PNAS

Supporting Information for

AttnPacker: An end-to-end deep learning method for protein side-chain packing

Matthew McPartlon and Jinbo Xu

Jinbo Xu
E-mail: Jinbo.Xu@gmail.com

This PDF file includes:

Supporting Information Text

Supplementary Material

Contents

List of Figures

List of Tables

A. Data Collection.

A.1. Side-Chain Packing. SCWRL4(ver 4.02) and FASPR were run with default configurations. Rosetta's fixbb application was run with non-flexible backbone coordinates and the maximum number of rotamers by passing *-EX1*, *-EX2*, *-EX3* and *-EX4* flags. We also included flags *-packing:repack_only* to disable design, *-no_his_his_pairE*, and *-multi_cool_annealer 10* to set the number of annealing iterations - these settings are recommended in the rosetta tutorial. We ran Rosetta Packer 5 times for each target protein using Rosetta's *ref2015* energy function and selected the conformation with the lowest energy. DLPacker was run using the pre-trained release from the author's github, downloaded on Sept. 17th, 2021. Side-chains were reconstructed in non-increasing order of the number of atoms in the corresponding amino acid's microenvironment.

AlphaFold2-predicted structures were generated with ColabFold(1) using default settings and a single template PDB containing only backbone heavy atoms from the corresponding native target. We ran inference twice for each target, once using ColabFold's default MSA generation(2), and once without MSA information, by passing the *single-sequence* flag. For both settings, we chose the predicted structure having the highest predicted lDDT score for analysis. We note that AMBER relaxation was not performed on any of the AlphaFold2 predictions. Omegafold source code was downloaded from the official GitHub repo, and the standard inference script was used. CASP14 predictions for RosettaFold are linked on the official GitHub page*. The first RosettaFold model for each target was selected for comparison.

A.2. Fixed Backbone Design. To produce Rosetta sequence and side-chain co-designs, we follow Rosetta's FastDesign protocol, which performs a combination of rotamer packing and sequence design. To generate a res_file, we specify *pack=True, design=True, input_sc=False*, which corresponds to enabling side-chain prediction, designing sequence, and ignoring input side chains. We then relax each design and native structure with five separate trajectories, taking the average energy value over each run. When relaxing native structures and designs, we use the default FastRelax protocol, and specify the flags *-use_input_sc* to ensure initial side-chain conformations are considered, and *-relax:coord_constrain_sidechains -relax:coord_cst_stdev <SD>*, where *SD=0.1*, to penalize deviation from side-chain rotamer predictions. The source code used for these procedures is available in our official GitHub repo.

Inference with ProteinMPNN was performed using source code from the author's google colab notebook. During inference, we did not add any noise to input coordinates, as we noticed this hurt performance. We ran inference on full-backbone atom models, trained with 0.02Å, 0.01Å, 0.002Å, and set the sampling temperature to 0 (i.e. selected the amino acid type with the highest predicted probability for each position). Sequences for Rosetta (RosettaDesign) were generated using the same methodology as for side-chain packing but with sequence design enabled. Designs for GVP-GNN + CATH4.2 were generated with the inference script and pre-trained model from the official github repo, downloaded on March 13th, 2022, and trained on the CATH4.2 dataset, curated by Ingraham et al.(3). To provide a more fair comparison with this method, we retrained GVP-GNN from scratch on our BC40 training dataset and report separate results. We refer to the latter variant as GVP-GNN+BC40.

B. Training Details.

We train our models for 10 epochs with an initial learning rate of 10^{-3}. The learning rate is decreased by a factor of two every three epochs, and we do not use any warm-up.

To optimize our models, we use Adam(4) with parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$, and use a minibatch size of 16. To stabilize training and avoid using learning rate warm-up schedules, we use ReZero(5), for every residual connection in our transformer blocks. We also apply gradient clipping by global norm(6) to clip the gradients of each example in a minibatch to have ℓ_2 norm at most 1.

We apply gradient checkpointing on the triangle attention logits and TFN kernel outputs. This yields a massive decrease in memory consumption during training, at a cost of a $\approx 50\%$ decrease in speed. Details are provided in Section H. Overall, each model was trained for roughly six days on a single Nvidia RTX A6000 GPU.

Each protein in our training set is cropped to at most 400 residues. For a protein with L residues, $r_1, \ldots r_{L-1}$, $L > 400$, we randomly select a contiguous subset by sampling $t \sim [1, L + 1 - 400]$, and then choose $r_t, \ldots, r_{400+t-1}$ as the training crop.

B.1. Overview of Hyperparameters. We tried to maintain consistent hyperparameters for all models. We mainly tuned parameters for model depth, number of nearest neighbors, number of attention heads, head dimension, and the distance at which residues or pair features should be considered neighbors. We settled upon the hyperparameter values listed in Table S1. In choosing the parameters, we aimed to balance memory usage with model capacity in each submodule. The final settings required ~32GB of GPU memory during training when full triangle updates are used (this is based on a maximum sequence length of 400 residues).

* download link: https://files.ipd.uw.edu/pub/RoseTTAFold/casp14_models.tar.gz

	Graph Transformer Full Tri. Updates *(residue, pair)*	**Graph Transformer** Local Tri. Updates *(residue, pair)*	**TFN-Transformer** *(scalar, point)*
Depth	12	12	8
Hidden Dim.	256, 128	256, 128	200, 16
Num. Attention Heads	8, 4	8,4	12, 12
Head Dim.	32, 32	32, 32	16, 4
Neighbor Distance Cutoff	N/A	16, 16	16
Max. Nearest Neighbors	N/A	30, 30	16

Table S1. Hyperparameters used for our models (unless otherwise specified). A description of input feature dimensions can be found in Section C.

B.2. Model Loss . Four separate loss functions are applied to the output - one for each of the predicted residue, pair, and coordinate features. When training with missing residue types, cross-entropy (NSR) loss is computed on predicted residue features after applying a shallow feed-forward network to produce residue-type logits. Notably, we compute loss on the predicted residue feature even if the corresponding amino acid type was included in the input since we corrupt each residue type uniformly at random with probability $p = 0.05$

We also compute an auxiliary loss over predicted distances of distal side-chain atoms (see 'tip-atom' defined in (7)). This term is applied to the pair output of the locality-aware graph transformer. The pair output is first symmetrized, and then logits are obtained by linearly projecting into 46 bins covering $2\text{Å} - 20\text{Å}$. Two bins are also added for a predicted distance less than 2Å and greater than 20Å. If locality-aware attention is used, the loss is only computed for pair ij if the corresponding residues are adjacent. Pairwise distograms are obtained by taking a softmax of the logits, and an averaged cross-entropy loss is then applied.

The final loss term is applied to the predicted coordinates. For every residue, we predict the positions of all 33 side chain atom types and only compute loss on the coordinates which are present in the native structure. Let $C_j^{(i)}$ denote the j^{th} side-chain atom coordinate for residue i in the native structure. Define $\hat{C}_{ij}^{(i)}$ analogously for the predicted structure. The loss is computed as

$$L^{(i)} = \text{mean}_j \left(\text{huber} \left(C_j^{(i)}, \hat{C}_j^{(i)}, \beta = 0.25 \right) \right) \quad [1]$$

Where $\text{huber}(x, y, \beta)$ is the Huber loss between x and y with smoothing parameter β. The final loss is computed as $\text{mean}_i \left(L^{(i)} \right)$.

Some care must be taken in computing Equation (1) since some residues have symmetric sidechains. For these residues, we consider all possible symmetries by swapping the coordinates of symmetric atoms and take the lesser of the swapped and not-swapped loss for the respective residue. A list of residues with symmetric sidechains and pairs of atoms for which we swap coordinates can be found in Table S2.

Arg	**Asn**	**Asp**	**Gln**	**Glu**	**Leu**
NH1,NH2	OD1,ND2	OD1, OD2	OE1, NE2	OE1, OE2	CD1, CD2

His	**Leu**	**Phe**	**Tyr**	**Val**
ND1, CD2	CD1, CD2	CD1,CD2	CD1, CD2	CG1, CG2
NE2, CE1	-	CE1,CE2	CE1, CE2	-

Table S2. Symmetric Sidechains. Amino acids with sidechain symmetries and the atom pairs which constitute these symmetries.

The overall loss function is given by

$$\mathcal{L} = 1 \cdot \mathcal{L}_{\text{coord}} + 0.2 \cdot \mathcal{L}_{\text{dist}} + 0.15 \cdot \mathcal{L}_{\text{plDDT}} + 0.15 \cdot \mathcal{L}_{\text{seq}}. \quad [2]$$

We note that little time was spent in tuning the weights of our loss function.

C. Model Input.

C.1. Input Features. Our model uses only input features derived directly from primary sequence and backbone coordinates. An overview of input feature types and the corresponding shape can be found in Table S3.

Features Name & Shape	Description
res_type $[L]$	A number in the range 0 to 21 representing the corresponding amino acid type, a gap, or a missing residue
bb_dihedral $[L, 3]$	Backbone phi, psi, and omega dihedral angles.
seq_pos $[L]$	The sequence index of the respective residue, from 1 to L
centrality $[L]$	The number of C_β atoms within 16Å of residue of the respective residue's C_β atom. For this procedure, $C\beta$ atoms are imputed according to Equation (3).
atom_distance $[L, L, 3]$	One-hot encoded binned distance between $C_\alpha - C_\alpha$, $C_\alpha - N$, and $N - O$ atoms in each residue. Each bin represents a distances from 2Å-20Å with two separate bins used for distances falling outside of this range.
tr_orientations $[L, L, 3]$	Dihedral and planar angles defined by Yang et al(8).

Table S3. Input features used by AttnPacker. The shape of the corresponding type for a protein with L residues is shown below each feature.

We use standard residue-level features and encodings for our input. These include a representation of the amino acid type, binned relative sequence position, and embeddings/encodings of backbone dihedral angles. Less standard is the inclusion of residue centrality. The use of centrality-based encodings was studied in (9), where the authors found that transformers significantly benefit from the addition of centrality encodings with graph-like data. We choose to incorporate this information at the input level rather than in each attention block (as is proposed in (9)). Originally, we included SS3 secondary structure as part of our input but found that this feature gave the same (if not slightly worse) results in terms of average residue RMSD scores.

For our pair embedding, we follow a scheme similar to (10), but also incorporate residue pair information, using two separate embeddings for residue types. Denoting the embeddings as E_A and E_B,, the pair feature for residues i and j of type r_i and r_j (resp.) is given by the outer-sum of $E_A(r_i)$ and $E_B(r_j)$. The authors further augment this information by adding an embedding of the relative sequence separation between the two residues, and we follow the same approach here.

To generate $C\beta$ atom positions for orientation and centrality, we impute a unit vector in the direction $C\beta_i - C\alpha_i$ before computing the respective features. The imputed vector is calculated as in (11) using

$$\sqrt{\frac{1}{3}}\langle \mathbf{n} \times \mathbf{c} \rangle - \sqrt{\frac{2}{3}}\langle \mathbf{n} + \mathbf{c} \rangle \qquad [3]$$

where $\langle x \rangle = x/\|x\|_2$, $\mathbf{n} = N_i - C\alpha_i$, and $\mathbf{c} = C_i - C\alpha_i$.

Procedure	Description
bin_centrality(c) : $\mathbb{N} \to [0..6]$	Mapping given by $c \mapsto \min\left(\lfloor \frac{c}{8} \rfloor, 6\right)$
bin_angle(θ) : $(-\pi, \pi) \to [0..35]$	Mapping given given by $\theta \mapsto \lfloor 36 \cdot \frac{\theta+\pi}{2\pi} \rfloor$
encode_angle(θ) : $(-\pi, \pi) \to [0, 1]^2$	Mapping given by $\theta \mapsto (\cos\theta, \sin\theta)$
bin_rel_pos(p) : $[1..L] \to [0..9]$	Mapping given by $p \mapsto \lfloor 10 \cdot \frac{p-1}{L} \rfloor$
bin_rel_sep(s) : $[0..L-1] \to [0..48]$	Maps the (signed) sequence separation s to the index of a predefined interval. We chose intervals [0, 1), [1, 2), [2, 3), [3, 4), [4, 5), [5, 6), [6, 7), [7, 8), [8, 10), [10, 12), [12, 15), [15, 20), [20, 30), [30,∞) plus the negation of each interval.
bin_dist(d) : $\mathbb{R}^{\geq 0} \to [0..33]$	Maps the pairwise distance d to one of 32 equal width bins in the range 2Å..20Å. Distances less than 2Å and distances greater than 20Å are placed into separate bins.

