THE UNIVERSITY OF CHICAGO

ACCELERATING PROTEIN DESIGN WITH DEEP LEARNING

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY

MATTHEW TIMOTHY MCPARTLON

CHICAGO, ILLINOIS

AUGUST 2023

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to Jinbo Xu, for his mentorship and for making this work possible. Your ability to identify the crux of a problem is exceptional. You pushed me to develop a skill set that is highly transferable and somehow both broad and deep. Our discussions have forever changed the way I approach science.

I would like to extend my sincerest thanks to Andy Drucker, who helped start me on my research journey. I am also indebted to Janos Simon for his mentorship at the start of my Ph.D. and continued guidance thereafter. I am deeply grateful to Gerry Brady, for whom I was a teaching assistant at least a dozen times. You have made my experience $\omega(1)$ times more enjoyable than it would have otherwise been.

I am fortunate to have made incredible friends during my Ph.D. To Chris Jones, Goutham Rajendran, Jafar Jafarov, Nathan Mull, Sudarshan Babu, Andrew Eckhart, Lonnie Hatcher, Hy Trong Son, Ben Lai, and Mourad Heddaya, thank you all for making grad school fun.

This journey was made possible by many educators and friends along the way. Thank you to Don Fremont for your encouragement and for constantly promoting the joys of learning. Thank you to Cristian Rojas and Aaron Wood for pushing me to study mathematics and entertaining my tangential questions during office hours. To Aaron Ackbarali, thank you for your friendship and for introducing me to advanced mathematics. Special thanks to Paul Kehle for exposing me to research and allowing me the freedom to become completely enthralled by a problem. I am also thankful to David Eck for his invaluable guidance and friendship. Finally, I would be remiss not to mention Stina Bridgeman, who first exposed me to computer science and has forever shaped how I think about programming and problem-solving.

Last and most importantly, I'd like to thank my family for seven years of support and encouragement. To my parents, Tim and Melissa, thank you for promoting creativity, hard work, and self-reliance. You are both my role models, and none of this would have been possible without you.

ABSTRACT

The human proteome comprises tens of thousands of proteins, each tailored for a specific function by the selective pressures of evolution. The field of protein design seeks to develop proteins with new or enhanced functions at will, ultimately bypassing the evolutionary clock. In this thesis, we introduce general machine-learning methods for accelerating protein design, with a particular focus on modeling protein structure.

First, we propose an approach for fixed-backbone design (Chapter 2), the problem of designing primary sequence and side-chain rotamers for a given backbone conformation. Whereas classic approaches formulate sequence and rotamer design tasks separately, we offer an approach to solve both simultaneously. To realize this, we develop a deep neural network that effectively leverages backbone coordinates. By exploiting backbone geometry, we efficiently represent atomic microenvironments at the coordinate level and ultimately avoid discrete rotamer sampling. This results in more robust designs and accurate quality estimates for downstream tasks.

Next, we introduce a framework for flexible protein-protein docking (Chapter 3), the task of determining the structure of a protein complex given the unbound structures of its constituent chains. Traditional docking methods are limited by their reliance on empirical physics-based scoring functions, inability to accommodate conformational flexibility, and failure to incorporate binding sites. To address these challenges, we propose an end-to-end approach that can model conformational changes and target specific interactions while significantly reducing computational time. As one of the pioneering deep learning methods for this task, we uncover key determinants underlying our success and provide important insights for future research. Finally, we highlight the generality of our approach by extending it to simultaneously dock and co-design the sequence and structure of antibody complementarity-determining regions targeting a specified epitope.

NOTATION AND CONVENTIONS

We typically use lowercase letters for vector-valued variables and capital letters for matrices. We use a calligraphic font for collections of objects such as sets or graphs.

We adopt the convention of using x_i and bold $\boldsymbol{x}_i$ to distinguish between vector and scalar values, i.e. $x_i \in \mathbb{R}$, and $\boldsymbol{x}_i \in \mathbb{R}^{>1}$. When the vector represents 3D coordinates, we add an arrow, i.e., $\vec{\boldsymbol{x}}$.

We use $\{\boldsymbol{x}_i\}$ to distinguish the specific datapoint $\boldsymbol{x}_i$ from the list of datapoints $\{\boldsymbol{x}_i\} = \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ (or list of scalars $\{x_i\}$) indexed by i. For example, we may use $\{\boldsymbol{e}_{ij}\}$ to denote the set of all pair features in a graph neural network.

A protein $\mathcal{P} = (\{s_i\}, \{\vec{\boldsymbol{x}}_i^a : a \in \mathcal{A}\})$ with L residues labeled $1..L$ with atom types $a \in \mathcal{A}$, is represented by its amino acid sequence $\{s_i\} = s_1, \ldots s_L$, and atom coordinates $\{\vec{\boldsymbol{x}}_i^a\} = \vec{\boldsymbol{x}}_1^a, \ldots, \vec{\boldsymbol{x}}_L^a \subset \mathbb{R}^3$. Each element s_i can be any of the 20 naturally occurring amino acid types.

When presenting architectural details in text, algorithms, and diagrams, we capitalize operator names that encapsulate learnable parameters, e.g., we use $\text{Linear}\,(\cdot)$ to represent a linear transformation with a learnable weight matrix W and bias vector $\boldsymbol{b}$. Operators without learnable parameters are written in lowercase, e.g., 'softmax,' 'mean,' etc.

CHAPTER 1

INTRODUCTION

1.1 Outline of Thesis

In Chapter 2, we introduce a fixed backbone protein design method. This problem comprises two major sub-tasks, inverse folding, and side-chain packing, both of which are fundamental problems in protein design and have been extensively studied individually.[1] As such, we often compare with methods designed exclusively for one of these sub-tasks and separate our results accordingly. Code and tutorials for our method are made publicly available at https://github.com/MattMcPartlon/attnpacker.

In Chapter 3, we propose a model and framework for learning flexible protein-protein docking. We highlight the generality of our approach by simultaneously docking and co-designing antibody CDR loops. Code for the main components of our model is made publicly available at https://github.com/MattMcPartlon/protein-docking. All code used to generate results for competing methods is also available on our GitHub page.

In the remainder of this chapter, we give a brief overview of proteins and machine learning. We try to focus on the aspects most relevant to this thesis for each topic. For proteins, we focus primarily on understanding their many representations. To tie this in with machine learning, we emphasize architectures and learning paradigms most amenable to these representations.

1.2 Background: Proteins

A protein is a large, complex molecule comprising one or more linear chains of amino acids. Within each chain, amino acid residues[2] are joined to their neighbors by a covalent "peptide

1. [Cao+10; QZ20; Dau+22; Str+20; Jin+20; Xio+14; JKS21; Jin+22; Kuh19; Hsu+22; HPZ20; DSK09; Mis+21; Ana+22; YZY22; XB06; Che+20; NRB12]

2. Chemically, the term 'residue' refers to something which remains after a part of it is consumed in a chemical reaction.

bond". Peptide bonds form by condensation reactions, in which the amino acids lose a single water molecule (hence the name 'residue'). In analogy to language, amino acids function as the letters in the alphabet, and proteins function as words. Just as words have disparate meanings, even two very small proteins can have drastically different properties, such as thermostability, structure, and function.

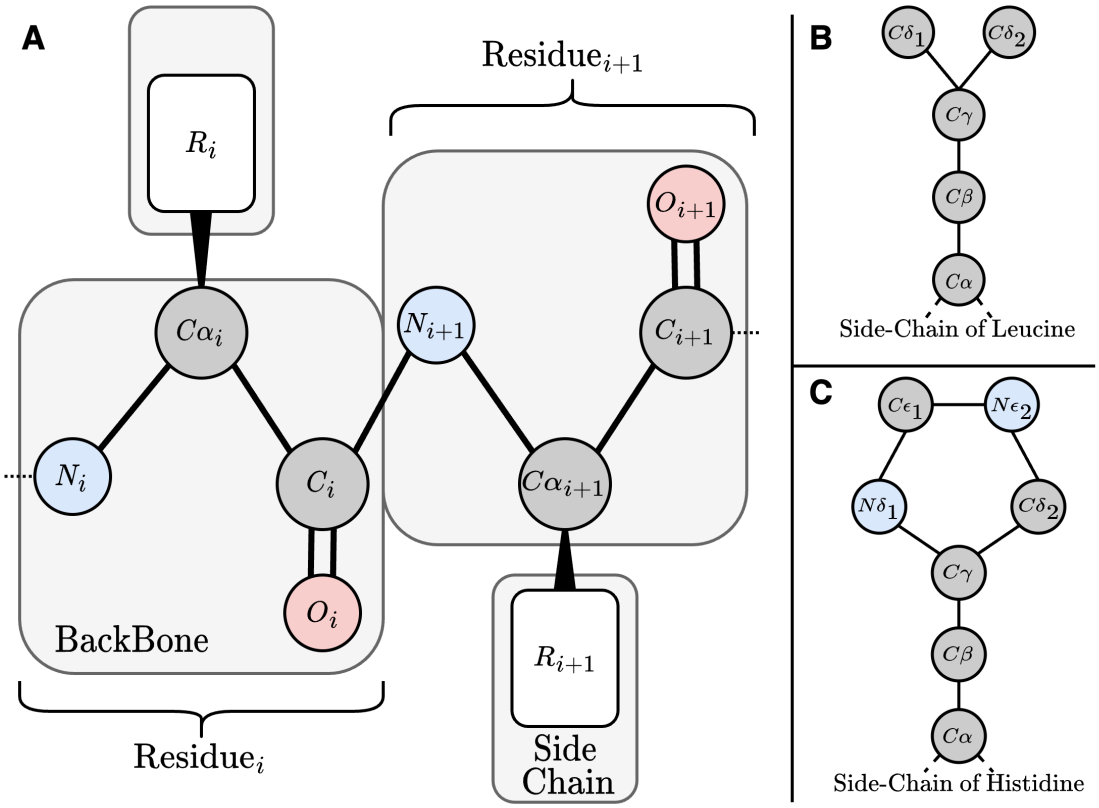

Figure 1.1: **Molecular structure of an amino acid residue in a protein chain.** Hydrogen atoms are omitted. Atoms are colored following the CPK coloring (Corey–Pauling–Koltun) convention. **(A)** Illustration of two bonded amino acids at consecutive positions in a protein chain. Each amino acid residue consists of a *backbone* and side-chain (also called R, or *Rotamer*) group. **(B)** and **(C)** molecular structure of side chains for amino acids Leucine and Histidine. Atoms are labeled following conventions.

At a molecular level, amino acids each share a similar structure. Each has a carboxyl and amino group bonded to the same central carbon ($C\alpha$) atom. The twenty naturally occurring amino acids are distinguished by their side-chain (sometimes called "R-group"), illustrated in Figure 1.1A. Traditionally, non-hydrogen atoms in the R-Group are labeled with their

atomic symbol and a Greek letter. Greek letters are chosen alphabetically based on the number of bonds separating the atom from the backbone alpha-carbon. If the same number of bonds separates two atoms, then a number is added to distinguish them. An example is shown in Figure 1.1B and C.

A protein chain of length L is fully characterized by its amino acid sequence $\{s\} = (s_1, \ldots, s_L) \in [1, \ldots, 20]^L$, and atom coordinates, $\{\vec{\boldsymbol{x}}_i^a\}$, where i is the index of the corresponding amino acid, and a is the atom type. Some examples of proteins include antibodies, which protect the body from foreign compounds, and enzymes, which facilitate almost all chemical reactions in cells.

Protein Structure

Protein Structure is broken down into four classes: primary, secondary, tertiary, and quaternary (see Figure 1.2). The primary structure corresponds to the linear order of a protein's amino acid sequence. The secondary structure describes the local three-dimensional geometry. The tertiary structure defines a single protein chain's global shape, or *fold*. Last, the quaternary structure corresponds to the arrangement of multiple interacting chains.

