SB force field plus quadratic constraint terms that restrain the position of atoms outside the generated CDR.

E Source Code

Code and data are available at `https://github.com/luost26/diffab`