Table S4. Description of Input Embedding Procedures (referenced in Figure S1). The name of the procedure (top), along with the domain and range (bottom) are given in the first column. Here, we use L to denote the protein length.

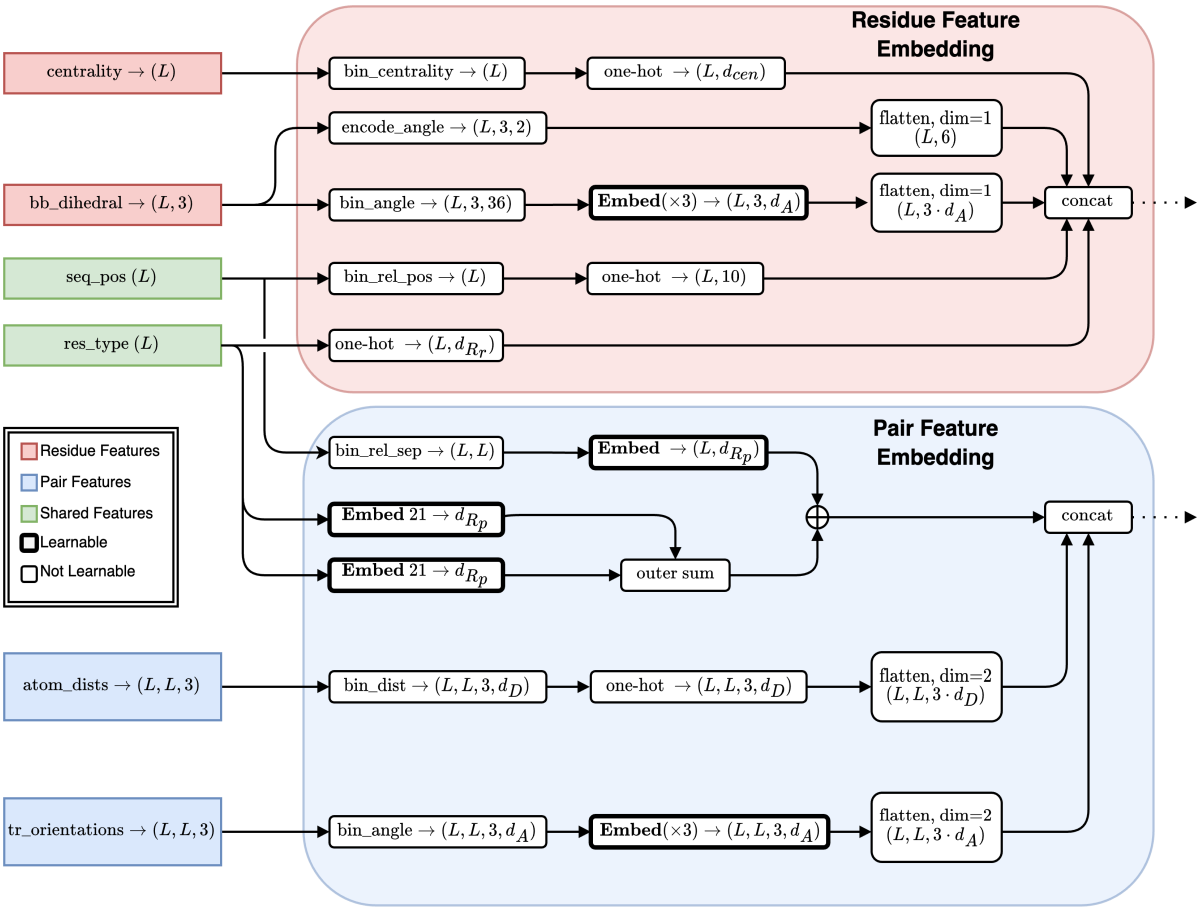

Fig. S1. Overview of input feature embedding. The feature shape is shown in parentheses. Residue and pairwise features are embedded separately. Only information derived from primary sequence and backbone coordinates is considered. Full explanations of procedures referenced in this figure are given in Table S4. For residue features in our final models, we use $d_{cen} = 7$ for the one-hot encodings of centrality features, $d_A = 6$ for our backbone torsion angle embedding dimension, and $d_{R_r} = 32$ for residue type embedding dimension. For pairwise features, we use $d_D = 34$ for each one-hot distance encodings, $d_{R_p} = 48$ for residue pair embeddings, and $d_A = 6$ for embeddings of trRosetta orientations.

C.2. Rotamer Conditioning. It is also possible to condition AttnPacker on partial rotamer information. The implementation of AttnPacker described in this work does not support this, but we offer a variant with this functionality in our GitHub repo.

Rotamer conditioning can be performed by passing the coordinates as input to the TFN-Transformer, or by passing an invariant encoding of the side-chain dihedrals to the graph transformer. We opted for the latter, using sine and cosine encodings of each side-chain dihedral $\chi_1 - \chi_4$ when present and using 0 otherwise. The training procedure is identical, except that we select a random subset of unmasked residues (i.e., residues for which sequence identity is known) and provide these dihedrals as input.

D. Confidence Prediction. Residue output features are used to predict the per-residue local distance difference test scores (plDDT).

$$\text{plDDT}_i \quad = \quad \frac{1}{|\mathcal{N}(i)|} \cdot \sum_{j \in \mathcal{N}(i)} \text{lDDT}\left(\text{abs}\left(d_{ij} - d^*_{ij}\right)\right) \qquad [4]$$

where d_{ij}, d^*_{ij} are the predicted and ground truth distance between atoms i and j,

$$\mathcal{N}(i) = \left\{j : d^*_{ij} < 12\text{Å}\right\},$$

and

$$\text{lDDT}(d) = 100 \cdot \frac{1}{4} \cdot \sum_{k=0}^{3} \mathbf{1}_{d \leq 2^{k-1}} \qquad [5]$$

To compute plDDT loss, we pass our output residue features through a shallow feedforward network with output representing 25 equal-width binned log likelihoods in the range $[0, 1]$. The predictions are compared with the ground-truth labels plDDT_i by cross-entropy loss. Atom types used for each residue are given in Table S5

ALA	ARG	ASN	ASP	CYS	GLN	GLU	GLY	HIS	ILE
CB	*CZ*	*CG*	*CG*	*SG*	*CG*	*CG*	*CA*	*CG1*	*CG*
LEU	**LYS**	**MET**	**PHE**	**PRO**	**SER**	**THR**	**TRP**	**TYR**	**VAL**
CG	*CD*	*CG*	*CG*	*CG*	*OG*	*OG1*	*CG*	*CG*	*CG2*

Table S5. Definition of the distal side-chain atom for each residue type.

Donors	OH OG OG1 N ND1 ND2 NE NE1 NE2 NH1 NH2 NZ
Acceptors	O OD1 OD2 OE1 OE2 OH OG OG1 ND1 NE2

Table S6. Hydrogen Bond Donor and Acceptor atoms. Note that some atoms may act as both donors and acceptors and some atoms may appear in multiple residues

E. Post-Processing Procedure. Directly predicting side chain coordinates can sometimes result in physically unrealistic conformations. To ensure the validity of our model's side chain predictions, we developed a post-processing procedure that projects side chain coordinates onto a continuous rotamer set, while simultaneously minimizing both steric clashes and RMSD.

Algorithm 1 Post-Process Predictions

Input

$\mathbf{x}_i^a$: All-atom coordinates for each residue, in $\mathbb{R}^{n\times 37\times 3}$
χ_i^t: Side chain dihedrals for each residue i
$\mathbf{S}_i$: Amino acid type of each residue i
t_{vdw}: Steric clash tolerance parameter, in $\mathbb{R}$

Output

$\mathcal{L}_{\text{RMS}}$: RMSD loss
$\mathcal{L}_{\text{steric}}$: Steric clash loss

function ProjectRotamerStep($\mathbf{x}_i, \chi_i^t, \mathbf{s}_i, t_{\text{vdw}}$) :

$\hat{\mathbf{x}}_i^a = \text{ToAllAtomCoordinates}\left(\mathbf{x}_i^{\{N,C\alpha,C}\}, \chi_i^t, \mathbf{S}_i\right)$

$d_{i,j}^{a,a'} = \left\|\hat{\mathbf{x}}_i^a - \mathbf{x}_j^{a,a'}\right\|_2$

$\mathcal{L}_{\text{RMS}} = \text{mean}_{i,a}\left(d_{i,i}^{a,a}\right)$

$\mathcal{L}_{\text{steric}} = \sum_{i,j,a,a'} \text{ReLU}\left(\text{vdW}\left(i,j,a,a'\right)\cdot t_{\text{vdw}} - d_{i,j}^{a,a'}\right)$

return $\mathcal{L}_{\text{RMS}}$, $\mathcal{L}_{\text{steric}}$

In Algorithm 1, $\text{vdW}\left(i,j,a,a'\right)$ is a pre-computed table giving the minimum distance at which atoms a, a' in residues i and j are considered to clash. In computing this table, we consider only atom pairs separated by at least four bonds, where at most one of a and a' is a backbone atom (as backbone flexibility is not modeled by our algorithm). We also subtract 0.4 Å from hydrogen donor and acceptor pairs, given in Table S6. Last, we set the clash tolerance to 1.8 Å for pairs of Cysteine sulfur atoms to account for disulfide bonds (typically observed at a distance of 2.05 Å). the The function ToAllAtomCoordinates converts backbone frames and side-chain torsion angles to 3D-coordinates, following (12, Supplementary Information, Algorithm 24). In converting to 3D coordinates, we use idealized bond lengths and angles for each residue type, taken from OpenFold(13). We also use code from OpenFold to compute side-chain dihedral angles. At the start of post-processing, we initialize each dihedral according to our coordinate predictions and update dihedrals χ_i^t according to the gradients of the steric and RMSD loss. We remark that this procedure can also be performed on native structures, and we find an average side-chain RMSD difference of 0.08Å before and after post-processing CASP13 native structures.

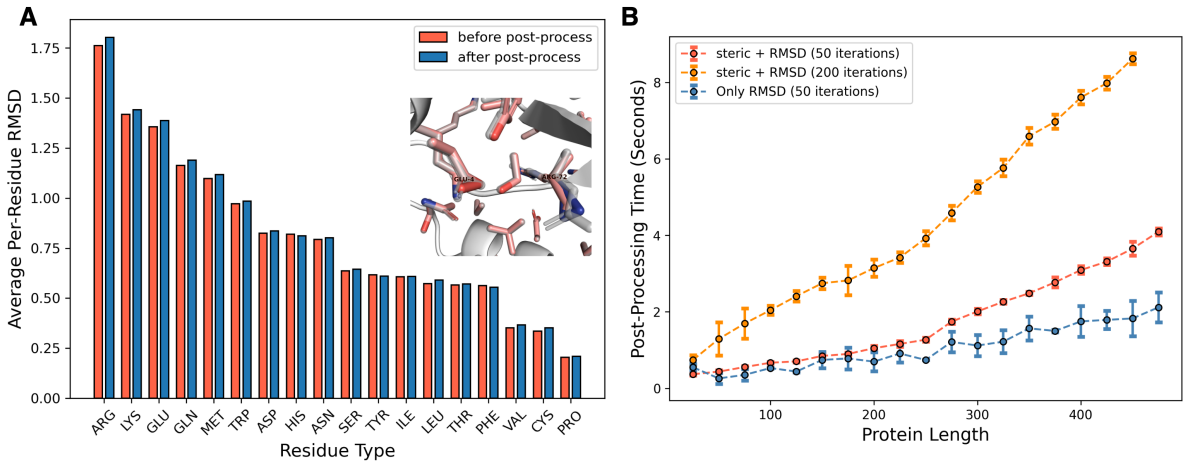

Fig. S2. Post-Processing Procedure. (**A**) Bar plot of average RMSD of each residue type before (red) and after (blue) rotamer and steric clash optimization. We also add a small illustration of side-chain conformations before (red) and after (blue) post-processing on CASP target T1043. (**A**) Running time of the post-processing procedure (y-axis) against input protein length (x-axis) for three different settings. For example, the red line shows the amount of time taken to run 50 LBFGS(14) optimization steps with both steric clash and rotamer RMSD loss. On average, 50 optimization steps will remove all clashes at a 0.9 tolerance threshold.