The nature of covalent peptide bonds in the protein backbone significantly constrains the local structure of a fold. Because of this, segments of backbone atoms often conform to similar secondary structure motifs. There are three broad classes of secondary structural elements: α-helices, β-sheets, and loops. Regular dihedral angles between consecutive triplets of backbone atoms characterize sheets and helices. Loops correspond to disordered regions without regular patterns. Examples of each motif are illustrated in Figure 1.2B.

The three-dimensional tertiary structure (*fold*) of a protein reflects its function, and it is uniquely determined by its amino acid sequence (up to the environment). The fold itself is stabilized by inter-residue interactions such as salt bridges, hydrogen bonds, disulfide bonds, and Van der Waals forces as well as hydrophobic interactions with the surrounding aqueous environment. An example of protein tertiary structure is shown in Figure 1.2C.

To carry out tasks in the cell, proteins often interact with one another to form a complex. Quaternary structure is the three-dimensional shape of two or more interacting protein chains. To avoid confusion, we often refer to proteins with multiple chains as *multimers* and those with a single chain as *monomers*.

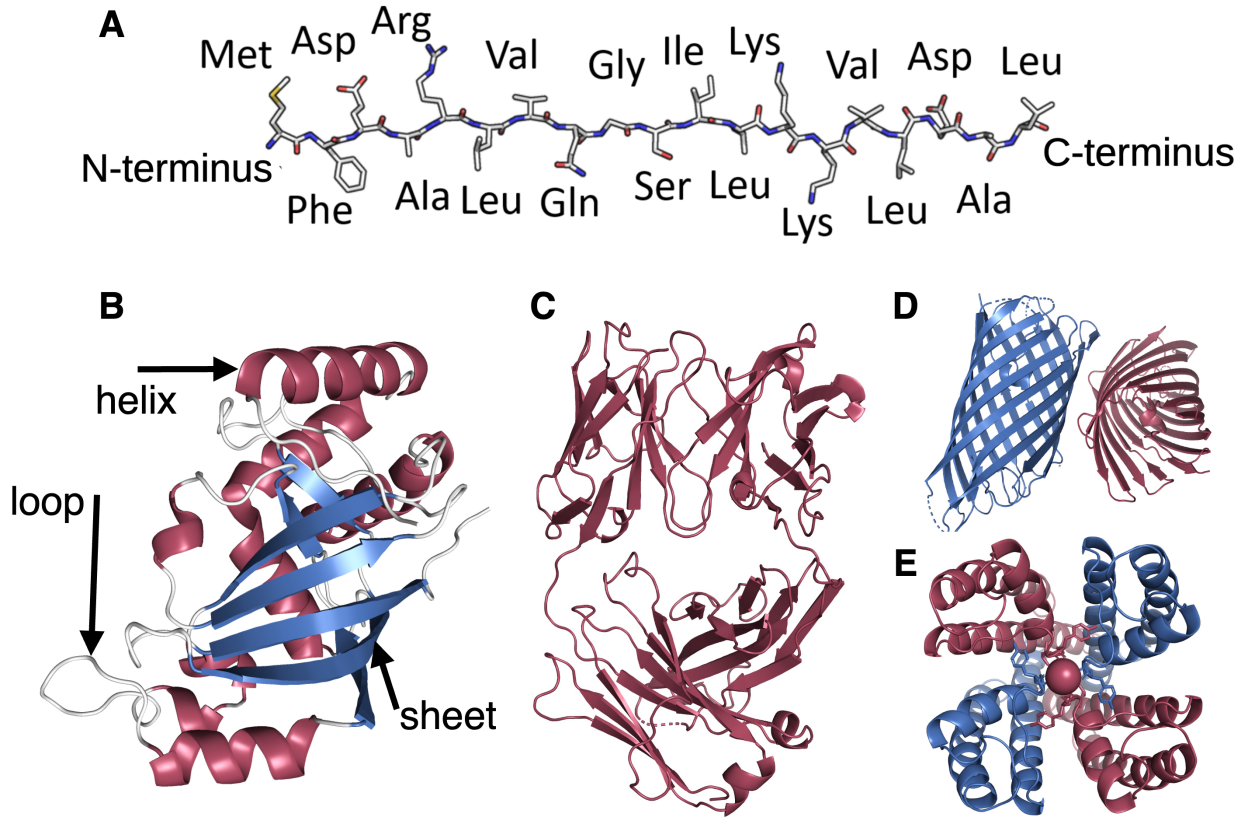

Figure 1.2: **The four levels of protein structure.** (**A**) Illustration of protein primary structure, showing each amino acid's backbone and side chain atoms in stick representation. (**B**) Cartoon representation of PDB entry 2A9K colored by secondary structure. Loops, helices, and sheets are colored white, red, and blue respectively. (**C**) Cartoon representation of PDB Entry 1JPS. (**D** and **E**) Cartoon representations of protein complex 2QOM (D) and 1BL8 (E) with individual chains colored red or blue. These two examples illustrate protein complexes formed by two or more copies of the same chain. The graphic used to illustrate primary sequence (A) was taken from https://bair.berkeley.edu/blog/2019/11/04/proteins.

1.3 Background: Machine Learning

Machine learning broadly describes the process of fitting computational models to examples. A list of parameters typically describes a model, and the "learning" process involves finding specific parameters that minimize a given objective function. A simple example is fitting the slope and intercept coefficients of a linear regression. *Deep learning* is a subset of machine learning that uses neural networks with many layers to parameterize models.

Geometric Deep Learning The field of geometric deep learning is concerned with modeling data with underlying geometric relationships. Typically, this involves developing architectures that are *invariant* or *equivariant* with respect to the action of some symmetry group. Notable examples include the permutation equivariance of graph neural networks and the translation equivariance of convolutional neural networks.

This thesis is entirely concerned with geometric deep-learning methods and the properties of the underlying networks. For example, we may wish to design a neural network $f : \mathcal{X} \to \mathcal{Y}$ that is *invariant* to some action $h : \mathcal{X} \to \mathcal{X}$, that is

$$f(\boldsymbol{x}) = f(h(\boldsymbol{x})) \ \forall \boldsymbol{x} \in \mathcal{X}. \tag{1.1}$$

More generally, we try to develop networks that capture properties of the input with respect to certain actions that should leave the property unchanged. In more technical terms, f is said to be *equivariant* with respect to the action of a group G and a corresponding set of transformations $T_g : \mathcal{V} \to \mathcal{V}$, if there exists a transformation $S_g : \mathcal{Y} \to \mathcal{Y}$ such that,

$$S_g[f(\boldsymbol{x})] = f(T_g[\boldsymbol{x}]), \ \forall g \in G.$$

The special case where S_g is the identity transformation corresponds to invariance.

In the case of proteins, we are primarily interested in equivariance to 3D rotations and translations, which the special Euclidean group characterizes in 3 dimensions, $SE(3) =$

$\{(R, \vec{\boldsymbol{t}})\}$. Given input coordinates $\vec{\boldsymbol{x}}$ and transformed coordinates $R\vec{\boldsymbol{x}} + \vec{\boldsymbol{t}}$, a network that is equivariant to rotations and translations receives the same gradient updates for each example. This allows the network to learn properties of coordinates independent of transformations that do not modify chemical characteristics.

In the remainder of this section, we provide some additional background and discuss broader classes of neural networks. Much of this thesis is dedicated to describing new machine learning architectures, and extensive details regarding their design are provided in each chapter.

1.4 Graph Neural Networks

Graph neural networks (GNNs) are functions on featurized graphs $\mathcal{G} = \big(\mathcal{V} = \{\boldsymbol{v}_i\}, \mathcal{E} = \{\boldsymbol{e}_{ij}\}\big)$, where each node $\boldsymbol{v}_i \in \mathbb{R}^{d_v}$ and edge $\boldsymbol{e}_{ij} \in \mathbb{R}^{d_e}$ is associated with some data. In the context of proteins, nodes may represent amino acids, and data vector $\boldsymbol{v}_i$ can be used to encode the corresponding residue type as a one-hot encoding (e.g., $d_v = 20$). Typically, edge data represent physical quantities such as inter-atom distance or the existence of molecular bonds. GNNs come in many different flavors (e.g., GAT [Vel+18], GCN [KW16], MPNN [Gil+17], SchNet [Sch+18], EGNN [SHW21a]), and most commonly consist of several repeated layers. Each layer is permutation invariant (or permutation equivariant) with respect to the ordering of the vertices. Each layer in a GNN typically has the form:

$$\boldsymbol{m}_{ij} = \phi_m^\ell\left(\boldsymbol{v}_i^\ell, \boldsymbol{v}_j^\ell, \boldsymbol{e}_{ij}\right) \qquad \text{(Message Passing)} \tag{1.2}$$

$$\boldsymbol{m}_i = \bigoplus_{j \in \mathcal{N}(i)} \boldsymbol{m}_{ij} \qquad \text{(Aggregation/Pooling)} \tag{1.3}$$

$$\boldsymbol{v}_i^{\ell+1} = \phi_v^\ell\left(\boldsymbol{v}_i^\ell, \boldsymbol{m}_i\right) \qquad \text{(Update)} \tag{1.4}$$

where ℓ is the index of the GNN layer, ϕ_m^ℓ, and ϕ_v^ℓ are functions (most often neural networks), and $\bigoplus$ is some permutation invariant or equivariant operation (most often a summation).

1.5 Transformers

The transformer model [Vas+17] is prevalent in almost every major sub-field of deep learning. It underlies large language models (e.g. BERT [Dev+18], T5 [Raf+19], GPT-3 [Bro+20], ChatGPT [Ast87]), and is now state-of-the-art for image classification (e.g. ViT [Dos+20], DETR [Car+20]) and generation (stable diffusion [Rom+21], RIN [JFC22]). The primary building block of the transformer architecture is the *attention head*. Here, we describe the ubiquitous special case, the *self-attention* head with cosine similarity, and refer the reader to [Lin+21] for a more comprehensive review. Beforehand, we briefly motivate the use of this architecture for molecular

The Transformer Model and Proteins

In the context of deep-learning applications to proteins, the transformer model has been applied with unparalleled success on tasks such as protein structure prediction [Jum+21; Bae+21; Lin+22; Wu+22], protein sequence modeling [Riv+19; Mad+23; Mei+21], and inverse folding [Ing+19; Jin+20; MLX22; Hsu+22], among others. Much of the success of this architecture stems from its ability to model sequential and graph-like data while naturally capturing long-range dependencies. Unlike text, protein residues have both sequential and spatial dependencies. Indeed, residues that interact in 3D space (i.e., in the protein's folded state) tend to co-evolve, which has implications on both the primary sequence and 3D structure. Nevertheless, spatially close residues may be arbitrarily distant in sequential order. These types of relationships are naturally captured by the transformer's self-attention mechanism (Section 1.5), which computes L^2 weights indicating interaction weights between all residue pairs. In contrast, an architecture like convolutional neural networks (CNNs)

has a much harder time modeling long-range dependencies because each layer operates only on sequentially local features. As another example, recurrent neural networks often fail to learn long-range dependencies as a result of exploding or vanishing gradients during training [BSF94].

