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

Anonymous Author(s)
Affiliation
Address
email

Appendix

Table of Contents

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

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 [8]:

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

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 [7]:

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

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

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 such that we can use the standard normal distribution with zero-mean and unit-variance as the prior. 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 the 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: Aixs-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 ($\theta \in \mathbb{R}$). For more details about the axis-angle representation, we refer the reader to [6, 18, 19].

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** [6], 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 [6]:

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

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

Exponential of Skew-Symmetric Matrices The inversion of rotation matrix logarithm is the exponential of skew-symmetric matrices. Derived from the definition of matrix exponential, the conversion formula is [6]:

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

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 length equals to 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 the identity transform.

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 denotes 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 [13, 14, 11]:

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

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

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

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 efficiently done by first sampling from the 3D standard Gaussian distribution and then normalize 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 (with in 1024 steps). 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. To draw samples from the discretized distribution, we first randomly select a bin according to their weights. 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, so that we can draw sample efficiently. When σ is smaller than 0.1, we approximate the distribution using the truncated Gaussian distribution whose mean is 2σ and 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 $p_{\text{angular}}(\theta|\sigma^2)$, 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 $\mathbb{S}^3$ [16]. 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 denotes by $\boldsymbol{e}_i^{\text{coord}}$. If some heavy

atoms are missing, their local coordinates are filled by 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 an amino acid, including ϕ, ψ, and ω [12, 9], is 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 [17].

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

Clarification of Notations The notations F, G, and H do *not only* denote the MLPs following the encoder that output 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.

Errata On Line 219, the update rule for orientations should be $\widehat{\mathbf{O}}_j^{t-1} = \mathbf{O}_j^t \boldsymbol{M}_j$, and H is equivariant rather than invariant.

Eq.16 should be $H(\boldsymbol{R}\mathcal{R}^t + \boldsymbol{r}, \boldsymbol{R}\mathcal{C} + \boldsymbol{r}) = \boldsymbol{R}H(\mathcal{R}^t, \mathcal{C})$, as the predicted orientations are equivariant.

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.*

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

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:

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

The formulas for computing dihedral angles (ϕ_i,ψ_i,ω_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 showed 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:

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

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, such network consisting of multiple attention layers is invariant. □

Proposition 1. *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 almost inflexible. To construct the position of N, C_α, C, and C_β for the i-th amino acid, we use the following formula:

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

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

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

where

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

D.2 Sidechain Packing and Full Atom Refinement

We use `PackRotamersMover` in PyRosetta [3] to pack sidechains only for amino acids on the generated CDR. The packing program is based on the Dunbrack 2010 rotamer library [15] and the `REF2015` energy function [1].

After packing sidechains, we refine the structure with OpenMM [5]. 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 non-CDR atoms.

E Experiments

E.1 Dataset Curation

We curated additional structures similar to antibody-antigen complexes from PDB. Before finding these structures, we select PDB entries that match the following rules: (1) resolution better than 3.5Å, (2) not containing nucleic acids, (3) total number of residues no more than 2000, (4) not appearing in SAbDab.

Next, we identify loops using DSSP [10]. A loop is defined as a continuous sequence of amino acids that do not form helices or strands. For such loop-forming residues, DSSP marks the secondary structure of them as turn (`T`), bend (`S`), or none (`-`). In addition, we also consider amino acids in regions with secondary structures no longer than 3 amino acids as loop-forming. We find subsequences that only contain loop-forming residues. Then, we retain loops that meet all of the following criteria: (1) containing at least 5 amino acids and at most 20 amino acids, (2) containing at least one amino acid that interacts with at least one amino acid on other chains (two amino acids are consider interacting if their minimum atom distance is less than 5.0Å). Finally, we cluster the loops at 50% sequence identity and remove duplicates. We use a loop along with the chains that it interacts with as an antibody-antigen-like example.

E.2 Hyper-parameters

The number of diffusion time steps T is 100. The variances of the diffusion processes for positions and rotations increase linearly from $\beta_1^{\text{pos,ori}} = 0.00001$ to $\beta_T^{\text{pos,ori}} = 0.02$. For amino acid type diffusion, the variances increase linearly from $\beta_1^{\text{seq}} = 0.00001$ to $\beta_T^{\text{seq}} = 0.1$. The number of dimensions for amino acid representations is 128 and for pairwise embeddings is 64. The encoder consists of 6 attention layers. Each attention layer has 12 heads, and the number of key-query-dimensions is 32. We train the model on a single NVIDIA A100 GPU for 300K iterations (an iteration is one forward and one backward pass). The structure for fed to the model is cropped by *k-nearest-neighbor*. 256 amino acids nearest to the training CDR are kept. The batch size is set to 16 and the peak GPU memory usage is 38GB. Training data points from SAbDab and PDB are sampled 1:1 in a batch. We use the Adam optimizer and the learning rate is 0.0001. The learning rate starts to decay at the 40K-th iteration. It decays by 0.8 every 10K iterations for 16 times.

E.3 Interaction Energy

The interaction energy (relative binding free energy) between two groups of molecules is defined as the difference of free energy between the bound state and the unbound state [4]:

$$\Delta G = G_{\text{AB}} - (G_{\text{A}} + G_{\text{B}}). \tag{43}$$

In the setting of this work, A and B denote antibody and antigen respectively. G_{AB} is the energy of the antibody-antigen complex. G_{A} and G_{B} are the energies of the antibody alone and the antigen alone.

We use the Rosetta energy function `REF2015` [1] to estimate G's for the whole complex (G_{AB}), the antibody alone (G_{A}), and the antigen alone (G_{B}). Before calculation, we use the `FastRelax` routine provided by Rosetta to further refine sidechains in order to get a better energy estimation [2]. Finally, we apply Eq.43 to get the interaction energy.

E.4 Code and Data Availability

Code and data of this work will be available once the paper is made public

E.5 Additional Results

Table 4: The performance of the baselines and our method in the fix-backbone design task for CDR-Ls. Supplement to Table 2.

	IMPROVE% (%, ↑)			AAR (%, ↑)		
CDR	L1	L2	L3	L1	L2	L3
FixBB	**31.49 (1.1)**	21.83 (0.1)	25.17 (0.3)	30.76 (0.2)	26.11 (0.1)	17.33 (0.1)
AR	25.23 (3.2)	40.67 (2.0)	**30.46 (1.7)**	66.09 (2.8)	**73.45 (1.3)**	52.60 (3.8)
Ours	23.90 (1.7)	**43.83 (5.4)**	29.84 (3.2)	**68.56 (1.1)**	66.75 (2.4)	**52.78 (1.5)**

References

[1] 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.

[2] 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.

[3] 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.

[4] Zoe Cournia, Bryce Allen, and Woody Sherman. Relative binding free energy calculations in drug discovery: recent advances and practical considerations. *Journal of chemical information and modeling*, 57(12):2911–2937, 2017.

[5] 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.

[6] 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.

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

[8] 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.

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

[10] Wolfgang Kabsch and Christian Sander. Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers: Original Research on Biomolecules*, 22(12):2577–2637, 1983.

[11] 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.

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

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

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

[15] 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.

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

[17] 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.

[18] Wikipedia. Axis–angle representation — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Axis%E2%80%93angle%20representation&oldid=1081876619`, 2022. [Online; accessed 22-May-2022].

[19] Wikipedia. Rotation matrix — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Rotation%20matrix&oldid=1084060907`, 2022. [Online; accessed 22-May-2022].