We find that side-chain RMSD increases only slightly after post-processing (see Figure S2A). Upon visually inspecting the raw coordinates output from our method, we found that the majority of residue side-chains were predicted with near-ideal geometry. Most violations occurred for bulky amino acids like Arg or Glu, where the two branching atoms at terminal groups had short bond lengths (illustrated in Figure S2B). This is likely a side-effect of training with RMSD loss, as the last dihedral is difficult to predict for these residues, and shortening bond lengths can reduce the RMSD of incorrect predictions. Finally, we remark that our post-processing procedure requires only a PDB file as input and is broadly applicable for a range of other tasks.

F. Inference on Large Proteins. AttnPacker can be used to pack side-chains for proteins of arbitrary length. To do this, we linearly crop the input structure and sequence into overlapping segments of a predefined length N, where consecutive segments S_i, S_{i+1} overlap on the last and first $N/2$ residues, respectively. For residues contained in multiple segments, we choose the side chain/sequence predictions with higher predicted confidence scores.

G. Network Architecture.

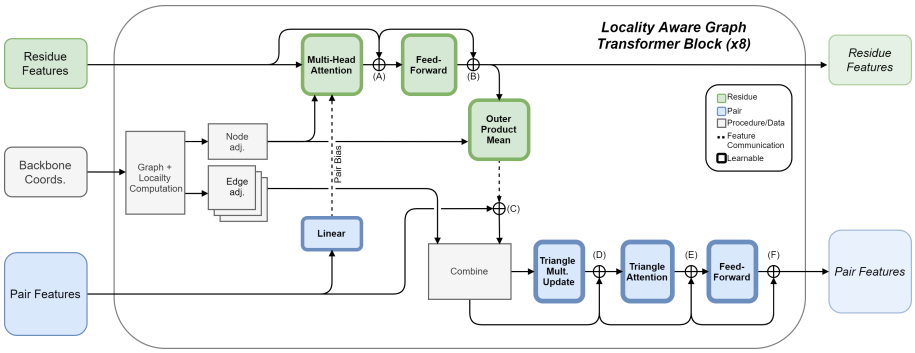

Fig. S3. Architecture of Locality Aware Graph Transformer Block. Standard multi-headed graph attention is used to update node features while edges between adjacent nodes are updated with triangle multiplication and attention. For each triangle update, edge adjacency information is used to select only those triangles having maximum side-length less than a fixed threshold.

G.1. Locality Aware Triangle Attention. Consider a set of points $P = \{p_i\}_{i=1..n} \subseteq \mathbb{R}^d$. Define

$$\Delta\left(P;\theta\right) := \left\{(i,j,k) : \|p_x - p_y\| \leq \theta \;\forall x,y \in (i,j,k)\right\}, \quad [6]$$

the set of triangles in P with maximum side length at most θ.

We develop a hypergraph approach to performing triangle updates $\mathcal{G}\left(P;\theta\right) = (V,E,d)$, where

$$V := V\left(P;\theta\right) = \left\{v_{ij} : \|p_i - p_j\|_2 \leq \theta,\; p_i,p_j \in P\right\} \quad [7]$$

$$d\left(v_{ij},v_{jk}\right) = \max\left(\left\{\|p_x - p_y\|_2 : x,y \in (i,j,k)\right\}\right) \quad [8]$$

The edge set E is defined by the adjacency relation

$$\mathcal{N}_{\mathcal{G}}\left(v_{ij}\right) := \mathcal{N}_{\mathcal{G}}\left(\left(i,j\right), P; \theta\right) = \left\{\left\{v_{ik}, v_{jk}\right\} : \max\left(d\left(v_{ij}, v_{ik}\right), d\left(v_{ij}, v_{ik}\right)\right) \leq \theta\right\} \quad [9]$$

Observe that there is a node for every pair of points within a distance θ, and the distance between two nodes sharing a common underlying point is the maximum edge length on the corresponding triangle.

When the underlying point set consists of backbone atom coordinates for a protein, a reasonable choice of θ results in a relatively small set of triangles. Furthermore, to efficiently compute triangle updates for each node v_{ij}, we restrict $\mathcal{N}_{\mathcal{G}}\left(v_{ij}\right)$ to the N_e nearest neighbors under $d\left(\cdot,\cdot\right)$. This yields a 3-uniform, $O\left(N_e\right)$- regular hypergraph with as many nodes as there are underlying points within distance θ. We denote the latter quantity by N_v.

In theory, the pairwise features selected for triangle attention can be further restricted to the (at most) N nearest neighbors of each residue, as only the pair features for each residue's N nearest neighbors are used to bias residue attention logits. In practice, we restrict pair features to the (at most) $2N$ nearest neighbors of each residue. More formally, the vertices of our triangle hypergraph consist of pairs (i,j) such that j is among the top $2N$ nearest neighbors of i. Triangle multiplication and attention updates then require time and space proportional to $O\left(N_v \cdot N_e\right) = O\left(n \cdot N_e^2\right)$ in total. These updates are identical to that of AlphaFold2 but restricted to this subset of triangles.

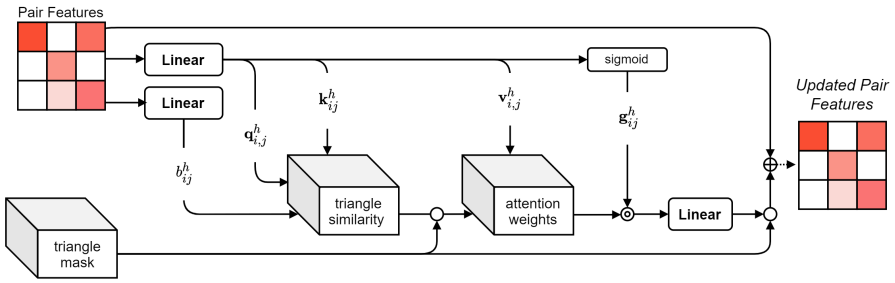

Fig. S4. A simple implementation of local triangle attention updates. A mask of dimension $L \times L \times L$ is used to control which triangles (and respective edge features) are updated during attention calculations. Masked positions are kept unchanged, and unmasked positions are filled with a large negative value (-10^6) before softmax is applied to compute attention weights.

Alternatively, we can quickly implement local triangle updates by computing a global triangle mask for each sample (see Figure S4). This approach does not yield the time and space savings previously discussed but does serve as a simple drop-in replacement for existing models.

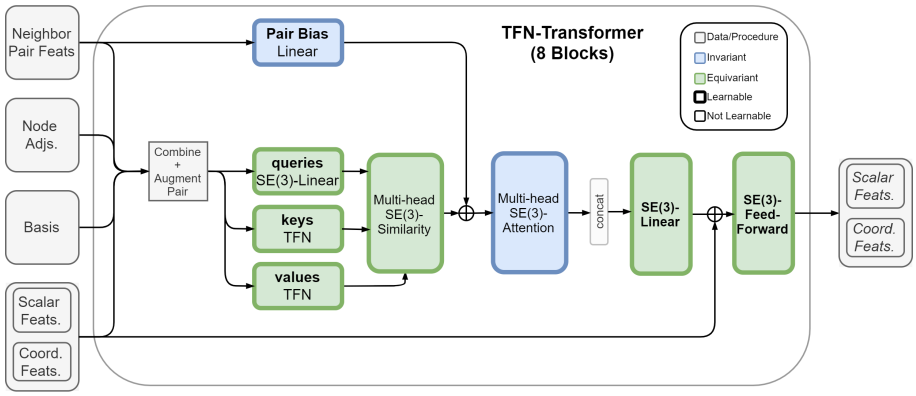

Fig. S5. Architecture of TFN-based transformer block. Keys and values are computed using TFNs. The input to each TFN radial kernel is pair features and distance information between points for the corresponding pair. Pair features are also linearly projected to bias the similarity logits for adjacent pairs of residues. The multi-headed SE(3)-similarity and SE(3)-Attention modules are detailed in Table S8. Learnable modules are displayed in bold font.

G.2. TFN-Transformer. For normalization, we propose a method similar to Layer normalization with a norm-based non-linearity. Several papers have proposed SE(3)-Equivariant Normalization schemes (e.g., (15–17)), most include some form of Layer normalization (18), or restriction on the ℓ_2 norm of coordinate features. In our experiments, we found that applying layer normalization to coordinate norms (and subsequently scaling by these values) sometimes caused instability in the early stages of training. In light of this, In Algorithm 2, we simply learn a scale and bias σ^ℓ and β^ℓ for each feature type.

In augmenting the pair features with distance information, we choose to append both normalized and un-normalized distances between the hidden points. The output of this function is passed directly to the TFN radial kernel, which employs a 3-layer MLP with GELU nonlinearity to produce pairwise kernels for each pair of input feature types.

Algorithm 2 SE(3)-Equivariant Normalization

Input
$\mathbf{x}_i^\ell$: type-ℓ features in $\mathbb{R}^{d_\ell \times 2\ell+1}$
Output
$\bar{\mathbf{x}}_i^\ell$: normalized type-ℓ features in $\mathbb{R}^{d_\ell \times 2\ell+1}$

function SE3Norm($\mathbf{x}_i^\ell$, nonlin = GELU) :
$norm_i^\ell = \text{concat}_{k=1..d_\ell}\left(\left\|\mathbf{x}_{i,k}^\ell\right\|\right)$
$t_i^\ell = \text{nonlin}\left(norm_i^\ell \circ \sigma^\ell + \beta^\ell\right)$
$\bar{\mathbf{x}}_i^\ell = \text{concat}_{k=1..d_\ell}\left(\frac{\mathbf{x}_{i,k}^\ell}{norm_{i,k}^\ell}\right) \circ t_i^\ell$
return $\bar{\mathbf{x}}_i^\ell$

Algorithm 3 Augment Edge Features

Input
$\mathbf{z}_{ij}$: pair features in $\mathbb{R}^{d_z}$
$\mathbf{c}_i^{(k)}$: hidden coordinate features in $\mathbb{R}^{d_p \times 3}$ for each residue i
r_{ij}: input relative coordinates in $\mathbb{R}^{1\times 3}$
Output
$\tilde{\mathbf{z}}_{ij}$: augmented edge features in $\mathbb{R}^{d_z + 2\cdot d_p}$

function AugmentPairFeats($\mathbf{z}_{ij}, \mathbf{c}_i^{(k)}, r_{ij}$) :
$\mathbf{d}_{ij} = \text{concat}_{k=1..d_p}\left(\left\|r_{ij} + \left(\mathbf{c}_j^{(k)} - \mathbf{c}_i^{(k)}\right)\right\|_2\right)$ ▷ $\mathbf{d}_{ij} \in \mathbb{R}^{d_p}$
$\tilde{\mathbf{z}}_{ij} = \text{concat}\left(\mathbf{z}_{ij}, \text{LayerNorm}\left(\mathbf{d}_{ij}\right), \mathbf{d}_{ij}\right)$
return $\tilde{\mathbf{z}}_{ij}$