Self-Attention

The self-attention head consists of three learnable functions, ϕ_Q, ϕ_K, and ϕ_V. Each learnable function $\phi_{(\cdot)}$ is typically implemented as a linear projection. However, it can be beneficial to consider more general classes of functions, as we will see in Chapter 2. Given a sequence of inputs $\{\boldsymbol{x}_i\}$, we associate each with a query, key, and value:

$$\boldsymbol{q}_i = \phi_Q(\boldsymbol{x}_i) \in \mathbb{R}^d, \quad \boldsymbol{k}_i = \phi_K(\boldsymbol{x}_i) \in \mathbb{R}^d, \quad \boldsymbol{v}_i = \phi_V(\boldsymbol{x}_i) \in \mathbb{R}^c. \tag{1.5}$$

Keys and queries for different data points are then compared to compute similarity scores σ_{ij}, and then normalized to obtain attention weights α_{ij}

$$\sigma_{ij} = \text{sim}\left(\boldsymbol{q}_i, \boldsymbol{k}_j\right), \qquad \alpha_{ij} = \text{softmax}_j\left(\sigma_{ij}\right) = \frac{\exp \sigma_{ij}}{\sum_j \exp \sigma_{ij}}. \tag{1.6}$$

Most often, $\text{sim}\left(\boldsymbol{q}_i, \boldsymbol{k}_j\right)$ is implemented as the *cosine similarity* between query and key,

$$\text{sim}\left(\boldsymbol{q}_i, \boldsymbol{k}_j\right) = \frac{\boldsymbol{q}_i^\top \boldsymbol{k}_j}{\sqrt{d}}. \tag{1.7}$$

To stabilize training, the $\sqrt{d}$ in the denominator was suggested in [Vas+17]. If we assume each entry in $\boldsymbol{q}_i, \boldsymbol{k}_j$ is distributed as a unit normal $\mathcal{N}\left(\mu = 0, \sigma = 1\right))$, then $\boldsymbol{q}_i^\top \boldsymbol{k}_j$ is distributed as $\mathcal{N}\left(\mu = 0, \sigma = \sqrt{d})\right)$.

Finally, an attention-modulated sum of the values is used to produce an embedding $\boldsymbol{a}_i$ of each data point:

$$\boldsymbol{a}_i = \sum_j \alpha_{ij} \boldsymbol{v}_j. \tag{1.8}$$

Each layer of the transformer consists of several self-attention heads (multi-head attention), each learning independent representations of the underlying data. The multi-head-attention block aggregates each head's output by concatenating the embeddings and linearly projecting back to the original input dimension with a learned weight matrix. This is illustrated in Figure 1.3.

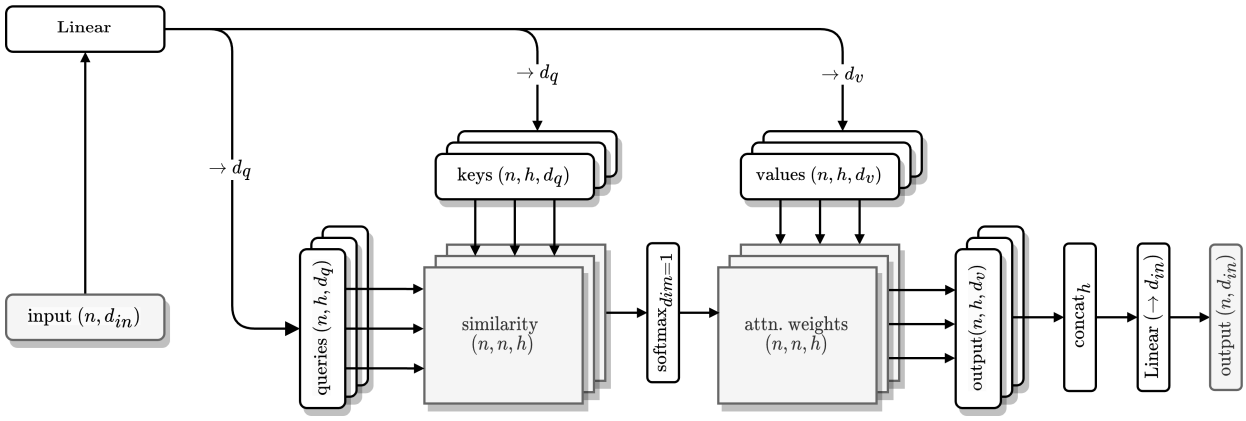

Figure 1.3: **Multi-head Attention**. Illustration of multi-head attention block. shapes of underlying data tensors are shown in parentheses. n is used to denote the number of input features, and h is used to denote the number of heads. Feature dimensions: d_{in}: input, d_q: keys and queries, d_v: values.

The attention head is easy to generalize to the GNN framework. In fact, the self-attention update described in Section 1.5 is already permutation-equivariant with respect to vertices. To incorporate edges, restrict the sums in Equations (1.6) and (1.8) to the neighbors of i.

$$\alpha_{ij} = \frac{\exp\left[\sigma_{ij}\right]}{\sum_{j \in \mathcal{N}(i)} \exp[\sigma_{ij}]} \qquad \boldsymbol{a}_i = \sum_{j \in \mathcal{N}(i)} \alpha_{ij} \boldsymbol{v}_j. \tag{1.9}$$

That is, we compute the attention weights for node i considering only nodes j with $i \sim j$.

CHAPTER 2

FIXED BACKBONE DESIGN

Protein fixed-backbone design, the task of predicting protein primary sequence and side-chain rotamers given backbone atom coordinates, has important applications to structure prediction, refinement, and more general design problems. Traditionally, the two sub-tasks are carried out independently, with inverse folding algorithms handling sequence design and side-chain packing (SCP) methods predicting rotamers for the designed sequence. Although the two tasks differ in output, both require reasoning around sequence-structure compatibility, and we hypothesized that a single architecture could effectively model both tasks.

In this chapter, we present AttnPacker, a deep-learning method for fixed-backbone design. Unlike existing methods, AttnPacker directly incorporates backbone 3D geometry to simultaneously compute all side-chain coordinates without delegating to a discrete rotamer library or performing expensive conformational search and sampling steps. This enables a significant increase in computational efficiency, decreasing inference time by over 100x compared to the deep-learning-based method DLPacker and physics-based RosettaPacker. Different from traditional SCP approaches, AttnPacker is easily extended to co-design sequences and side-chains, producing designs with sub-native Rosetta energy and high *in silico* consistency. Tested on the CASP13 and CASP14 native and non-native protein backbones, AttnPacker computes physically realistic designs, reducing steric clashes and improving RMSD and dihedral accuracy compared to state-of-the-art methods SCWRL4, FASPR, RosettaPacker, and DLPacker.

This chapter presents work with Jinbo Xu and was accepted to Proceedings of the National Academy of Sciences (PNAS) in April 2023 [MX22].

2.1 Introduction

Protein side-chain packing (PSCP) involves predicting the three dimensional coordinates of a protein's side-chain atoms given the backbone conformation (coordinates) and primary sequence. This problem has important applications to protein structure prediction [FBB07; Far+17; Chi+95], design [OJK15; Sim+19; DH99; Wat+16], and protein-protein interactions [WBA16; Hog+18]. Traditional methods for PSCP rely on minimizing some energy function over a set of rotamers from a fixed library [HPZ20; YSW07; Lia+11; BKK20; Alf+17; DSK09; XB06; Cao+10]. These methods tend to differ primarily in their choice of rotamer library [Ren+14; SD11; Dun02], energy function [Jum+18; YSW08; FW11; GPS16], and energy minimization procedure. Although many of them have shown success, the use of search heuristics coupled with a discrete sampling procedure could ultimately limit their accuracy. Currently, the fastest among them (OSCAR-star [Lia+11], FASPR [HPZ20], SCWRL4 [DSK09]) do not employ deep learning (DL) and are rotamer library-based.

Aside from traditional approaches, several machine learning (ML) methods have been developed for PSCP [NRB12; Mis+21; XWM20; XWM21; YSW07; Liu+17; XB06]. One of the earliest methods, SIDEPro [NRB12], attempts to learn an additive energy function over pairwise atomic distances for each side-chain rotamer. This is achieved by training a family of 156 feedforward networks - one for each amino acid and contacting atom type. The rotamer with the lowest energy is then selected. DLPacker [Mis+21] formulates PSCP as an image-to-image transformation problem and employs a deep U-net style neural network. The method iteratively predicts side-chain atom positions using a voxelized representation of the residue's local environment as input and outputs densities for the respective side-chain atoms. To convert the network's output to coordinates, the densities are then compared to a rotamer database, and the closest conformation is selected. The most recent version of OPUS-Rota4 [XWM21] uses a pipeline of multiple deep networks to predict side-chain coordinates. The method uses predicted side-chain dihedral angles to obtain an initial model and then applies gradient descent on predicted distance constraints to obtain a final structure. It is worth

noting that OPUS-Rota4, to the best of our knowledge, is the only ML-based PSCP method that directly utilizes multiple sequence alignments (MSAs) as part of its input. Apart from methods specifically designed for PSCP, many protein structure prediction methods also produce side-chain coordinate information. AlphaFold2 [Jum+21], and RosettaFold [Bae+21] can produce highly accurate structures from primary sequence and MSA information along with optional template structures. Another class of structure predictors, including ESMFold [Lin+22] and OmegaFold [Wu+22], rely on massive pre-trained language models to derive structure from primary sequence alone.

In a complementary line of research, a spate of machine learning methods have been developed for protein inverse folding [QZ20; Jin+20; Hsu+22; YZY22; Dau+22]. Akin to PSCP, inverse folding methods attempt to find a protein sequence that folds to a given backbone structure. Although the two tasks differ in output, both problems require reasoning around sequence-structure compatibility, and we hypothesized that a single architecture could effectively model both tasks.

Here we present AttnPacker, a new deep architecture for PSCP. Our method is inspired by recent advancements in modeling three-dimensional data, and architectures for protein structure prediction - most notably AlphaFold2 [Jum+21], Tensor Field Networks (TFN) [Tho+18], and the SE(3)-Transformer [Fuc+20]. By modifying and combining components of these architectures, we can significantly outperform other PSCP methods, in terms of speed, memory efficiency, side-chain atom clashes, and overall accuracy, using only features derived directly from the primary sequence and backbone coordinates.

Specifically, we introduce a deep graph transformer architecture leveraging both geometric and relational aspects of PSCP. Inspired by AlphaFold2, we propose *locality-aware triangle updates* to refine our pairwise features using a graph-based framework for computing triangle attention and multiplication updates. By doing this, we can significantly reduce the memory and build higher-capacity models. In addition, we explore several SE(3)-equivariant attention mechanisms and propose an equivariant transformer architecture for learning from

3D points.

Our method, AttnPacker, guarantees physically realistic rotamers with negligible deviation from ideal bond lengths and angles and minimal steric hindrance. AttnPacker also predicts per-residue confidence scores, which correlate strongly with side-chain root mean squared deviation (RMSD) and χ_1 dihedral error. AttnPacker significantly outperforms traditional PSCP methods on CASP13 and CASP14 native backbones with average reconstructed RMSD over 18% lower than the next best method on each test set. AttnPacker also surpasses the deep learning method DLPacker, with an 11% lower average RMSD while significantly improving side-chain dihedral accuracy. In addition to accuracy, we show that AttnPacker consistently produces packings with notably fewer atom clashes than other methods.

To demonstrate our method's broader applicability to protein design, we analyze AttnPacker's efficacy in (1) handling non-native backbones and (2) simultaneously designing primary sequence and packing side-chains. We show that AttnPacker performs favorably to state-of-the-art structure prediction methods AlphaFold2, RosettaFold, and OmegaFold, inferring accurate conformations from predicted backbone structures. For the second task, we train a variant of AttnPacker for co-design, which achieves native sequence recovery competitive with state-of-the-art methods, while also producing highly accurate packings. We validate these designs in-silico with Rosetta and find that AttnPacker-designed structures often yield sub-native Rosetta energy.

2.2 Methods

We now describe our input representation (Section 2.2.1), model architecture (Section 2.2.2), loss function (Section 2.2.6), training details (Section 2.2.8), and test datasets (Section 2.2.9). We also explain how our architecture enables co-design of sequence and structure (Section 2.2.3) and how estimates of packing quality can be learned in conjunction with these tasks (Section 2.2.5). Our post-processing procedure, which ensures ideal rotamer geometry

and low steric hindrance, is detailed in Section 2.2.7. Information regarding data collection and hyperparameters are withheld to Appendix A, Appendix A.1, and Appendix A.2. A schematic overview of AttnPacker is shown in Figure 2.1.

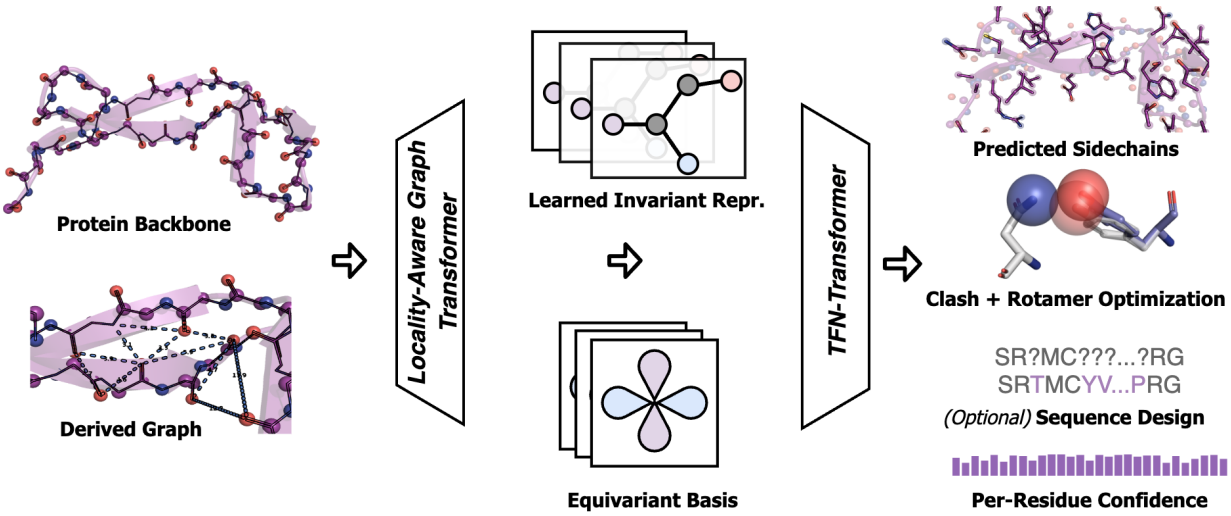

Figure 2.1: **Overview of AttnPacker.** AttnPacker generates input features from protein backbone coordinates and primary sequence. Backbone coordinates are also used to derive a spatial feature graph and equivariant basis for the TFN-Transformer sub-network. An invariant graph-transformer module processes the initial feature graph. The updated representation is passed to an equivariant TFN-Transformer that computes side-chain coordinates, per-residue confidence scores, and (optionally) a designed sequence. Predicted coordinates are post-processed to remove steric clashes and ensure idealized geometry.

2.2.1 Input Representation

We are given a protein $\mathcal{P} = (\{s_i\}, \{\vec{\boldsymbol{x}}_i^a\})$ with L residues, atoms $a \in \{N, C\alpha, C, O\}$, indexed by i. All input features are derived from primary sequence and backbone heavy-atom coordinates. The input to our network is a featurized graph

$$\mathcal{G}_\theta = \left(\{\boldsymbol{x}_i\}_{i=1..L}, \{\boldsymbol{e}_{ij} : d(i,j) < \theta\}\right) \tag{2.1}$$

where $d(x, y)$ denotes the distance between $C\alpha$ atoms for residues x and y, and $\theta \in \mathbb{R}$ is a pre-defined threshold.

Input Features Residue (node) features consist of encodings for amino acid type, backbone dihedral angles, relative sequence position, and the number of atoms in a residue's immediate micro-environment (centrality). Pair (edge) features encode inter-residue orientation as defined by [Yan+20b], along with inter-atom distance and a joint encoding of relative sequence separation and amino acid type. An overview of input feature types and the corresponding shape can be found in Table 2.1. AttnPacker's input embedding procedures are provided in Table 2.2 and Figure 2.2.

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 (2.2).
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[Yan+20b].

Table 2.1: **Input features used by AttnPacker**. The shape of the corresponding type for a protein with L residues is shown below each feature.

Most of AttnPacker's input features are standard amongst contemporary protein design methods [Ing+19; QZ20; Dau+22; MLX22; JX21; JKS21]. The exception is residue degree centrality. The use of centrality-based encodings was studied in [Yin+21], where the authors found that transformers significantly benefit from adding 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 [Yin+21]). Originally, we included SS3 secondary structure as a residue feature but found that this had no impact on average rotamer RMSD scores.

Input Embedding Encodings of each feature are generated following the procedures in Table 2.2. To incorporate sequence information into pair features, we learn two separate linear embeddings, E_A and E_B, for residue types. The pair feature for residues i and j of type s_i and s_j is given by the outer-sum of $E_A(s_i)$ and $E_B(s_j)$. Following [Jum+21], we also add a learned embedding of the signed relative sequence separation $i - j$. To enable sequence design, a separate $\langle \mathsf{MASK} \rangle$ token is added to represent a "missing residue" type. A diagram of the embedding procedure is shown in Figure 2.2.

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 2.2: **Description of Input Embedding Procedures**. 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. These procedures are referenced in Figure 2.2.

To generate $C\beta$ atom positions for orientation and centrality, we impute a unit vector in the direction $\vec{\boldsymbol{x}}_i^{C\beta} - \vec{\boldsymbol{x}}_i^{C\alpha}$ before computing the respective features. The imputed vector is

calculated as in [Jin+20] using

$$\sqrt{\frac{1}{3}}\langle \vec{\boldsymbol{n}} \times \vec{\boldsymbol{x}} \rangle - \sqrt{\frac{2}{3}}\langle \vec{\boldsymbol{n}} + \vec{\boldsymbol{x}} \rangle \tag{2.2}$$

where $\langle \vec{\boldsymbol{x}} \rangle = \vec{\boldsymbol{x}}/\|\vec{\boldsymbol{x}}\|_2$, $\vec{\boldsymbol{n}} = \vec{\boldsymbol{x}}_i^N - \vec{\boldsymbol{x}}_i^{C\alpha}$, and $\vec{\boldsymbol{x}} = \vec{\boldsymbol{x}}_i^C - \vec{\boldsymbol{x}}_i^{C\alpha}$.

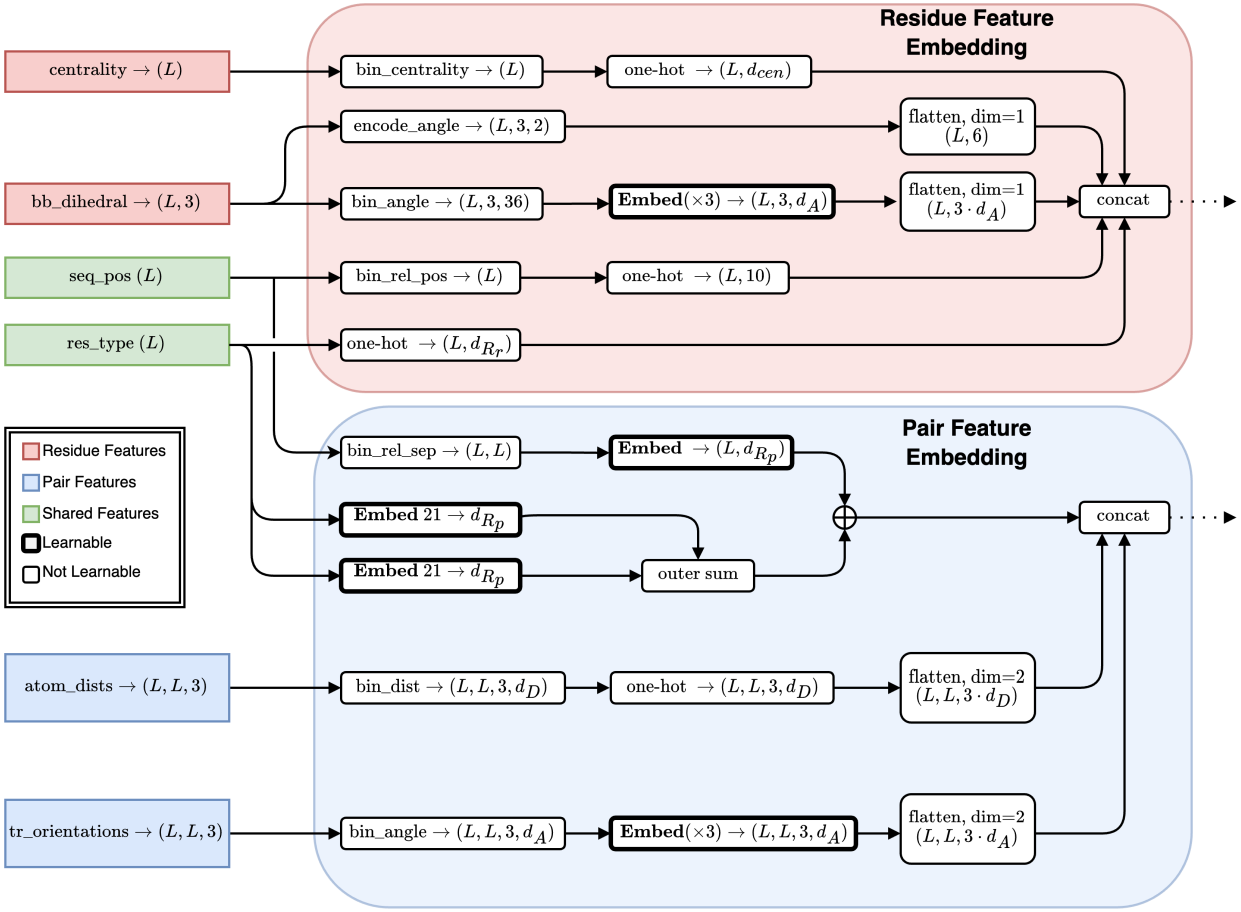

Figure 2.2: **Overview of input feature embedding**. The feature shape is shown in parentheses. Residue and pairwise features are embedded separately. Only information derived from primary sequence and backbone coordinates is considered. Full explanations of procedures referenced in this figure are given in Table 2.2. 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.

2.2.2 Network Architecture

Our network comprises two modules. The first is a *locality-aware graph transformer* which selectively updates node and pair features, and the second is a TFN-based SE(3)-equivariant transformer inspired by Fuchs et al. [Fuc+20].

The initial graph transformer learns invariant representations of each residue and edge in the input graph. The TFN-transformer incorporates backbone coordinates and uses the learned representations to predict amino acid types and side-chain atom coordinates.

Unlike the locality-aware graph transformer, the TFN-Transformer does not update pair features. Instead, pair features are passed through an MLP to generate radial kernels transforming residue features at each attention layer. We hypothesized that learning pair representations may lead to better performance, which is justified by our ablation studies (see Section 2.4). The choice of components also allows us to predict the 3D coordinates of all side-chain atoms for a given protein without relying on rotamer libraries or expensive conformational sampling.

In the remainder of this section, we provide an in-depth overview of our architecture components. Ablation studies comparing the impact of specific components are reported in Section 2.4, and Tables 2.15 and 2.16. Schematic representations of attention blocks are given in Figures A.1 and A.2.

Locality-Aware Graph Transformer

Before incorporating backbone coordinates into AttnPacker, we first generate an invariant graph representation using a deep transformer network. This module draws components from the Evoformer, introduced in AlphaFold2, for processing MSA and pair features. Namely, we introduce variants of pair-biased self-attention for node updates, and triangle-attention and multiplication for pair updates. Different from the protein structure prediction setting for which these updates were designed, we are given a backbone as input. We take advantage of this by restricting updates to spatially close residues and generalize each component

to arbitrary graphs. Concretely, we restrict residue attention to the top-k spatially closest nearest neighbors as defined by $C\alpha$ distance. Next, we introduce *locality-aware triangle updates*, which non-trivially restrict multiplication (Algorithm 1) and attention (Algorithm 2) to a subset of pre-defined triangles in the input graph. Throughout this section, we describe our modifications and graph representation in more detail. We provide pseudocode for triangle attention and triangle multiplication updates as implemented in AlphaFold2 in Algorithms 1 and 2 for completeness. For simplicity, we describe all procedures for undirected graphs, though it is straightforward to extend to directed graphs.