Algorithm 4 TFN-Transformer Attention Head

Input
$\mathbf{x}_i^\ell$: type-ℓ features in $\mathbb{R}^{d_\ell \times 2\ell+1}$
$\mathbf{z}_{ij}$: pair features in $\mathbb{R}^{d_z}$
$\mathcal{B}_{ij}^{\ell,k}$: equivariant basis mapping type ℓ features to type k features
r_{ij}: input relative coordinates in $\mathbb{R}^{1\times 3}$
$\mathcal{N}_i$: List of neighbor indices for each residue $1..i$
Output
$\mathbf{o}_i^\ell$: attention head features for each input type $\ell = 0\ldots$

function TFNAttnetionHead($\mathbf{x}_i^\ell, \mathbf{z}_{ij}, \mathcal{B}_{ij}^{\ell,k} r_{ij}, \mathcal{N}_i$) :
$\mathbf{z}_{ij},\ \mathbf{x}_i^\ell \leftarrow \text{LayerNorm}\left(\mathbf{z}_{ij}\right),\ \text{SE3Norm}(\mathbf{x}_i^\ell)$
 # Compute TFN keys and values
$\tilde{\mathbf{z}}_{ij} \leftarrow \text{AugmentPairFeats}\left(\mathbf{z}_{ij}, \mathbf{x}_i^1, r_{ij}\right)$
$\mathbf{k}_{ij}^\ell, \mathbf{v}_{ij}^\ell = \text{TFN}(\mathbf{x}_i^\ell, \tilde{\mathbf{z}}_{ij}, \mathcal{B}_{ij}^{\ell,k}, \mathcal{N}_i)$ ▷ defined for i, j such that $j \in \mathcal{N}_i \setminus i$
$\mathbf{q}_i^\ell = \text{SE3Linear}(\mathbf{x}_i^\ell)$ ▷ $\mathbf{k}^\ell, \mathbf{q}^\ell, \mathbf{v}^\ell \in \mathbb{R}^{h^\ell}$
$b_{ij} = \text{LinearNoBias}(\mathbf{z}_{ij})$ ▷ pair bias, $b_{ij} \in \mathbb{R}$
$\hat{\mathbf{z}}_{ij} = \text{LinearNoBias}(\mathbf{z}_{ij}) \circ \text{sigmoid}\left(\text{LinearNoBias}(\mathbf{z}_{ij})\right)$ ▷ pair features, $\hat{\mathbf{z}}_{ij} \in \mathbb{R}^{1\times d_z'}$
$w^\ell = \frac{1}{\sqrt{(2\ell+1)\cdot h^\ell}}$ ▷ head weights for each input type $\ell = 0\ldots$
 # Compute pair similarity and self-similarity
$\sigma_{ij}^\ell = \sum_{c=1}^{d_\ell} \mathbf{q}_{i,c}^\ell \cdot \mathbf{k}_{i,j,c}^\ell$
$\sigma_{ii}^\ell = \text{SE3Linear}(\mathbf{x}_i^\ell)$
 # Share Attention weights for each i and each type
$\alpha_{ij} = \text{softmax}_{j\in\mathcal{N}_i}\left(b_{ij} + \sum_\ell w^\ell \cdot \gamma^\ell \cdot \sigma_{i,j}^\ell\right)$
$\mathbf{o}_i^\ell = \sum_{j\in\mathcal{N}_i} \alpha_{ij} \cdot \mathbf{v}_{ij}^\ell$
$\mathbf{o}_{pair,i} = \sum_{j\in\mathcal{N}_i} \alpha_{ij} \cdot \hat{\mathbf{z}}_{ij}$
 # Scalar output augmented with attention-weighted edge information
$\mathbf{o}_i^{(0)} \leftarrow \text{concat}\left(\mathbf{o}_i^{(0)}, \mathbf{o}_{pair,i}\right)$
return $\mathbf{o}_i^\ell$

In Algorithm 4, we use d_ℓ to denote the input dimension, and h^ℓ to denote the hidden dimension (head dimension) of input feature type ℓ. Each attention head has a separate learnable weight γ^ℓ for each input type ℓ. This weight is the softplus of a learnable scalar and is initialized so that $\gamma^\ell = 1$. For each input type, the output of each attention head is concatenated and linearly projected so that the output dimension matches the original input dimension. All linear projections (SE3Linear) follow the scheme proposed in (16).

We give a full description of our TFN-Transformer in Algorithm 5. Our TFN-Transformer consists of three components: (1) An equivariant mapping of scalar and coordinate input features to hidden feature types/dimensions (2) multiple TFN-based attention layers and (3) An equivariant mapping from hidden feature types/dimensions to output feature types/dimensions.

Algorithm 5 TFN-Transformer

Input

$\mathbf{s}_i$: scalar residue features in $\mathbb{R}^{s\times 1}$
c_i: backbone coordinates for each residue in $\mathbb{R}^{c\times 3}$
$\mathbf{z}_{ij}$: pair features in $\mathbb{R}^{d_z}$
$\mathcal{N}_i$: List of neighbor indices for each residue $1..i$
ℓ_{max}: number of hidden types $0..\ell_{max}$

Output

$\mathbf{s}_{out,i}$: updated scalar residue features in $\mathbb{R}^{d_{scalar}\times 1}$
$\mathbf{c}_{out,i}$: updated coordinate features $\in \mathbb{R}^{d_{coord}\times 3}$

function TFN-Transformer$(\mathbf{s}_i, \mathbf{c}_i, \mathbf{z}_{ij}, \mathcal{N}_i)$:

$r_{ij} = \mathbf{c}_{j,0} - \mathbf{c}_{i,0}$ ▷ relative coordinates $\in \mathbb{R}^{1\times 3}$
$\mathcal{B}_{ij}^{\ell,k} = \text{ComputeEquivariantBasis}\,(r_{ij}, \ell_{max})$
$\mathbf{c}_i \leftarrow \text{concat}_k\,(\mathbf{c}_{i,k} - \mathbf{c}_{i,0})$
 # Equivariant Input Mapping
 # Map to hidden dimension and hidden feature types
$\mathbf{x}_{hid,i}^\ell = \text{TFN}\left((\mathbf{s}_i, \mathbf{c}_i), \mathbf{z}_{ij}, \mathcal{B}_{ij}^{\ell,k}, \mathcal{N}_i\right)$ ▷ $\tilde{\mathbf{x}}_i^\ell \in \mathbb{R}^{d_\ell \times 2\ell+1}$, $\ell_{in} = 0, 1$ and $\ell_{out} = 0..\ell_{max}$
$\mathbf{x}_{hid,i}^\ell \leftarrow \text{SE3Norm}\left(\mathbf{x}_{hid,i}^\ell\right)$
 # Attention Layers

for $1..N_{Layers}$ **do**
  $\mathbf{x}_{res,i}^\ell = \text{TFNAttention}\left(\mathbf{x}_{hid,i}^\ell, \mathbf{z}_{ij}, \mathcal{B}_{ij}^{\ell,k}, r_{ij}, \mathcal{N}_i\right)$
  $\mathbf{x}_{hid,i}^\ell \leftarrow \text{ReZero}\left(\mathbf{x}_{hid,i}^\ell, \mathbf{x}_{res,i}^\ell\right)$
  $\mathbf{x}_{res,i}^\ell = \text{SE3FeedForward}\left(\text{SE3Norm}\left(\mathbf{x}_{hid,i}^\ell\right)\right)$
  $\mathbf{x}_{hid,i}^\ell \leftarrow \text{ReZero}\left(\mathbf{x}_{hid,i}^\ell, \mathbf{x}_{res,i}^\ell\right)$
 # Equivariant Output Mapping
 # Normalize and map to output dimension/ output types via TFN
$\mathbf{x}_{hid,i}^\ell \leftarrow \text{SE3Norm}\left(\mathbf{x}_{hid,i}^\ell\right)$
$\mathbf{x}_{out,i}^\ell = \text{TFN}\left(\mathbf{x}_{hid,i}^\ell, \mathbf{z}_{ij}, \mathcal{B}_{ij}^{\ell,k}, \mathcal{N}_i\right)$ ▷ $\ell_{in} = 0..\ell_{max},\ \ell_{out} = 0, 1$
$\mathbf{x}_{out,i}^\ell \leftarrow \text{SE3Linear}\left(\text{SE3Norm}\left(\mathbf{x}_{out,i}^\ell, \text{nonlin} = \text{Identity}\right)\right)$ ▷ $\mathbf{x}_{out,i}^0 \in \mathbb{R}^{d_{scalar}\times 1}$, $\mathbf{x}_{out,i}^0 \in \mathbb{R}^{d_{coord}\times 3}$
$\mathbf{s}_{out,i}, \mathbf{c}_{out,i} = \mathbf{x}_{out,i}^0, \mathbf{x}_{out,i}^1$
return $\mathbf{s}_{out,i}, \mathbf{c}_{out,i}$

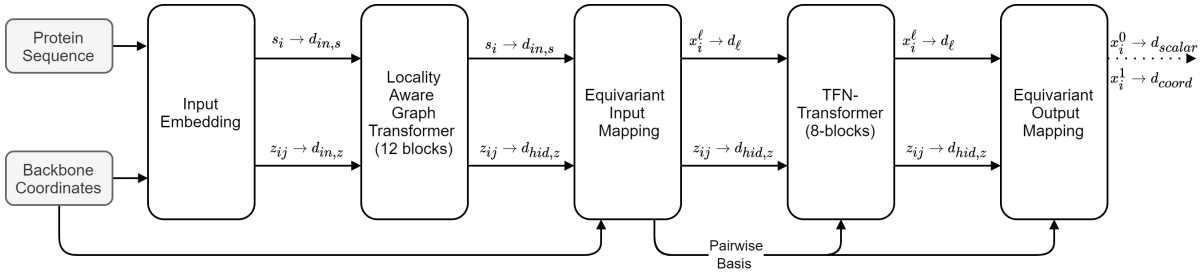

Fig. S6. Full architecture of AttnPacker. we use s_i to denote scalar (residue) features, and z_{ij} to denote pair features. Backbone coordinates are used to define residue and pair adjacencies, and included in the equivariant input mapping (see Algorithm 5). The Equivariant input mapping returns type ℓ features x_i^ℓ for user-defined types $\ell > 0$. In our model, scalar output features x_i^0 are discarded, and coordinate output x_i^1 has $d_{coord} = 32$, one channel for each possible side-chain atom type.

G.3. Full Architecture. Our full architecture is shown in Figure S6. It consists of an input embedding (Figure S1) to produce scalar features for residues and pairs, and is followed by our locality-aware graph transformer and TFN-Transformer. Pair features are only modified in the locality-aware graph transformer block. The TFN-Transformer still makes use of pair features to produce radial kernels, bias attention logits, and augment the output of each attention head.

H. Memory Consumption.

H.1. Triangle Attention. In Section G.1, we determine that the number of pair features requiring triangle updates is bounded by $2NL$, where N is the maximum number of neighbors per point, and L is the protein length. Locality aware triangle attention stores at most N attention logits per pairwise feature. It follows that the number of attention logits stored for an attention layer with h heads is $2N^2L \cdot h$. In practice, computing neighbor-wise attention can be costly. Depending on the implementation, repeating or grouping of neighbor features may cause the gradients for attention logits to require significantly more space. We experimented with several implementation, but ultimately decided to mask full triangle attention logits ($O\left(L^3\right)$ space) as a proof of concept. Masking attention logits effectively stops the gradient flow for all pair pair features which are not part of the triangle graph. This resulted in a modest improvement in memory, but more work will be needed to realize the full time and space benefits of this architecture.

	bits for attention logits using 32-bit float precision	$L = 300$	$L = 500$
Triangle Attention	$32 \cdot L^3 \cdot h \cdot D$	5.2 Gb	24.0 Gb
Locality Aware Triangle Attention	$32 \cdot 2N^2L \cdot h \cdot D$	0.10 Gb	0.17 Gb

Table S7. A comparison of memory usage for storing pair attention logits. L is the length of the input sequence. The third and fourth columns show the memory usage for an input sequence of length 300 and length 500 respectively. Values are calculated by fixing the number of heads $h = 4$, the depth $D = 12$, the number of nearest neighbors $N = 30$ per pair.

H.2. TFN Transformer Memory Analysis. One of the main drawbacks of the SE(3)-Transformer is high memory usage caused by computing equivariant pairwise kernels in each attention block. To alleviate some of this overhead, we modify the TFN implementation used in the original SE3-Transformer proposed by Fuchs et al.(15). Given input feature tensors of size (d_{in}, o_{in}),(d_{out}, o_{out}) respectively, the corresponding basis element mapping between these types has shape $(o_{in}, o_{in} \cdot o_{out})$. Let $f = \min(o_{in}, o_{out})$ denote the frequency of the mapping, then for each pair of features we require a radial kernel of size (d_{in}, d_{out}, f). To ensure equivariance, the kernel passes through the corresponding basis element, yielding an intermediate tensor of shape

$$(d_{in}, d_{out}, f, o_{in}, o_{in} \cdot o_{out})$$

That is, the kernel is obtained by multiplying the radial weights for each pair through the corresponding basis element. The input features are then multiplied through the respective kernel to yield the desired output and the process is repeated for each pair of input and output types.

As TFNs are used to produce key and value vectors in each attention block, the intermediate kernel mapping between type-ℓ features at each block requires memory proportional to

$$2 \cdot d_{in}^{\ell} \cdot h^{\ell} \cdot d^{\ell} \cdot (2\ell + 1)^4$$

where h^{ℓ}, d^{ℓ} are are the number of heads and head dimension for type ℓ features.