Algorithm 1 Triangle Multiplicative Update ("outgoing", [Jum+21])

Input
  $\boldsymbol{z}_{ij}$: Pair Features, in $\mathbb{R}^{d_z}$
  c: intermediate dimension, in $\mathbb{N}$
Output
  $\boldsymbol{z}'_{ij}$: Updated Pair Features, in $\mathbb{R}^{d_z}$

function TRIANGLEMULTIPLICATIONOUTGOING($\boldsymbol{z}_{ij}$, $c = 128$) :
  $\boldsymbol{z}_{ij} \leftarrow \text{LayerNorm}\left(\boldsymbol{z}_{ij}\right)$
  $\boldsymbol{a}_{ij},\ \boldsymbol{b}_{ij} = \text{sigmoid}\left(\text{Linear}\left(\boldsymbol{z}_{ij}\right)\right) \odot \text{Linear}\left(\boldsymbol{z}_{ij}\right)$ ▷ $\boldsymbol{a}_{ij},\ \boldsymbol{b}_{ij} \in \mathbb{R}^{c}$
  $\boldsymbol{g}_{ij} = \text{sigmoid}\left(\text{Linear}\left(\boldsymbol{z}_{ij}\right)\right)$ ▷ $\boldsymbol{g}_{ij} \in \mathbb{R}^{d_z}$
  $\boldsymbol{z}'_{ij} = \boldsymbol{g}_{ij} \odot \text{Linear}\left(\text{LayerNorm}\left(\sum_k \boldsymbol{a}_{ik} \odot \boldsymbol{b}_{jk}\right)\right)$ ▷ $\boldsymbol{z}'_{ij} \in \mathbb{R}^{d_z}$
  return $\boldsymbol{z}'_{ij}$

Global Triangle Updates Global triangle updates have the general form:

$$\boldsymbol{e}_{ij}^{(\ell+1)} = f\left(\boldsymbol{e}_{ij}^{(\ell)}, \left\{\left(\boldsymbol{e}_{ik}^{(\ell)}, \boldsymbol{e}_{jk}^{(\ell)}\right)\right\}_{k=1,\ldots,L}\right) \tag{2.3}$$

where $\boldsymbol{e}^{(\ell)}$ denotes the pair features at layer ℓ in the network. The pair feature updates presented in Algorithms 1 and 2 can be seen as aggregating information over each triangle in the input graph $\mathcal{G}$. The algorithms correspond to the special case where $\mathcal{G}$ is complete, resulting in $\Omega(L^3)$ time complexity per operation at each block. Moreover, an additional $\Omega(h \cdot L^3)$ space is needed to store triangle attention logits (i.e. α_{ijk} in Algorithm 2) for

backpropagation, where h is the number of attention heads. Although techniques such as gradient checkpointing [Che+16] or mixed precision training [Mic+17] may be applied to reduce some of this burden, the time and space complexity significantly limit the practitioner's ability to experiment with these operations, as batch sizes, sequence crop lengths, and precision must all be reduced. For example, triangle attention over a complete graph with 1000 nodes would require over four gigabytes of storage per head, assuming standard 32-bit floating point precision.

Algorithm 2 Triangle Attention Head ("starting", [Jum+21])

1: **Input**
2: $\boldsymbol{z}_{ij}$: Pair Features, in $\mathbb{R}^{d_z}$
3: d_h: intermediate dimension, in $\mathbb{N}$
4: **Output**
5: $\boldsymbol{z}_{ij}$: Output of a single triangle-attention head, in $\mathbb{R}^{d_h}$
6:
7: **function** TRIANGLEATTENTIONHEADSTARTING($\boldsymbol{z}_{ij}$, $d_h = 32$):
8: $\boldsymbol{z}_{ij} \leftarrow \text{LayerNorm}\left(\boldsymbol{z}_{ij}\right)$
9: $\boldsymbol{q}_{ij},\ \boldsymbol{k}_{ij},\ \boldsymbol{v}_{ij},\ \boldsymbol{g}_{ij}, b_{ij} = \text{LinearNoBias}\left(\boldsymbol{z}_{ij}\right)$ ▷ $\boldsymbol{q}_{ij},\ \boldsymbol{k}_{ij},\ \boldsymbol{v}_{ij},\ \boldsymbol{g}_{ij} \in \mathbb{R}^{d_h},\ b_{ij} \in \mathbb{R}$
10: *# Similarity*
11: $\sigma_{ijk} = \left(\frac{1}{\sqrt{d_h}} \boldsymbol{q}_{ij}^\top \boldsymbol{k}_{ik} + b_{jk}\right)$
12: $\alpha_{ijk} = \text{softmax}_k\left(\sigma_{ijk}\right)$
13: $\boldsymbol{z}_{ij} = \text{sigmoid}\left(\boldsymbol{g}_{ij}\right) \odot \sum_k \alpha_{ijk} \boldsymbol{v}_{ik}$
14: **return** $\boldsymbol{z}_{ij}$

Locality Aware Triangle Updates Locality-aware triangle updates generalize Equation (2.3) by restricting only to triangles of an underlying graph $\mathcal{G}$:

$$\boldsymbol{e}_{ij}^{(\ell+1)} = f\left(\boldsymbol{e}_{ij}^{(\ell)}, \left\{\left(\boldsymbol{e}_{ik}^{(\ell)}, \boldsymbol{e}_{jk}^{(\ell)}\right) : k \in \mathcal{N}(i) \cap \mathcal{N}(j)\right\}_{k=1,\ldots,L}\right). \tag{2.4}$$

Concretely, we replace line 11 in Algorithm 1 with

$$\boldsymbol{z}_{ij}^{'} = \boldsymbol{g}_{ij} \odot \text{Linear}\left(\text{LayerNorm}\left(\sum_{k \in \mathcal{N}(i) \cap \mathcal{N}(j)} \boldsymbol{a}_{ik} \odot \boldsymbol{b}_{jk}\right)\right), \tag{2.5}$$

where the feature $\boldsymbol{z}_{ij}$ is valid only for those edges $(i, j) \in E(G)$. Similarly, we can restrict the softmax and sum in lines 11 and 12 of Algorithm 2 to the same subset as Equation (2.5)

$$\alpha_{ijk} = \text{softmax}_{k \in \mathcal{N}(i) \cap \mathcal{N}(j)} \left(\sigma_{ijk} \right) \tag{2.6}$$

$$\boldsymbol{z}_{ij} = \text{sigmoid} \left(\boldsymbol{g}_{ij} \right) \odot \sum_{k \in \mathcal{N}(i) \cap \mathcal{N}(j)} \alpha_{ijk} \boldsymbol{v}_{ik}. \tag{2.7}$$

We refer to the modifications proposed in Equations (2.5) and (2.6) as "locality-aware" triangle updates. To see the correspondence, we can quickly implement local triangle updates by computing a global triangle mask to modulate similarity calculations (see Figure 2.3). This approach does not yield the time and space savings previously discussed but does serve as a simple proof of concept and drop-in replacement for existing models.

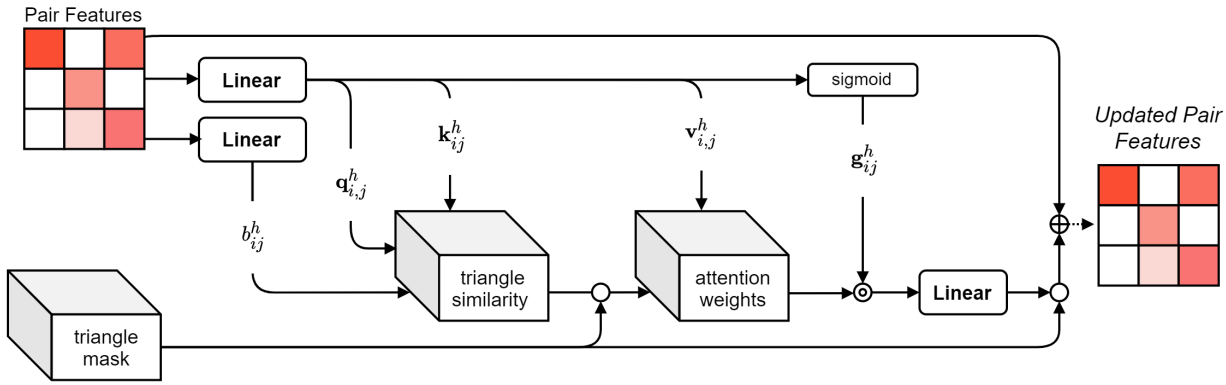

Figure 2.3: **A simple implementation of local triangle attention updates**. A mask of dimension $L \times L \times L$ is used to control which triangles (and respective edge features) are updated during attention calculations. Masked positions are kept unchanged, and unmasked positions are filled with a large negative value (-10^6) before softmax is applied to compute attention weights.

Pair-Biased Self-Attention Pair features are used to bias residue self-attention logits. The standard formula for computing key-value similarity (Equation (1.6)) is modified to

$$\sigma_{ij} = \frac{1}{\sqrt{2d}} \left(\boldsymbol{q}_i^\top \boldsymbol{k}_j + b_{ij} \right), \qquad \alpha_{ij} = \frac{\exp \sigma_{ij}}{\sum_{j \in \mathcal{N}(i)} \exp \sigma_{ij}}, \tag{2.8}$$

where d is the dimension of $\boldsymbol{q}_i$, and $b_{ij} = \text{Linear}(\boldsymbol{z}_{ij})$ is a learnable scalar obtained as a linear projection of pair feature $\boldsymbol{z}_{ij} \in \mathbb{R}^{d_z}$.

Implementation Details Theoretically, the time and space complexity of node updates is $O(E)$, where E is the number of edges in the input graph. For triangle updates, complexity is bounded by the number of triangles in the input graph. Unfortunately, these bounds are difficult to achieve in practice, as the features are represented as tensors of a fixed dimension. Implementing these updates efficiently while taking advantage of built-in broadcasting and parallelism in popular deep learning software (e.g. PyTorch [Pas+19] or TensorFlow [Mar+15]) requires significant care. In light of this, we carefully construct our input graphs to exploit parallelism. These considerations and a formal description of our graph representation are described in the next sections.

Graph Representation Distinct from the protein structure prediction setting, PSCP requires the protein backbone coordinates as input. With this information, we can perform updates in a geometrically motivated manner – by considering only residue whose atoms are spatially close. Given backbone $C\alpha$ coordinates $\{\vec{\boldsymbol{x}}_k^{c\alpha}\}$ Our input graph $G = (V, E; \theta)$ is defined by the adjacency relation

$$\text{adj}_\theta\left(i, j; \{\vec{\boldsymbol{x}}_k^{c\alpha}\}_k\right) \triangleq \text{adj}_\theta\left(i, j\right) = \|\vec{\boldsymbol{x}}_i^{c\alpha} - \vec{\boldsymbol{x}}_j^{c\alpha}\|_2 < \theta. \tag{2.9}$$

In practice, we restrict the neighborhood of a node i to only the k spatially closest neighbors. This fixes the maximum degree in our graph and allows us to represent the adjacency function with a fixed index tensor of shape $L \times k$, plus an additional $L \times k$ Boolean mask tensor to indicate those residues with fewer than k neighbors.