In our implementation, we are able to obtain a factor $\approx o_{in} \cdot o_{out}$ reduction in memory by changing change the order of matrix multiplication in the TFN kernel. Rather than multiply the radial weights through the basis, we first multiply the features through the basis, and then multiply the result with the radial weights. The memory required to store the intermediate tensors is reduced to

$$\underbrace{d_{in} \cdot d_{out} \cdot f}_{\text{radial weights}} + \underbrace{d_{in} \cdot o_{in} \cdot o_{out}}_{f_{in} B_{in \to out}}$$

This greatly reduces the memory burden of TFNs and, together with gradient checkpointing allows us to fit a much deeper and larger model on a single GPU.

I. Ablation and Architecture Assessment. We consider several variants of SE(3)-equivariant self-attention. Each variant can be categorized by the operation used to compute the similarity between keys and queries of different feature types (i.e., 1D-scalar and 3D-point features) and the operation used to compute the final attention weights. We focus on the well-known dot product similarity, used in the SE(3)-Transformer and negative distance similarity, which was first presented in the Invariant Point Attention module of AlphaFold2. Aside from calculating the similarity between points, these architectures also differ in their

calculation of attention scores - AlphaFold2 uses the same shared attention weights for scalar and point features, whereas the SE(3)-Transformer computes attention weights for each type. This is outlined more formally in Table S8. Following the conventions of ((15, 19)), we use a superscript ℓ to denote type-ℓ features of dimension $2\ell + 1$.

	Similarity
Dot Product	$\sigma_{ij}^{\ell} = \left(q_i^{\ell}\right)^{\top} k_{ij}^{\ell} + b_{ij}^{\ell}$
Distance	$\sigma_{ij}^{\ell} = \begin{cases} \left(q_i^{\ell}\right)^{\top} k_{ij}^{\ell} & \ell = 0 \\ \left\| T_{ii}\left(q_i^{\ell}\right) - T_{ij}\left(k_{ij}^{\ell}\right) \right\| & \ell > 0 \end{cases}$
	Attention
Per-Type	$f_{out,i}^{\ell} = \sum_{j \in \mathcal{N}(i)} w^{\ell} \alpha_{ij}^{\ell} v_{ij}$
Shared	$f_{out,i}^{\ell} = \sum_{j \in \mathcal{N}(i)} \left(\sum_{\ell'} w^{\ell'} \alpha_{ij}^{\ell'} \right) v_{ij}^{\ell}$

Table S8. SE(3)-equivariant Similarity and attention types. Here, we use a superscript ℓ to denote type-ℓ features of dimension $2\ell + 1$ (i.e. $\ell = 0$ for scalar features and $\ell = 1$ for point features). For distance similarity, we use T_{xy} to denote some transformation mapping points into so-called "local frames" of the respective node used to ensure the equivariance of the operation.

We trained four TFN-Transformer models differing only by attention and similarity type. Hyperparameters were chosen to match those in Table S1. The average RMSD and dihedral accuracy results of the four attention variants are shown in Table S9.

		CASP13				CASP14			
		RMSD(Å)↓			χ_{1-4} Acc↑	RMSD(Å)↓			χ_{1-4} Acc↑
Similarity	**Attention**	All	Core	Sfc.	All	All	Core	Sfc.	All
Distance	per-type	0.811	0.555	1.074	55.1%	0.972	0.669	1.197	45.2%
	shared	0.764	0.512	0.943	57.3%	0.909	0.590	1.150	46.9%
Dot-Prod.	per-type	0.736	0.496	0.907	58.1%	0.878	0.572	1.113	47.4%
	shared	**0.710**	**0.487**	**0.895**	**58.4%**	**0.865**	**0.564**	**1.102**	**48.0%**

Table S9. Comparison of SE(3)-equivariant attention and similarity operations on performance for CASP13 and CASP14 targets. Performance is measured using average residue RMSD (Å) and χ_{1-4} prediction accuracy.

The results in Table S9 show that shared attention weights produce better results for both similarity types. Overall, dot-product-based similarity outperforms distance-based similarity, even when per-type attention is used. As pointed out by Fuchs et al. (15), this may be due to the fact that each basis kernel in a TFN is completely constrained in the angular direction. By using dot-product based similarity, the angular profile of the basis kernels is modulated by the attention weights.

We also experimented with different architectural variants. First, we used a linear projection rather than a TFN to compute keys at each attention head. Next, we augmented the input to the TFN radial kernel by concatenating the pairwise distances between hidden coordinates. Third, we tried removing the attention calculation between points and instead used only scalar features to compute attention weights. For each variant, shared attention weights and dot product similarity was used. As shown in Table S10, using a linear projection for attention keys has the largest impact on RMSD score - surprisingly much more than removing point-based attention all together. This suggests that TFN neighbor convolutions are an important component of the architecture. The results also show that RMSD scores are improved when pairwise distances between hidden coordinate features are concatenated to the input of the TFN radial kernel. In all ablations, we see little variation in RMSD scores for surface residues.

(A)	RMSD (Å)↓			(B)	RMSD (Å)↓		
TFN-Transformer	All	Core	Sfc.	**TFN+Local**	All	Core	Sfc.
Baseline	0.710	0.487	0.895	$\theta = 8$	0.695	0.480	0.895
+Pairwise Distance	**0.698**	**0.475**	**0.892**	$\theta = 12$	0.679	0.438	0.888
+ No coord. attention	0.743	0.539	0.962	$\theta = 16$	0.669	0.414	0.881
+ Linear Keys	0.809	0.565	1.001	$\theta = \infty$	**0.665**	**0.409**	**0.876**

Table S10. Architecture and hyperparameter study. The tables show the average side-chain RMSD on the CASP13 targets. Table (a) shows the effect of TFN architectural features. *Baseline* denotes our TFN-Transformer with shared dot-product attention and pair bias. The other three rows show results after changing some components. Table (b) shows the effect of neighbor distance threshold on RMSD with $\theta = 8,\ 12,\ 16$ and ∞. Each model in (b) uses our locality-aware graph transformer with distance cutoff θ and TFN-Transformer with shared dot-product similarity, pair bias, and pairwise distance features. All other hyperparameters were held constant.

We also considered the effect of the neighbor distance threshold on performance. This threshold is used as the maximum valid edge length for triangle updates and to determine residue adjacency in the locality-aware graph transformer. It is also used to define the neighborhood of scalar and point features in the TFN-Transformer. We tried three distance thresholds, 8Å, 12Å, and 16Å The results are shown in Table S10.

Average RMSD clearly decreases with increasing neighbor distance. The marginal improvement diminishes with increasing radius, and the bulk of the improvement comes from residues in the protein core.

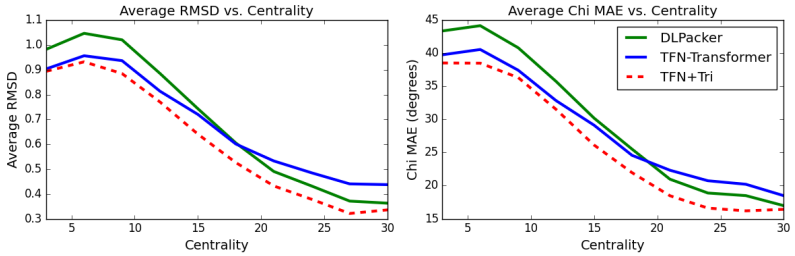

Fig. S7. Average RMSD (Å) and χ_{1-4}-MAE$^\circ$ against centrality. The y-axis shows average RMSD (**left**) and average χ-MAE (**right**) for methods DLPacker, TFN-Transformer (TFN), and AttnPacker against residue centrality (x-axis). Values were computed using all targets in both CASP13 and CASP14 data sets. Values were disregarded if the number of residues with the corresponding centrality was less than 30. We remark that RMSD was computed prior to performing post-processing for our methods.

Not surprisingly, RMSD and chi angle prediction errors decrease rapidly as centrality increases. It is also clear that the marginal improvement garnered from including our graph transformer module increases with centrality. For both RMSD and MAE, the performance gap between the TFN-Transformer and our complete model increases with centrality, suggesting that triangle updates are important for accurately determining side-chain conformations in protein cores. The opposite is true when comparing TFN+Tri with DLPacker, where the gap decreases as centrality increases. This is not surprising, as DLPacker iteratively constructs each residue's side-chain using only atoms in the residue's immediate microenvironment as input. This choice of input features implies that features for protein surface residues are more sparse than those for core residues.

J. Extended Results for Sequence Design. We provide results for AttnPacker on the fixed-backbone design task. We exclude side-chain statistics from this analysis as the amino acid type predicted by AttnPacker+Design may be different from that of the native. An example of AttnPacker designed sequence and side-chains is given in Figure S14

Protein MPNN is trained on a mix of monomeric and multimeric proteins gathered from the PDB as of August 2021. Targets are clustered by sequence identity, using a 30% cutoff. This results in roughly 25k clusters reserved for training. Different from our method, an example is randomly re-sampled from each cluster at every epoch.

We report (i) native sequence recovery (NSR) rate and (ii) model perplexity. The rationale is that NSR assesses how closely a designed sequence matches the native sequence of an input backbone structure, and perplexity tests the ability of the model to assign a high likelihood to the native sequence. Following Jing et al. (11), NSR and perplexity are reported as the median (over all structures) of the average of each metric for each target. Alternatively, as is the case with side-chain RMSD, the average over all residues in a data set can be reported, but the former definition is more typical in the fixed-backbone design literature.

	Random-Mask			**Linear-Mask**		
	(%) Masked Residues			**Length**		
	10%	25%	50%	5	10	20
	Native Sequence Recovery (NSR) ↑					
Mean	50.8%	51.3%	48.7%	50.9%	53.3%	52.4%
Med.	52.7%	52.7%	49.8%	60.0%	50.0%	55.0%
Std.	0.14	0.10	0.08	0.20	0.17	0.13
	Perplexity ↓					
Mean	4.65	4.71	5.01	5.15	4.78	4.64
Med.	4.21	4.12	4.30	3.89	4.07	4.05
Std.	1.97	1.70	1.59	3.77	2.84	2.33

Table S11. Inverse folding results with varying levels of sequence information, for CASP13 targets. Native sequence recovery (top) and perplexity (bottom) values are extracted only for residues where sequence information is withheld. Five designs were produced for each target in the CASP13 test set and each setting. Results for each setting were obtained by averaging over all designs.

In Table S11, we show results for AttnPacker+Design when various levels of partial sequence information are provided. We consider masking randomly selected subsets of residues (*Random-Mask*), and masking a contiguous segment (*Linear-Mask*) of 5, 10, or 20 residues were also chosen at random. In the Random-Mask setting, we choose the number of residues in each subset as a fixed percentage of the protein's length, using values 10%,25% and 50%.

Results in Table S11 indicate that median sequence recovery tends to decrease as the percentage of masked residues increases. A similar trend is seen for mean and median perplexity, which positively correlates with the percentage of masked residues. For instance, when only 10% of the input sequence is masked, AttnPacker achieves a median recovery of 52.7%, dropping to 48.6% when the entire sequence is withheld. These results suggest that AttnPacker is able to incorporate sequence context in its designs effectively.

Results on linear masking are somewhat surprising. Median perplexity is lowest for the smallest mask length of five residues and roughly the same for longer lengths. Standard deviations in perplexity and recovery also decrease as crop length increases. Like side-chain RMSD, sequence recovery is also highly correlated with residue degree centrality (3). Shorter crop lengths may have a higher chance of having all or no residues in the protein core, whereas longer crop lengths are more likely to have residues in both regions. Future work may analyze performance conditioned on the average residue centrality of masked residues.

	NSR ↑		Perplexity ↓	
Method	Mean	Med.	Mean	Med
AttnPacker+Design	47.8%	48.6%	**5.39**	5.09
ProteinMPNN-0.002	**48.5%**	**51.3%**	5.64	**4.85**
ProteinMPNN-0.01	44.8%	46.7%	6.41	5.53
ProteinMPNN-0.02	42.3%	44.3%	6.94	6.22
GVP-GNN + (CATH4.2)	42.2%	44.0%	5.77	5.12
GVP-GNN + (BC40)	41.9%	43.3%	5.82	5.04
RosettaDesign	33.8%	34.2%	-	-

Table S12. Inverse Folding Results on CASP13 targets. We suffix ProteinMPNN with the training noise level, e.g., ProteinMPNN-0.01 corresponds to ProteinMPNN trained with 0.1Å noise on coordinates. Methods GVP + BC40 and GVP + CATH4.2 use the same hyperparameters as reported by Jing et al.(11), but differ on the training set used - CATH4.2, and BC40, respectively.

In Table S12, we compare AttnPacker+Design to ProteinMPNN, GVP-GNN, and RosettaDesign. In comparing with GVP-GNN we re-trained the model on the same training set as AttnPacker+Design to provide an unbiased comparison. Pre-trained models were used for ProteinMPNN and we remark that there may be some train/test overlap for this model. The data collection process is described in detail in Section A.2.

As shown in Table S12, AttnPacker+Design achieves similar performance to ProteinMPNN while outperforming GVP-GNN trained on the same dataset. The GVP-GNN model is relatively shallow compared to AttnPacker, using only three encoder-decoder layers. GVP-GNN also uses much smaller hidden dimensions: 100 for scalar features, 16 for coordinate features, and 32 for pair features. We attempted to train wider and deeper variants of GVP-GNN, but were unable to significantly improve performance, likely because of the well-known over-smoothing issue(20) with graph neural networks.

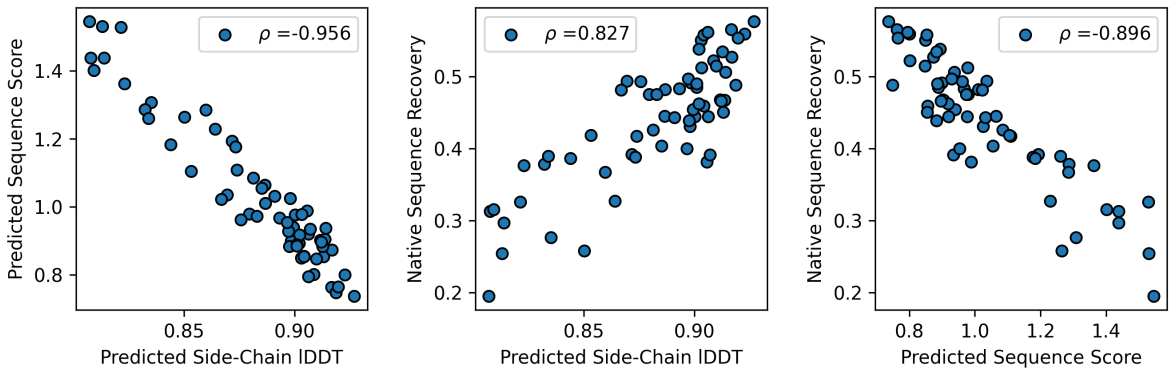

Fig. S8. Model Confidence Calibration. Scatter plots and correlation between predicted side-chain lDDT, sequence score, and native sequence recovery for CASP13 targets. AttnPacker+Design was run five times for each target with no partial sequence information provided. Each metric was averaged over each target to produce the plots.

J.1. In-Silico Analysis. To understand how well our generated sequences encode native backbone structure, we used ESMFold(21) to predict structures from sequence designs for CASP13 and CASP14 targets. We assess prediction quality using self-consistency TM-score (scTM) and ESMFold predicted lDDT (plDDT). scTM indicates how well generated sequences encode structure by measuring the TM-score(22) between predicted and ground-truth backbones. Predicted lDDT indicates how confidently a sequence folds to the predicted structure. Each metric ranges from 0 to 1, with 1 being the most favorable score.

In Figure S9, we show scTM and plDDT distributions for AttnPacker+Design and ProteinMPNN using native backbones of CASP13 and CASP14 targets. For this assessment, we follow the methodology in ProteinMPNN, restricting to targets with at most 350 residues, and generate eight sequences per target backbone. Sequences for AttnPacker were sampled using Gibbs sampling as described in (23), with a total of 20 transitions per design, using a 15% re-sampling rate and linear temperature decay from 0.5 to 0.1. A script for running this sampling procedure can be found on our GitHub repo.

Sequences generated from CASP13 and CASP14 native backbones are predicted confidently and accurately by both methods, though ProteinMPNN predicts a higher proportion of sequences with scTM and plDDT at the upper tails of the distribution. As mentioned in our discussion, this may be attributed to perturbing backbone coordinates with Gaussian noise or re-sampling clusters at inference time. We also remark that our training set is filtered against CASP13 and CASP14 target backbones. It is possible that the training data used for ProteinMPNN has some overlap with these targets.

The authors of ProteinMPNN note that AlphaFold success rate tends to increase monotonically in the number of recycling iterations and conjecture that single sequence structure prediction models may yield even higher success rates ((24), Supplementary Material, "Further in silico analysis"). Our results in Figure S9 offer some empirical evidence of this hypothesis. Independent assessment on CASP13 targets shows similar native sequence recovery rates as reported on the PDB test set (52% for monomers).

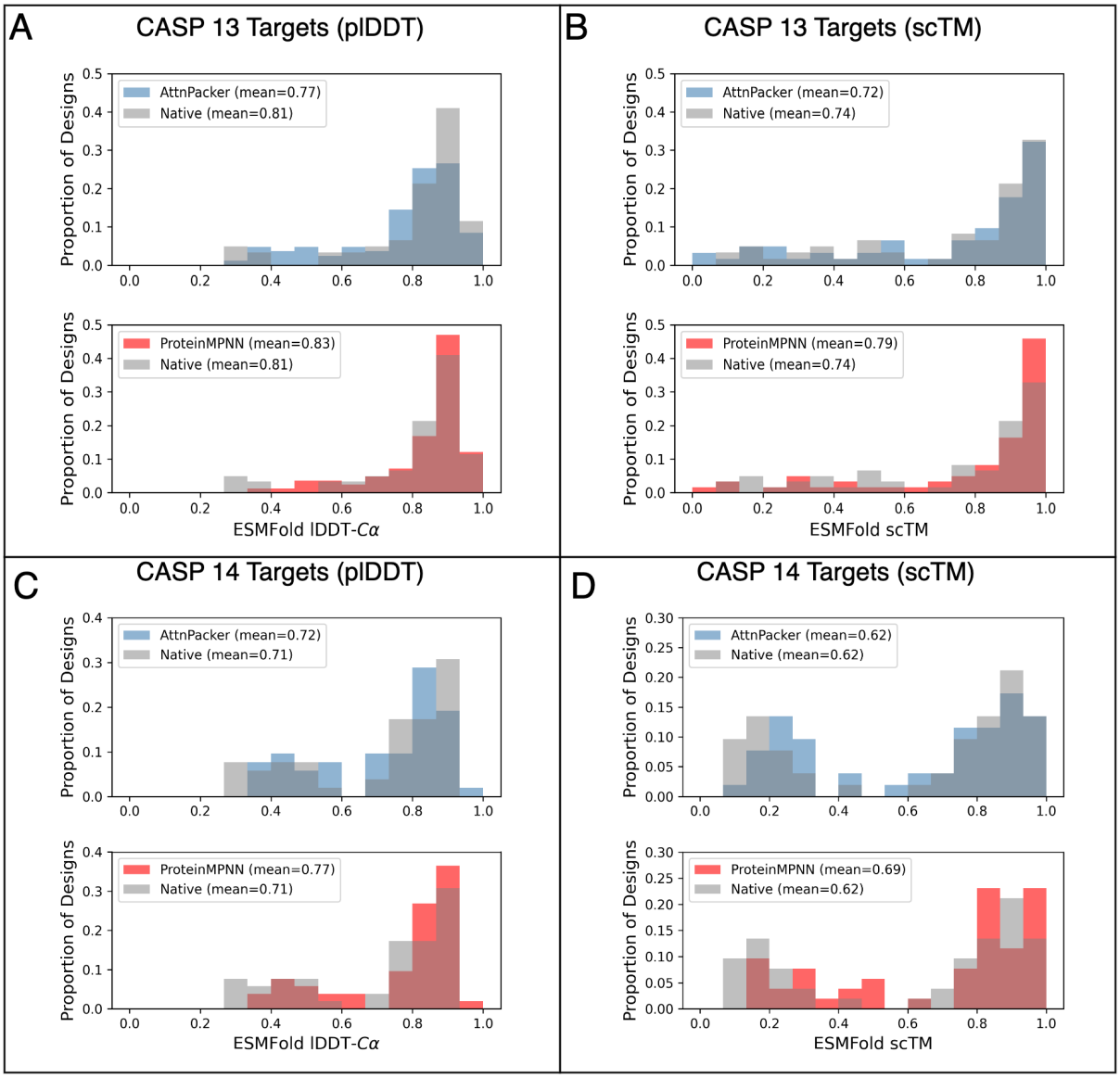

Fig. S9. ScTM and plDDT distributions for CASP13 and CASP14 targets. Distributions for AttnPacker, ProteinMPNN, and native sequences are shown in blue, gray, and red, respectively.

K. Extended Results for Side-Chain Packing.

Method	MAE° ↓				Acc(%)↑		Method	MAE° ↓			Acc(%)↑	
	χ_1	χ_2	χ_3	χ_4	χ_{1-2}	χ_{1-4}		χ_1	χ_2	χ_3	χ_{1-2}	χ_{1-4}
	ARG							**GLN**				
Ros. Pack.	35.87	34.75	64.74	62.44	43.5%	**16.9%**	Ros. Pack.	43.50	47.92	32.81	32.6%	17.4%
DLPack.	32.70	31.75	66.52	70.84	40.1%	10.0%	DLPack.	42.95	47.91	37.34	30.9%	14.3%
AttnPack.	30.28	30.40	**61.46**	**60.27**	42.9%	9.8%	AttnPack.	40.39	45.24	31.81	34.0%	18.6%
+Design	**29.29**	**29.58**	62.07	62.26	**44.2%**	9.6%	+Design	**37.86**	**43.29**	**30.76**	**35.8%**	**19.1%**
	LYS							**GLU**				
Ros. Pack.	39.86	38.01	48.74	52.25	38.2%	**12.6%**	Ros. Pack.	36.19	43.66	59.49	39.7%	20.4%
DLPack.	35.25	37.17	54.28	74.22	37.5%	6.5%	DLPack.	32.38	43.39	57.82	**44.8%**	18.9%
AttnPack.	30.44	**34.11**	49.55	**46.72**	**42.3%**	10.1%	AttnPack.	29.16	40.39	**49.79**	43.1%	**21.3%**
+Design	**30.27**	34.45	50.50	50.12	41.0%	9.0%	+Design	**28.53**	**38.60**	50.51	43.0%	**21.3%**

Table S13. Dihedral MAE results on CASP14 targets for charged and polar Amino Acids with high degrees of freedom.

Method	RMSD (Å)↓			χ−MAE°↓				χ-Acc(%)↑			Clash ↓		
	All	Sfc.	Core	χ_1	χ_2	χ_3	χ_4	All	Sfc.	Core	80%	90%	100%
SCWRL	0.947	1.174	0.554	27.51	29.70	49.20	59.74	56.4%	47.2%	73.1%	2.5	10.8	60.2
FASPR	0.930	1.154	0.557	27.17	29.38	50.14	59.75	56.4%	47.9%	71.6%	3.2	12.1	60.1
RosettaPacker	0.886	1.126	0.480	25.97	28.87	47.36	54.82	58.7%	47.9%	77.3%	2.5	6.1	41.3
DL-Packer	0.783	0.989	0.446	22.14	27.00	50.38	70.18	58.9%	49.4%	75.5%	0.9	4.1	33.8*
AttnPacker	0.672	0.856	0.375	18.32	24.24	46.28	57.97	62.7%	53.9%	77.7%	0.1*	1.2*	28.1*
+Design	0.669	0.843	0.382	17.86	24.13	45.75	57.57	61.7%	52.7%	77.1%	0.1*	0.9*	28.8*

Table S14. Average RMSD (Å), MAE, and clash results on the CASP13-FM and CASP14-FM targets. To better understand AttnPacker's ability to generalize to new folds, we evaluated each method on CASP13 and CASP14 free modeling (FM) targets. These datasets consist of proteins with previously unseen folds and hard analogous fold-based models (see Table S16 for a complete list). Results are divided by residue degree centrality (All, Core, and Surface). A total of 6866 residues were compared for this table, with 3649 surface residues and 1622 core residues. The number of clashes in native structures at 80%, 90%, and 100% thresholds is 0.1, 3.8, and 38.7