Similarly, we can order the triangles in $\mathcal{G}$ based on the maximum side length,

$$\ell(\Delta_{ijk}) = \max\left(\{\|\vec{\boldsymbol{x}}_s^{c\alpha} - \vec{\boldsymbol{x}}_t^{c\alpha}\|_2 : \ s,t \in (i,j,k)\}\right), \tag{2.10}$$

and restrict the triangles considered for each edge to the $k^{'}$ of minimum weight. We note that Equation (2.9) already implies that residues comprising any triangle in $\mathcal{G}$ have maximum pairwise distance θ.

To efficiently implement triangle updates, we store two separate adjacency lists $A_{ij}^{i}[\cdot]$, $A_{ij}^{j}[\cdot]$ for each edge (i,j); consisting of triangle edges incident node i, and triangle edges incident node j. Importantly, the lists are ordered such that $(i,j), A_{ij}^{i}[k], A_{ij}^{j}[k]$ is a triangle in $\mathcal{G}$. The adjacency lists use space $\Omega(E \times k^{'})$.

Time and Space Complexity We restrict our input graph to have a maximum degree at most k by selecting only the nearest neighbors of each node. This means that the maximum size of E is $k \cdot L$. Furthermore, no edge can appear in more than $k-1$ triangles, as this would contradict the maximum degree constraint. It follows that $\mathcal{G}$ contains at most $k^2 \cdot L$ triangles. Based on the bounds given in the previous sections, we can efficiently compute triangle updates in time and space $O(k^2 \cdot L)$, and residue updates in time $O(k \cdot L)$.

When the underlying point set consists of backbone atom coordinates for a protein, a reasonable choice of distance threshold θ results in a relatively small set of edges and triangles. Setting $k = 24$ and $\theta = 16$ resulted in approximately the same rotamer prediction accuracy using $k = \theta = \infty$. Technically, since k is fixed, updates are linear in the sequence length. In practice, the performance gains become noticeable when the sequence length L exceeds 200 residues.

TFN-Transformer

The second module is an SE(3)-equivariant neural network inspired by the SE(3)-Transformer introduced by Fuchs et al. [Fuc+20]. Like the SE(3)-Transformer, attention heads use TFNs

to produce keys and values for scalar and point features.

Our implementation differs in a few key areas. First, we use shared attention with cosine similarity (see Table 2.14). That is, we aggregate the attention logits of each feature type (e.g., scalars and points) to produce shared attention weights. Furthermore, in each attention block, we augment the input to the TFN radial kernel with pairwise distance information (Algorithm 5) between hidden coordinate features and make further use of the pair features to bias self-attention weights and update scalar features (Algorithm 3). Implementation details are provided throughout the remainder of this section.

Tensor Field Networks (TFNs) TFNs operate on irreducible representations of the rotation group SO(3). These representations have dimension $2\ell + 1$, where ℓ denotes the *rotation order*. The rotation orders $\ell = 0, 1, 2$ correspond to scalars, vectors in 3-space, and symmetric traceless matrices [Tho+18].

The input to a TFN can be represented as a vector field of the form

$$\boldsymbol{f}\left(\vec{\boldsymbol{x}}\right) = \sum_i \boldsymbol{x}_i \delta\left(\vec{\boldsymbol{x}} - \vec{\boldsymbol{x}}_i\right), \tag{2.11}$$

where δ is the Dirac delta function, $\{\vec{\boldsymbol{x}}_i\}$ are 3D points, and $\{\boldsymbol{x}_i\}$ are features of the corresponding point such as residue identity, charge, etc., represented as the concatenation of elements in SO(3). That is

$$\boldsymbol{x}_i = \bigoplus_{\ell \geq 0} \boldsymbol{x}_i^\ell, \qquad \boldsymbol{x}_i^\ell \in \mathbb{R}^{d_\ell \times 2\ell + 1}, \tag{2.12}$$

where $\bigoplus$ denotes concatenation.

TFN Attention Head We begin by describing a single Attention head (Algorithm 3). We use d_ℓ to denote the input dimension and h^ℓ to denote the hidden dimension (head dimension) of type ℓ features. Each attention head has a separate learnable weight γ^ℓ for

Algorithm 3 TFN-Transformer Attention Head

Input
  $\boldsymbol{x}_i^\ell$: type-ℓ features in $\mathbb{R}^{d_\ell \times 2\ell+1}$
  $\boldsymbol{z}_{ij}$: pair features in $\mathbb{R}^{d_z}$
  $\mathcal{B}_{ij}^{\ell,k}$: equivariant basis mapping type ℓ features to type k features
  $\vec{\boldsymbol{r}}_{ij}$: input relative coordinates in $\mathbb{R}^{1\times 3}$
  $\mathcal{N}_i$: List of neighbor indices for each residue $1..i$
Output
  $\boldsymbol{o}_i^\ell$: attention head features for each input type $\ell = 0\ldots$
function TFNAttnetionHead($\boldsymbol{x}_i^\ell,\ \boldsymbol{z}_{ij},\ \mathcal{B}_{ij}^{\ell,k},\ r_{ij},\ \mathcal{N}_i$):
  $\boldsymbol{z}_{ij},\ \boldsymbol{x}_i^\ell \leftarrow \text{LayerNorm}\left(\boldsymbol{z}_{ij}\right),\ \text{SE3Norm}(\boldsymbol{x}_i^\ell)$
  # Compute TFN keys and values
  $\tilde{\boldsymbol{z}}_{ij} \leftarrow \text{AugmentPairFeats}\left(\boldsymbol{z}_{ij}, \boldsymbol{x}_i^1, r_{ij}\right)$
  $\boldsymbol{k}_{ij}^\ell, \boldsymbol{v}_{ij}^\ell = \text{TFN}(\boldsymbol{x}_i^\ell, \tilde{\boldsymbol{z}}_{ij}, \mathcal{B}_{ij}^{\ell,k}, \mathcal{N}_i)$ ▷ defined for i, j such that $j \in \mathcal{N}_i \setminus i$
  $\boldsymbol{q}_i^\ell = \text{SE3Linear}(\boldsymbol{x}_i^\ell)$ ▷ $\boldsymbol{k}_i^\ell, \boldsymbol{q}_i^\ell, \boldsymbol{v}_i^\ell \in \mathbb{R}^{h^\ell}$
  $b_{ij} = \text{LinearNoBias}(\boldsymbol{z}_{ij})$ ▷ pair bias, $b_{ij} \in \mathbb{R}$
  $\hat{\boldsymbol{z}}_{ij} = \text{LinearNoBias}(\boldsymbol{z}_{ij}) \circ \text{sigmoid}\left(\text{LinearNoBias}(\boldsymbol{z}_{ij})\right)$ ▷ pair features, $\hat{\boldsymbol{z}}_{ij} \in \mathbb{R}^{d_{z'} \times 1}$
  $w^\ell = \frac{1}{\sqrt{(2\ell+1)\cdot h^\ell}}$ ▷ head weights for each input type $\ell = 0\ldots$
  # Compute pair similarity and self-similarity
  $\sigma_{ij}^\ell = \sum_{c=1}^{d_\ell} \boldsymbol{q}_{ic}^\ell \cdot \boldsymbol{k}_{ijc}^\ell$
  $\sigma_{ii}^\ell = \text{SE3Linear}(\boldsymbol{x}_i^\ell)$
  # Share Attention weights for each i and each type
  $\alpha_{ij} = \text{softmax}_{j\in\mathcal{N}_i}\left(b_{ij} + \sum_\ell w^\ell \cdot \gamma^\ell \cdot \sigma_{ij}^\ell\right)$
  $\boldsymbol{o}_i^\ell = \sum_{j\in\mathcal{N}_i} \alpha_{ij} \cdot \boldsymbol{v}_{ij}^\ell$
  $\boldsymbol{o}_i^{\text{pair}} = \sum_{j\in\mathcal{N}_i} \alpha_{ij} \cdot \hat{\boldsymbol{z}}_{ij}$
  # Scalar output augmented with attention-weighted edge information
  $\boldsymbol{o}_i^{(0)} \leftarrow \text{concat}\left(\boldsymbol{o}_i^{(0)}, \boldsymbol{o}_i^{\text{pair}}\right)$
  return $\boldsymbol{o}_i^\ell$

each input type ℓ. This weight is the softplus of a learnable scalar and is initialized so that $\gamma^\ell = 1$ (e.g., [Jum+21, Supplementary Information, Section 1.8.2]). 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 [Den+21].

Algorithm 4 SE(3)-Equivariant Normalization

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

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

Normalization We propose a method similar to Layer normalization (Algorithm 4) with a norm-based non-linearity. Several papers have proposed SE(3)-Equivariant Normalization schemes (e.g., [Fuc+20; Den+21; Jin+20]), most include some form of layer-normalization [BKH16], 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) caused instability. In light of this we simply learn a scale and bias σ^ℓ and β^ℓ for each feature type.

Algorithm 5 Augment Edge Features

Input
  $\boldsymbol{z}_{ij}$: pair features in $\mathbb{R}^{d_z}$
  $\vec{\boldsymbol{c}}_i^{(k)}$: 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
  $\tilde{\boldsymbol{z}}_{ij}$: augmented edge features in $\mathbb{R}^{d_z + 2\cdot d_p}$

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

Pair Feature Augmentation As mentioned in Section 2.2.2, we do not update pair features in the TFN-Transformer. Instead, pair features are used to parameterize radial kernels transforming the features of incident residues.

In augmenting the pair features with distance information (Algorithm 5), we chose 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.

TFN-transformer The TFN-Transformer (Algorithm 6) 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 and dimensions. A schematic overview is given in Figure A.2.

Algorithm 6 TFN-Transformer

Input
  $\boldsymbol{s}_i$: scalar residue features in $\mathbb{R}^{s\times 1}$
  c_i: backbone coordinates for each residue in $\mathbb{R}^{c\times 3}$
  $\boldsymbol{z}_{ij}$: 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
  $\boldsymbol{s}_{out,i}$: updated scalar residue features in $\mathbb{R}^{d_{scalar}\times 1}$
  $\boldsymbol{c}_{out,i}$: updated coordinate features $\in \mathbb{R}^{d_{coord}\times 3}$
function TFN-TRANSFORMER($\boldsymbol{s}_i, \boldsymbol{c}_i, \boldsymbol{z}_{ij}, \mathcal{N}_i$) :
  $r_{ij} = \boldsymbol{c}_{j,0} - \boldsymbol{c}_{i,0}$ ▷ relative coordinates $\in \mathbb{R}^{1\times 3}$
  $\mathcal{B}_{ij}^{\ell,k} = \text{COMPUTEEQUIVARIANTBASIS}\left(r_{ij}, \ell_{max}\right)$
  $\boldsymbol{c}_i \leftarrow \text{concat}_k\left(\boldsymbol{c}_{i,k} - \boldsymbol{c}_{i,0}\right)$
  # Equivariant Input Mapping
  $\boldsymbol{x}_{hid,i}^{\ell} = \text{TFN}\left((\boldsymbol{s}_i, \boldsymbol{c}_i), \boldsymbol{z}_{ij}, \mathcal{B}_{ij}^{\ell,k}, \mathcal{N}_i\right)$ ▷ $\tilde{\boldsymbol{x}}_i^{\ell} \in \mathbb{R}^{d_\ell\times 2\ell+1}$, $\ell_{in} = 0, 1$ and $\ell_{out} = 0..\ell_{max}$
  $\boldsymbol{x}_{hid,i}^{\ell} \leftarrow \text{SE3Norm}\left(\boldsymbol{x}_{hid,i}^{\ell}\right)$

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

Memory Optimization

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

	Storage for attention logits using B-Float presision	$L = 300$	$L = 500$
Triangle Attention	$B \cdot L^3 \cdot h{\cdot}D$	5.2 Gb	24.0 Gb
Locality Aware Triangle Attention	$B \cdot 2N^2 L \cdot h \cdot D$	0.10 Gb	0.17 Gb

Table 2.3: **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 $N = 30$, and the floating point precision $B = 32$.

Fuchs et al. [Fuc+20]. 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 can obtain a factor $\approx o_{in} \cdot o_{out}$ reduction in memory by simplifying the calculation of tensor products between representations. In practice, this

corresponds to refactoring the matrix multiplications 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.

Full Architecture

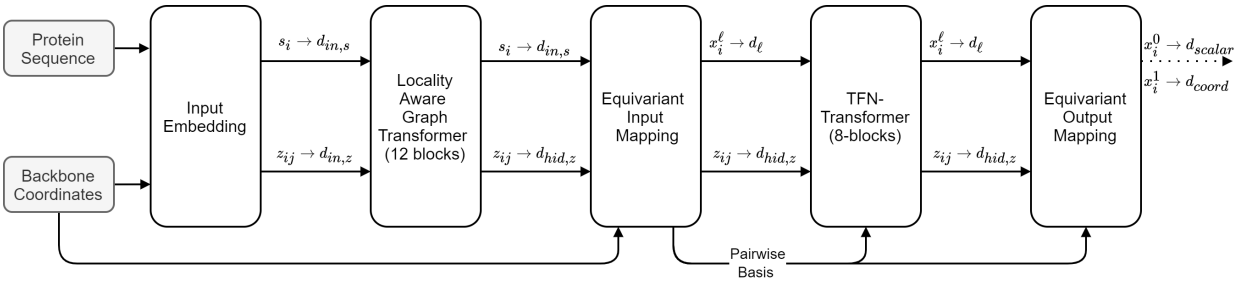

Figure 2.4: **Full architecture of AttnPacker**. we use s_i to denote scalar (residue) features, and z_{ij} to denote pair features. Backbone coordinates define residue and pair adjacencies and are included in the equivariant input mapping (see Algorithm 6). The Equivariant input mapping returns type 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.

Our full architecture is shown in Figure 2.4. It consists of an input embedding (Figure 2.2) to produce scalar features for residues and pairs and is followed by a locality-aware graph transformer and TFN-transformer. Pair features are only modified in the locality-aware graph transformer block. The TFN-Transformer still uses pair features to produce radial kernels, bias attention logits, and augment the output of each attention head.

2.2.3 Sequence Design

On top of rotamer prediction, we train a variant of AttnPacker (AttnPacker + Design) to residue types from partial sequence information. Both PSCP and inverse-folding involve reasoning over sequence and structure compatibility, and we hypothesized that a single architecture could be used for both tasks. To enable this, we randomly mask and corrupt subsets of the input sequence during training and ask our model to predict the amino acid type and side-chain conformation for these missing residues. To generate sequences, we perform Gibbs sampling, as described in [Joh+21].

2.2.4 Rotamer Conditioning

AttnPacker+Design is able to condition on partial sequence information. For residues with known identities, 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.

2.2.5 Per-Residue Confidence Prediction

Estimating Packing Quality

Along with side-chain conformations, AttnPacker also outputs per-residue estimates of sequence prediction and packing quality. This is useful in the protein design setting, where the practitioner may be interested in determining the degree of sequence-structure compatibility. To estimate packing quality, we predict a per-residue local distance dissimilarity test (plDDT) score for a predefined atom in each amino acid side-chain.

$$\text{plDDT}_i \quad = \quad \frac{1}{|\mathcal{N}(i)|} \cdot \sum_{j \in \mathcal{N}(i)} \text{lDDT}\left(\text{abs}\left(d_{ij} - d_{ij}^*\right)\right) \tag{2.13}$$

where $d_{ij},\ d_{ij}^*$ are the predicted and ground truth distance between selected atoms in residues i and j,

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

and

$$\text{lDDT}(d) = \frac{1}{4} \cdot \sum_{k=0}^{3} \mathbf{1}_{d \leq 2^{k-1}} \tag{2.14}$$

To compute a loss for this prediction, 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 2.4 For each residue type, we selected the most distal atom in the χ_1 or χ_2 dihedral group, with $C\alpha$ and $C\beta$ chosen for Gly and Ala, respectively (see Table 2.4). In Figure 2.9, we show that this metric correlates strongly with ground-truth side-chain plDDT, side-chain RMSD, and χ_1 dihedral error.

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 2.4: Definition of the distal side-chain atom for each residue type.

Packing Large Proteins We remark that confidence predictions are used to produce accurate side-chain packings of large proteins. To do this, we linearly crop the input structure and sequence into overlapping segments of a predefined length L_{crop}, where consecutive

segments overlap on the last and first $L_{\text{crop}}/2$ residues, respectively. For residues contained in multiple segments, we choose the side chain/sequence predictions with higher predicted confidence scores.

Estimating Sequence Quality

When using AttnPacker to design sequence and rotamers jointly, we also output sequence confidence scores obtained by taking the average cross entropy between predicted sequence labels and predicted amino acid type probabilities. Concretely, given sequence labels $\{s_i\} \in \{0, \dots 19\}^L$ and residue-type probabilities $\{\boldsymbol{p}_i\}$ with $\boldsymbol{p}_i \in R^{20}$ for each residue i, the sequence score is defined as

$$\text{seq-score} = -\frac{1}{L}\sum_i \log p_i[s_i]. \tag{2.15}$$

where $p_i[s_i]$ is the probability assigned to amino acid type s_i at position i.

Shown in Figure 2.14, we find a strong Spearman correlation between sequence score and native sequence recovery ($|\rho| = 0.90$) and between predicted plDDT and sequence recovery ($|\rho| = 0.83$).

2.2.6 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 [Hir+20]). 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. The loss is computed as

$$L^{(i)} = \text{mean}\{\text{huber}\left(\vec{\boldsymbol{x}}_i^a - \vec{\boldsymbol{x}}_i^{a,*}\right) : a \in \mathcal{A}(i)\}, \tag{2.16}$$

where $\mathcal{A}(i)$ denotes the set of atoms in the side chain of residue i and $\text{huber}\,(x, y, \beta)$ 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 (2.16) since some residues have symmetric side-chains. 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 side-chains and pairs of atoms for which we swap coordinates can be found in Table 2.5.

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 2.5: **Symmetric side-chains**. Amino acids with side-chain symmetries and the atom pairs which constitute these symmetries.

The overall loss function is given by

$$\mathcal{L} = 1 \cdot \mathcal{L}_{\text{coord}} + 0.2 \cdot \mathcal{L}_{\text{dist}} + 0.15 \cdot \mathcal{L}_{\text{plDDT}} + 0.15 \cdot \mathcal{L}_{\text{seq}}. \tag{2.17}$$

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

2.2.7 Rotamer and Clash Optimization

As our method directly predicts side-chain atom coordinates, there is no guarantee that bond lengths and angles between predicted atoms will be physically realistic. To avoid limitations inherent in discrete rotamer sampling, we develop a continuous mapping from dihedrals to rotamers and from atomic distances to steric clashes. This enables us to use gradient descent in optimizing our conformations while also providing guarantees of idealized rotamer geometry. Although our objective function is highly non-convex, we show empirically that proper rotamer initialization results in a minimal increase in per-residue RMSD.

Rotamer Post-Processing

In Algorithm 7, $\text{vdW}\left(i, j, a, a^{'}\right)$ 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 2.6. 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 [Jum+21, Supplementary Information, Algorithm 24]. In converting to 3D coordinates, we use idealized bond lengths and angles for each residue type, taken from OpenFold [Ahd+22]. We also use code from OpenFold to compute side-chain dihedral angles. At the start of post-processing,

Algorithm 7 Post-Process Predictions

Input
  $\vec{\boldsymbol{x}}_i^a$: All-atom coordinates for each residue, i and atom a in $\mathbb{R}^3$
  $\chi_i^{(t)}$: Side chain dihedrals for each residue i, and $t = 1, 2, 3, 4$
  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($\vec{\boldsymbol{x}}_i, \chi_i^{(t)}, s_i, t_{\text{vdw}}$) :
  $\vec{\boldsymbol{y}}_i^a = \text{TOALLATOMCOORDINATES}\left(\vec{\boldsymbol{x}}_i^{\{N,C\alpha,C\}}, \chi_i^{(t)}, \boldsymbol{S}_i\right)$
  $d_{i,j}^{a,a'} = \left\| \vec{\boldsymbol{y}}_i^a - \vec{\boldsymbol{x}}_j^{a'} \right\|_2$
  $\mathcal{L}_{\text{RMS}} = \text{mean}_{i,a}\left(d_{i,i}^{a,a}\right)$
  $\mathcal{L}_{\text{steric}} = \sum_{i,j,a,a'} \text{ReLU}\left(\text{vdW}\left(i,j,a,a'\right) \cdot t_{\text{vdw}} - d_{i,j}^{a,a'}\right)$
  return $\mathcal{L}_{\text{RMS}}$, $\mathcal{L}_{\text{steric}}$

we initialize each dihedral according to our coordinate predictions and update dihedrals $\chi_i^{(t)}$ 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.

Running Time and Effect on Per-Residue RMSD

We find that side-chain RMSD increases only slightly after post-processing (see Figure 2.5A). Upon visually inspecting the raw coordinates output from our method, we found that most 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 2.5B). 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

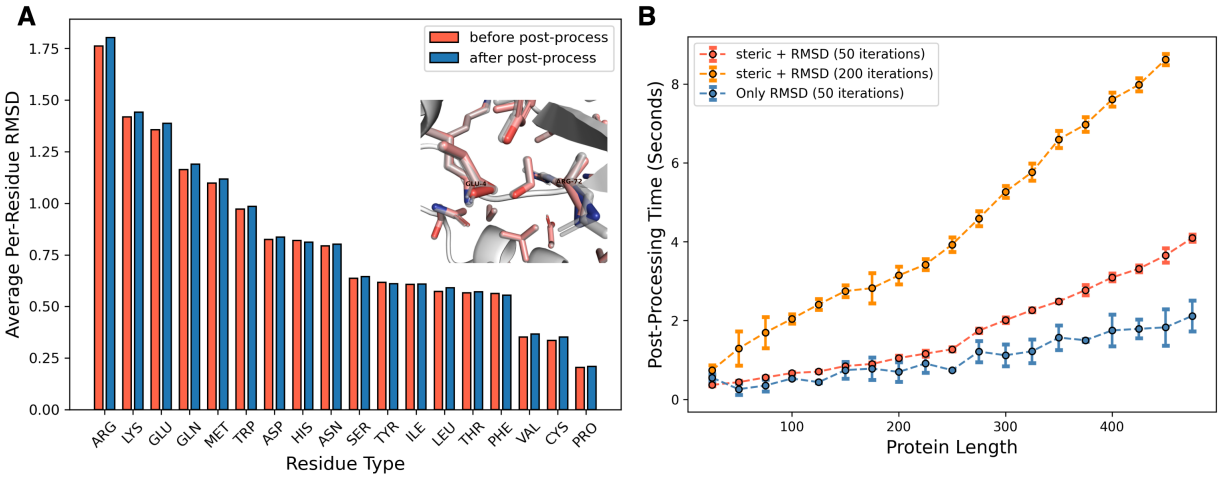

Figure 2.5: **Post-processing procedure**. (**A**) Bar plot of average RMSD of each residue type before (red) and after (blue) rotamer and steric clash optimization. We also add a small illustration of side-chain conformations before (red) and after (blue) post-processing on CASP target T1043. (**A**) 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 time taken to run 50 LBFGS [Byr+95] 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.

post-processing procedure requires only a PDB file as input and is broadly applicable for a range of other tasks.

2.2.8 Training Details

We trained and validated all models using the BC40 data set[1]. The data set contains roughly 39k proteins which are selected from the PDB database by 40% sequence identity cutoff. We filtered entries from this dataset so that no single target had greater than 40% sequence identity with any target in our test set, and we used a 90-10 split for training and validation. We train our models for ten epochs using the Adam [KB15] optimizer with default settings and a learning rate of 10^{-3}. To save memory, we crop all training examples to at most 400 residues by randomly selecting a single contiguous region. A complete summary of training data, optimization settings, and training procedure is given in Appendix A.2.