	CASP13						CASP14					
	#Res	SCWRL	FASPR	Ros-P	DL-P	Ours	#Res	SCWRL	FASPR	Ros-P	DL-P	Ours
ARG	1107	2.216	2.192	2.063	1.916	**1.679**	841	2.292	2.313	2.167	2.002	**1.825**
ASN	1089	0.923	0.894	0.883	0.768	**0.663**	1338	1.062	1.031	0.976	0.923	**0.806**
ASP	1212	0.888	0.906	0.872	0.725	**0.610**	1223	1.020	1.058	1.005	0.936	**0.841**
CYS	238	0.533	0.549	0.447	0.446	**0.341**	186	0.526	0.553	**0.304**	0.473	0.371
GLN	844	1.489	1.478	1.331	1.241	**1.048**	741	1.563	1.554	1.440	1.325	**1.199**
GLU	1264	1.469	1.466	1.419	1.316	**1.137**	1277	1.611	1.596	1.554	1.497	**1.395**
HIS	468	1.073	0.913	0.936	0.754	**0.681**	378	1.118	1.057	0.947	0.893	**0.815**
ILE	1315	0.577	0.577	0.545	0.519	**0.452**	1377	0.729	0.770	0.758	0.701	**0.616**
LEU	2114	0.604	0.592	0.566	0.518	**0.432**	1823	0.753	0.712	0.719	0.650	**0.573**
LYS	1161	1.751	1.732	1.647	1.524	**1.364**	1353	1.848	1.836	1.784	1.660	**1.462**
MET	511	1.324	1.228	1.167	1.105	**0.956**	401	1.527	1.341	1.357	1.308	**1.139**
PHE	949	0.825	0.731	0.742	0.564	**0.450**	933	0.905	0.885	0.811	0.731	**0.559**
PRO	1017	0.248	0.236	0.228	0.188	**0.178**	792	0.263	0.252	0.257	**0.218**	0.220
SER	1558	0.720	0.721	0.698	0.580	**0.510**	1420	0.866	0.844	0.815	0.736	**0.652**
THR	1401	0.514	0.506	0.490	0.448	**0.391**	1214	0.707	0.685	0.687	0.643	**0.578**
TRP	385	1.332	1.128	1.001	0.837	**0.698**	229	1.466	1.319	1.416	1.083	**0.996**
TYR	897	1.034	0.965	0.980	0.678	**0.603**	882	1.063	1.046	0.972	0.800	**0.618**
VAL	1588	0.328	0.328	0.316	0.291	**0.251**	1285	0.431	0.444	0.428	0.394	**0.352**
AVG.		0.934	0.910	0.872	0.772	**0.669**		1.062	1.048	1.006	0.929	**0.823**

Table S15. Average Per-Residue RMSD for CASP13 and CASP14 Targets by method (column) and residue type (row). We add a row AVG. to show the average RMSD over all residue types. Here, we shorten names for DLPacker (DL-P), RosettaPacker (Ros-P), and AttnPacker (Ours)

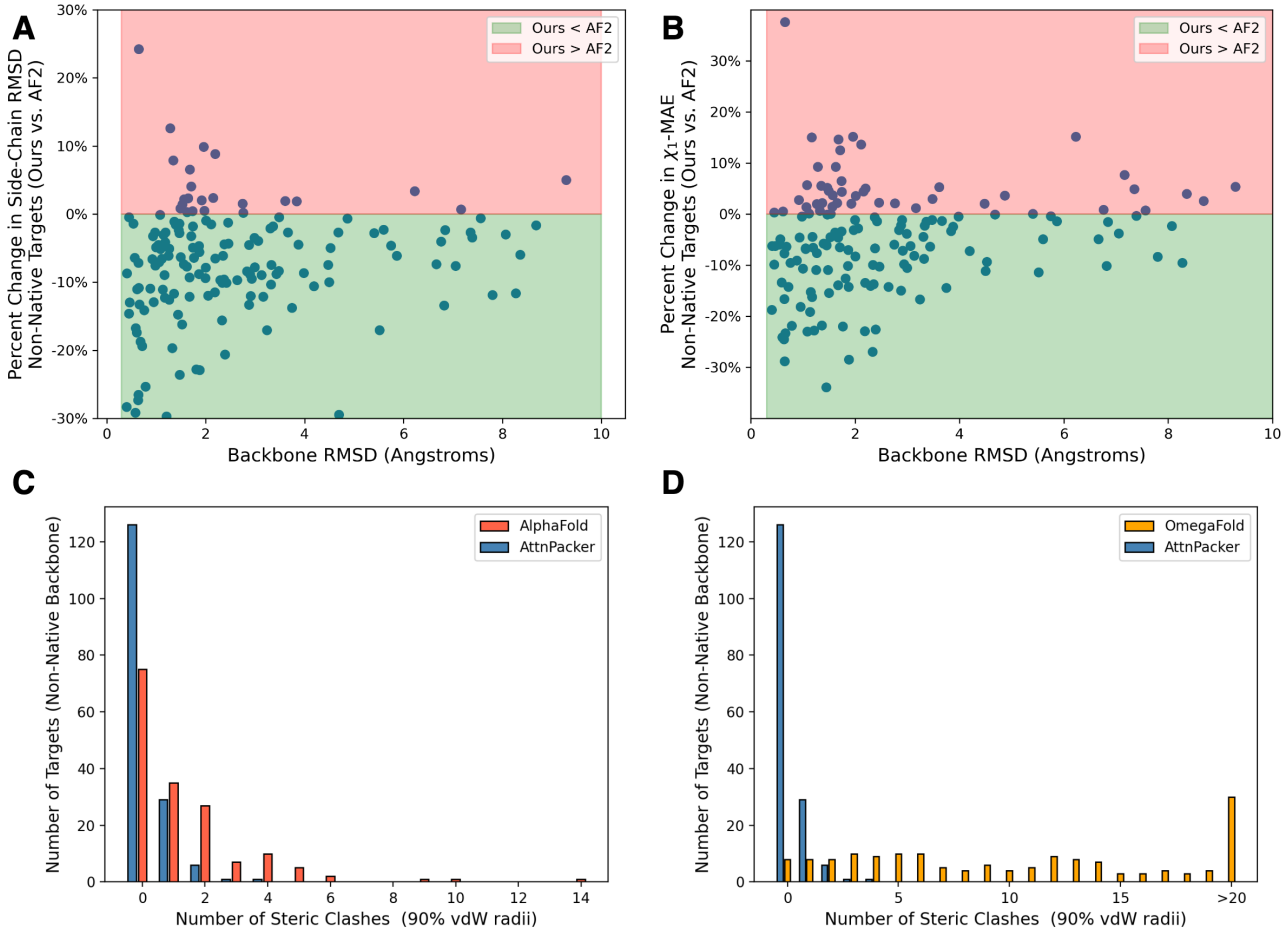

Fig. S10. (**A**) Per-target scatter plot comparing AttnPacker and AlphaFold2 (AF2) on non-native backbones for CASP13 and CASP14 targets. The x-axis shows backbone $C\alpha$ RMSD between AlphaFold2 predicted backbones and native structures. The y-axis shows the percentage difference in average side-chain RMSD using our method's prediction as the initial value. Each point represents the percentage difference in side-chain RMSD for a single protein packed by AF2 and AttnPacker. (**B**) Analogous plot using χ_1 MAE (degrees) as criteria. (**C**,**D**) show frequency plots of per-target steric clashes using 90% van der Waals radius. The x-axis shows the number of steric clashes, and the y-axis shows the number of (non-native backbone) targets having this many clashes. Our method is compared to AlphaFold2 in **C** and OmegaFold in **D**. For All plots, we restrict to backbone RMSD <10Å for clarity, as only a small fraction of targets are predicted with larger RMSD.

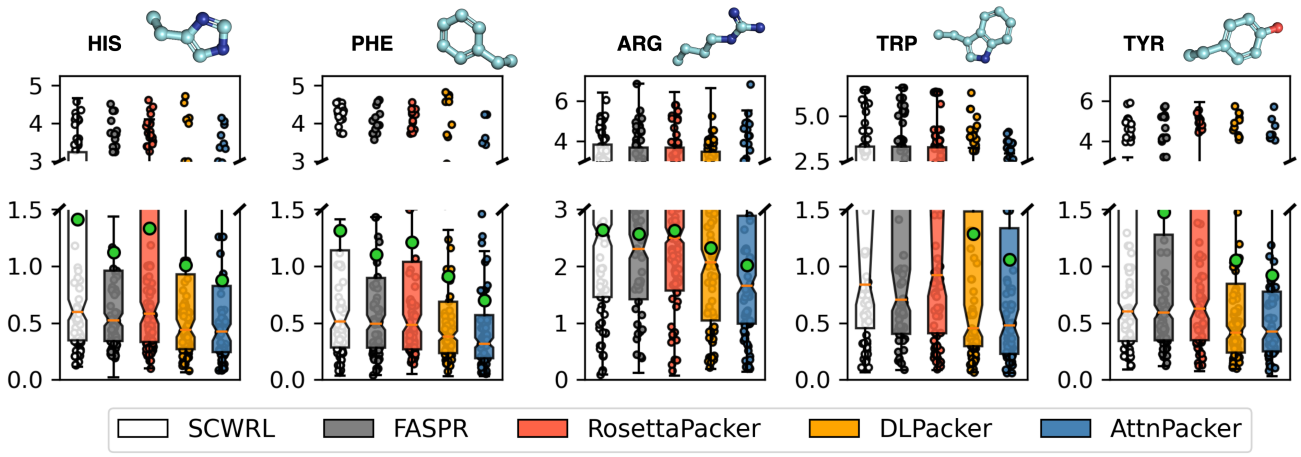

Fig. S11. RMSD box plots for Bulky Amino Acids on the CASP13 targets. Average RMSD (Å) for each residue type, restricted to the protein's surface (y-axis) of each method (x-axis) on His, Phe, Arg, Trp, and Tyr (from left to right). Each box extends from the lower to upper quartile values of the data, with a line at the median. The 95% confidence intervals around the median are shown with a notch, and the mean is shown with a green triangle. RMSD values are taken over the entire CASP13 data set for each residue type.

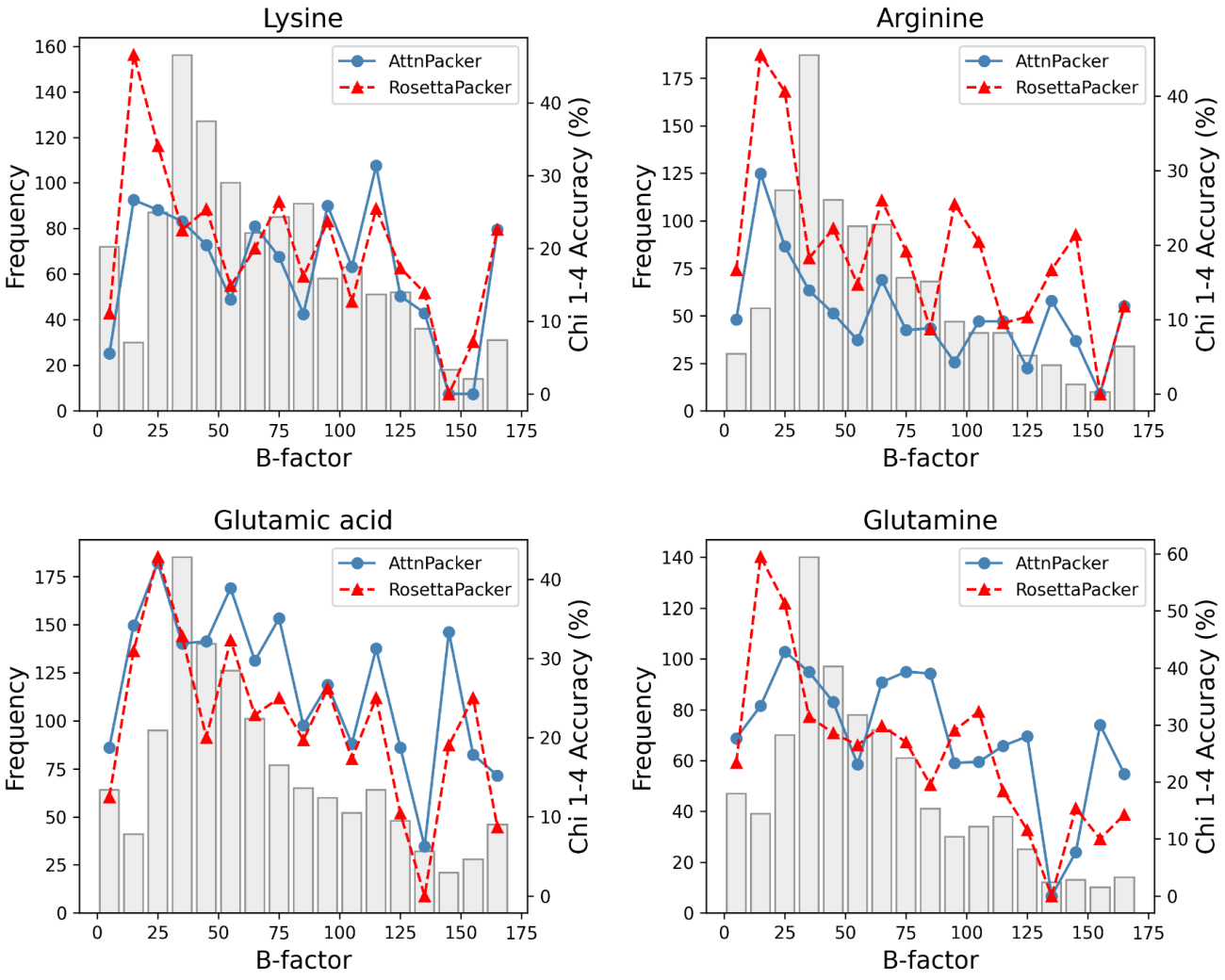

Fig. S12. χ_{1-4} **accuracy conditioned on average side-chain atom B-factor for CASP13 targets**. Dihedral accuracy for AttnPacker (blue) and RosettaPacker (red) are shown for four amino acid types. Gray bars indicate the frequency of the corresponding residue type and b-factor bin, as indicated by a secondary y-axis shown on the left of each plot. For each residue, we compute the B-factor by averaging over the values of each side-chain atom type.

L. Examples.

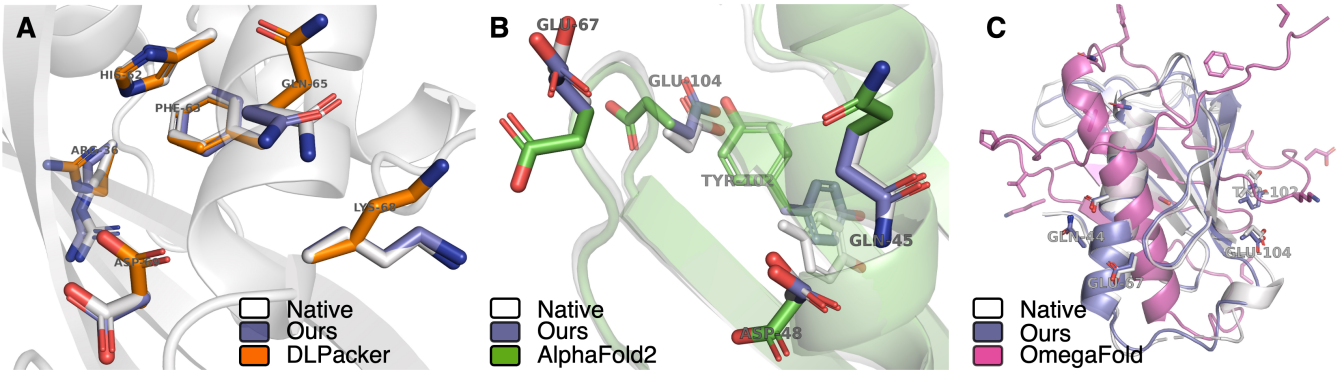

Fig. S13. Example backbone packing where AttnPacker yields correct results. (**A**) The backbone of CASP13 target T0968s1 is shown in cartoon representation, and selected side chains are shown using stick representation. Amino acid names and sequence positions are shown for select residues. AttnPacker achieves an average RMSD 0.36Å, and DLPacker achieves 0.62Å. **B** Native and AlphaFold2 predicted backbones for CASP13 target T0980s1 are shown in cartoon representation, and selected side chains are shown using stick representation. Amino acid names and sequence positions are shown for select residues. Here, AttnPacker achieves an average RMSD 0.83Å, average RMSD, and AlphaFold2 achieves 1.08Å. (**C**), shows OmegaFold's predicted backbone for the same target.

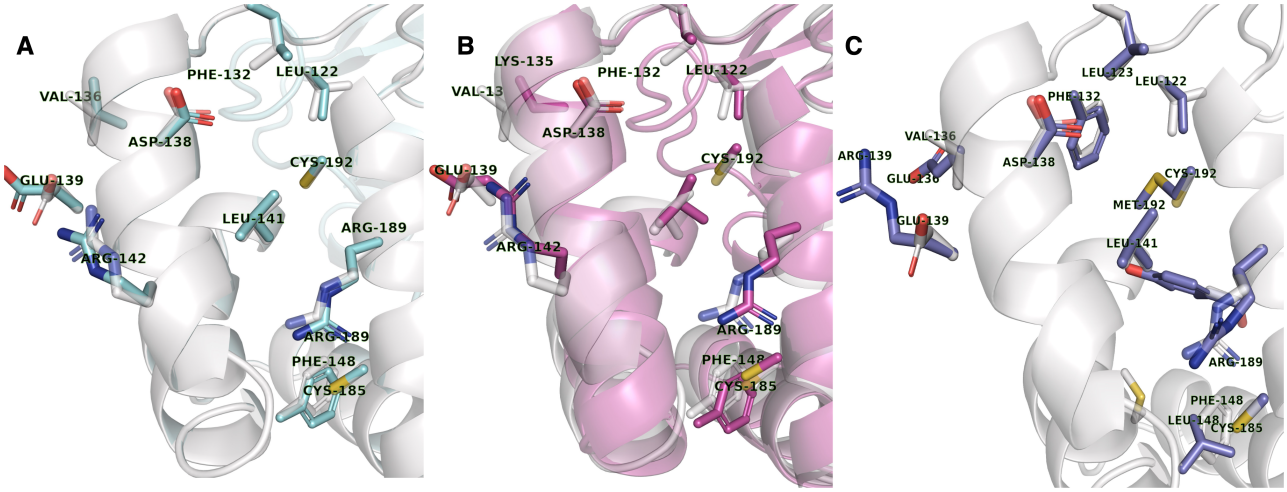

Fig. S14. Example Side-Chain Packings and co-design of side-chains and sequences. Native backbones are shown in what using cartoon format. Native side-chains are shown in white using sticks. (A) Illustration of correct and incorrect side-chain predictions (cyan)from AttnPacker for CASP target T0951. (B) AttnPacker prediction on AF2 predicted backbone for CASP target T0951(C) AttnPacker designed sequence and side-chains for the same target.

M. PDB Lists.

Data Set	Targets
CASP13	T0949, T0950, T0951, T0953s1, T0953s2, T0954, T0955, T0957s1, T0957s2, T0958, T0959, T0960, T0961, T0962, T0963, T0964, T0965, T0966, T0967, T0968s1, T0968s2, T0969, T0970, T0971, T0973, T0974s1, T0974s2, T0975, T0976, T0977, T0978, T0979, T0980s1, T0980s2, T0981, T0982, T0983, T0984, T0985, T0986s1, T0986s2, T0987, T0988, T0989, T0990, T0991, T0992, T0993s1, T0993s2, T0994, T0995, T0996, T0997, T0998, T1000, T1001, T1002, T1003, T1004, T1005, T1006, T1008, T1009, T1010, T1011, T1013, T1014, T1015s1, T1015s2, T1016, T1016_A, T1017s1, T1017s2, T1018, T1019s1, T1019s2, T1020, T1021s1, T1021s2, T1021s3, T1022s1, T1022s2
CASP14	T1024, T1025, T1026, T1027, T1028, T1030, T1031, T1032, T1033, T1034, T1035, T1036s1, T1037, T1038, T1039, T1040, T1042, T1043, T1045s1, T1045s2, T1046s1, T1046s2, T1047s1, T1047s2, T1048, T1049, T1052, T1053, T1054, T1055, T1056, T1057, T1058, T1060s2, T1060s3, T1062, T1064, T1065s1, T1065s2, T1067, T1068, T1070, T1072s1, T1073, T1074, T1078, T1079, T1080, T1082, T1083, T1084, T1087, T1088, T1089, T1090, T1091, T1092, T1093, T1094, T1095, T1096, T1098, T1099, T1100
CASP13-FM	T0950, T0953s1, T0953s2, T0957s1, T0957s2, T0960, T0963, T0968s1, T0968s2, T0969, T0975, T0980s1, T0981, T0986s2, T0987, T0989, T0990, T0991, T0998, T1000, T1001, T1010, T1015s1, T1017s2, T1021s3, T1022s1
CASP14-FM	T1027, T1031, T1033, T1037, T1038, T1039, T1040, T1042, T1043, T1047s1, T1049, T1064, T1070, T1074, T1090, T1093, T1094, T1096

Table S16. List of targets in each test data set.

References

1. Mirdita M, et al. (2022) Colabfold: making protein folding accessible to all. *Nature Methods* 19(6):679–682.
2. Steinegger M, Söding J (2017) Mmseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nature Biotechnology* 35(11):1026–1028.
3. Ingraham J, Garg V, Barzilay R, Jaakkola T (2019) Generative models for graph-based protein design in *Advances in Neural Information Processing Systems*, eds. Wallach H, et al. (Curran Associates, Inc.), Vol. 32.
4. Kingma DP, Ba J (2015) Adam: A method for stochastic optimization in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, eds. Bengio Y, LeCun Y.
5. Bachlechner T, Majumder BP, Mao HH, Cottrell GW, McAuley J (2020) Rezero is all you need: Fast convergence at large depth.
6. Pascanu R, Mikolov T, Bengio Y (2013) On the difficulty of training recurrent neural networks in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13. (JMLR.org), pp. III–1310–III–1318.
7. Hiranuma N, et al. (2020) Improved protein structure refinement guided by deep learning based accuracy estimation. *bioRxiv*.
8. Yang J, et al. (2020) Improved protein structure prediction using predicted interresidue orientations. *Proceedings of the National Academy of Sciences* 117(3):1496–1503.
9. Ying C, et al. (2021) Do transformers really perform bad for graph representation? *CoRR* abs/2106.05234.
10. Jing X, Xu J (2021) Fast and effective protein model refinement using deep graph neural networks. *Nat. Comput Sci* 1:462–469.
11. Jing B, Eismann S, Suriana P, Townshend RJL, Dror R (2020) Learning from protein structure with geometric vector perceptrons.
12. Jumper J, et al. (2021) Highly accurate protein structure prediction with alphafold. *Nature* 596(7873):583–589.
13. Ahdritz G, et al. (2022) Openfold: Retraining alphafold2 yields new insights into its learning mechanisms and capacity for generalization. *bioRxiv*.
14. Byrd RH, Lu P, Nocedal J, Zhu C (1995) A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing* 16(5):1190–1208.
15. Fuchs F, Worrall D, Fischer V, Welling M (2020) Se(3)-transformers: 3d roto-translation equivariant attention networks in *Advances in Neural Information Processing Systems*, eds. Larochelle H, Ranzato M, Hadsell R, Balcan MF, Lin H. (Curran Associates, Inc.), Vol. 33, pp. 1970–1981.
16. Deng C, et al. (2021) Vector neurons: A general framework for so(3)-equivariant networks. *ArXiv* abs/2104.12229.
17. Jing B, Eismann S, Suriana P, Townshend RJL, Dror R (2021) Learning from protein structure with geometric vector perceptrons.
18. Ba JL, Kiros JR, Hinton GE (2016) Layer normalization.
19. Thomas N, et al. (2018) Tensor field networks: Rotation- and translation-equivariant neural networks for 3d point clouds. *CoRR* abs/1802.08219.
20. Cai C, Wang Y (2020) A note on over-smoothing for graph neural networks. *CoRR* abs/2006.13318.
21. Lin Z, et al. (2022) Language models of protein sequences at the scale of evolution enable accurate structure prediction. *bioRxiv*.
22. Xu J, Zhang Y (2010) How significant is a protein structure similarity with TM-score = 0.5? *Bioinformatics* 26(7):889–895.
23. Johnson SR, Monaco S, Massie K, Syed Z (2021) Generating novel protein sequences using gibbs sampling of masked language models. *bioRxiv*.
24. Dauparas J, et al. (2022) Robust deep learning based protein sequence design using proteinmpnn. *Science* 378(6615):49–56.