1. BC40 is publicly available at: https://drug.ai.tencent.com/protein/bc40/download.html

2.2.9 Test Datasets

Native Backbones Our native backbone data consists of protein backbones downloaded from the 13^{th} and 14^{th} Critical Assessment of Techniques for Protein Structure Prediction (CASP) database. This includes 82 regular targets in CASP13 and 64 regular targets in CASP14 (see Table A.5 for a full list of targets). We also consider CASP13 and CASP14 free modeling targets and provide results in Table A.2. We chose to use the CASP13 and CASP14 test sets as there is no canonical training or validation sets for PSCP, and in most cases, these test sets have little overlap with the training sets used for the methods in comparison.

Predicted (Non-Native) Backbones We also generated a set of non-native backbone structures for the CASP13 and CASP14 targets, using AlphaFold2 from ColabFold [Mir+22]. We selected all non-native backbones with (1) at most 600 residues and (2) $C\alpha$ RMSD less than 2.5 Å from native. The rationale for this decision is given in Section 2.3.3. More details on data generation are given in Appendix A.1.

2.3 Results

Results for side-chain packing are given in Section 2.3.3, and sequence design in Section 2.3.4. We evaluate the quality of our confidence predictions in Section 2.3.5, and postpone ablation studies for Section 2.4.

2.3.1 Overview

When reporting results for our method, we distinguish between the model trained only for PSCP (AttnPacker) and the model trained for PSCP and fixed-backbone design (AttnPacker+Design). When assessing RMSD and dihedral prediction accuracy, the design variant of AttnPacker was provided with complete sequence information as input. When

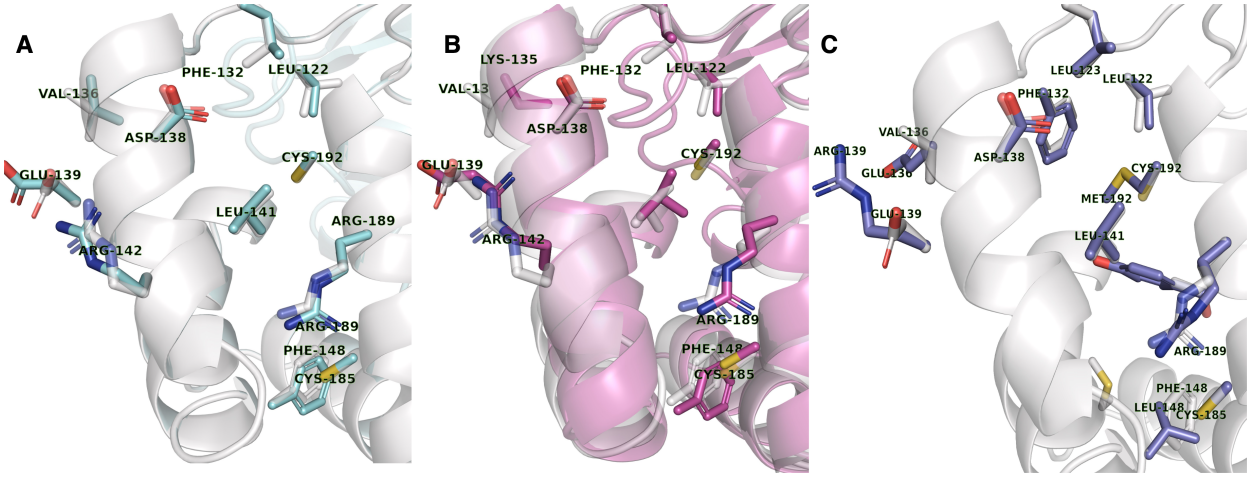

Figure 2.6: **Examples of Side-Chain Packing and co-design on native and non-native backbones.** Native backbones are shown in what using cartoon format. Native side-chains are shown in white using sticks. (**A**) Illustration of correct and incorrect side-chain predictions (cyan) from AttnPacker for 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.

designing a sequence, we do not give this variant any sequence information unless otherwise specified. We use the same hyperparameters for each model, shown in Appendix A.2. We also remark that all results for our methods are taken after post-processing, according to the procedure described in Section 2B.5.

Side-Chain Packing

When native backbones are given, we compare our method against several popular PSCP methods: DLPacker, RosettaPacker, SCWRL4, and FASPR. We use bold fonts and underlining in each table to mark the best and second-best performers in each category, respectively. For non-native backbones, we compare our method with DLPacker, RosettaPacker, RosettaFold, Omegafold, and AlphaFold2. Results for AlphaFold2 are split further by information provided at inference time. We consider MSA and backbone templates as input (AF2+MSA+Temp.), and without MSA, given only backbone templates (AF2+Temp.). Results for RosettaFold are included only for the CASP14 targets. Complete details on data collection are provided in Appendix A.1

Additional results for side-chain packing are given in Appendix A.4. Ablation studies and a detailed analysis of different architectural components can be found in Section 2.4.

2.3.2 Evaluation Criteria

Side Chain Packing

Per-Residue RMSD and Dihedral MAE We consider residue-level RMSD and dihedral angle deviations between predicted and native rotamers. Residue RMSD is calculated over non-hydrogen side chain atoms and excludes residues Ala and Gly. Overall RMSD is computed by averaging over all residues in all proteins for a dataset. For a side-chain dihedral χ_i, mean absolute error (MAE) is calculated analogously to RMSD. Some residue types have bimodal MAE distributions, and the average MAE tends to be much lower than the median. We also report dihedral accuracy to better illustrate instances where entire side-chains are correctly predicted. Overall accuracy (χ_{1-4} Acc.) is defined as the fraction of residues having all dihedral angles within 20° of the corresponding native angles. The accuracy of a fixed dihedral χ_i is defined analogously over all residues containing this angle. As pointed out by Zhang et al. [HPZ20], the prediction accuracy of side-chain dihedrals is much sharper when all side-chain dihedrals are considered, and a 20° cutoff, and hence, we opt to use this criterion for all reported accuracy scores. Following [HPZ20], in computing RMSD and angle MAE statistics, the symmetry of residues Asp, Glu, Phe, Arg, and Tyr were considered. We also consider flipping Asn, Gln, and His terminal groups when computing RMSD due to difficulties distinguishing these atoms [Cao+10]. A complete list of symmetries can be found in Table 2.5.

Steric Clashes To assess the quality of packed solutions, we also consider the average number of steric clashes in packing solutions. Two atoms are considered to clash if their distance is smaller than a fixed percentage of the sum of their van der Waals (vdW) radii, taken from the AMBER force field [Wan+04]. For this metric, we consider only atom pairs

separated by at least four bonds, where at least one atom belongs to a residue side chain. To account for the fact that atoms sharing a hydrogen bond can favorably approach distances within the sum of their vdW radii [LN98] by reducing the clash cutoff by 0.4Å for atom pairs comprised of donor-borne hydrogen and acceptor (see Table 2.6 donors and acceptors).

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 2.6: 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

Sequence Design

Sequence Recovery and Perplexity Inverse folding methods are notoriously difficult to benchmark. They are traditionally evaluated using (i) native sequence recovery (NSR) rate, and (ii) model perplexity (when applicable). 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. [Jin+20], NSR is reported as the median (over all structures) of the average percentage of residues correctly recovered.

***In Silico* Evaluation** While experimental validation remains the gold-standard for evaluating designed proteins, recent works [Tri+23; Wat+22; Wan+21] have used protein structure prediction methods as an *in silico* proxy. Following [Tri+23], we use self-consistency TM-score (scTM) and predicted lDDT (plDDT) to assess our designs. scTM indicates how well sequences encode structure by measuring the TM-score [ZS04] between predicted and ground-truth backbones. Predicted lDDT indicates how confidently a sequence folds to the predicted structure. Each metric is computed using ESMFold [Lin+22].

Co-Design It is difficult to evaluate sequence and rotamer co-designs. In particular, predicted rotamers will be incompatible with the native when the residue types do not match. Nevertheless, we report the Rosetta energy of our co-designs and compare that with the energy of the native structure (sequence and conformation). This experiment is meant to assess how well our designs agree with physics based energy functions, using the the native sequence and rotamer conformations as a baseline. In theory, the native sequence should yield the lowest energy for a given conformation, though we find that our designs yield significantly lower energy on average. We report these results alongside sequence design in Section 2.3.4.

Mutation Effect Prediction Besides predicting amino acid identities, inverse folding models have also been used to predict the effect of point mutations on protein stability, and other criteria [Ing+19; Hsu+22; YZY22]. To quantify our model's ability to predict protein mutation effects, we use the log ratios of the wild-type amino acid and the mutated amino acid at the mutated index i. Formally, for a protein sequence $S = s_1, \dots s_n$ and backbone conformation C, we compare the log probability ratio of mutation x_{mutant} appearing at sequence position i against the wild-type $x_{wild-type}$ while conditioning on the backbone conformation C. and S_{-i}; the identities of all amino acids aside form s_i. The calculation is shown in Equation (2.18).

$$\log p(s_i = x_{mutant} | S_{-i}, C) - \log p(s_i = x_{wild-type} | S_{-i}, C) \tag{2.18}$$

Definition of Core and Surface Residues

In some instances, we divide our results based on *residue centrality*; the number of $C\beta$ atoms within a 10Å ball of the query residue's $C\beta$ atom. For this measure, $C\beta$ atoms from the native conformation are used. *Core residues* are defined as those amino acids with a centrality of at least 20, and *surface residues* are those with at most 15 $C\beta$ atoms within

the same region of interest.

2.3.3 Side Chain Packing

We begin by comparing average RMSD and dihedral accuracy over CASP13 and CASP14 test sets. Additional results for CASP free-modeling targets can be found in Table A.2. Example side-chain packings are shown in Figures 2.6 and 2.7.

Native Backbones

As shown in Table 2.7, our method (AttnPacker and AttnPacker+Design) consistently achieves the lowest RMSD in each centrality category on both datasets. The performance carries over into dihedral prediction accuracy, where our method also achieves top performance for both datasets, regardless of residue centrality. Notably, physics-based RosettaPacker performs well on core residues, while its accuracy for surface residues ultimately hinders its overall performance. Compared to the deep learning method DLPacker, we obtain notably lower RMSD scores in all centrality categories, with the largest improvement on surface residues. We also improve overall dihedral accuracy over DLPacker by more than 3% for each test set.

Focusing on dihedral angle MAE across $\chi_1 - \chi_4$ degrees of freedom, we see that our method achieves top-1 performance on both CASP13 and CASP14 native backbones targets, and top-1 or top-2 performance on CASP14 native backbones. In line with the results reported by Misiura et al. [Mis+21], when compared to traditional methods, deep learning methods recover χ_1 dihedral angles considerably closer to those of the native structure. Compared to the next best non-deep-learning method, we improve average χ_1 MAE by 28% and 22% for CASP13 and CASP14 targets. Unlike DLPacker, this improvement carries over to χ_2 angle prediction, where we obtain 18% and 17% improvements over the next best non-deep-learning method on CASP13 and CASP14 targets.

As shown in Table 2.8, AttnPacker also produces much fewer steric clashes and is the

Method	RMSD (Å)↓			χ−MAE°↓			
	All	Surface	Core	χ_1	χ_2	χ_3	χ_4
				CASP13			
SCWRL	0.934	1.200	0.597	27.64	28.97	49.57	61.54
FASPR	0.910	1.167	0.604	27.04	28.41	50.30	60.89
RosettaPacker	0.872	1.171	0.509	25.88	28.25	48.13	59.82
DL-Packer	0.772	1.018	0.483	22.18	27.00	51.22	70.04
TFN-Transformer	0.738	0.968	0.470	20.18	25.50	46.32	60.64
AttnPacker	**0.669**	**0.881**	**0.414**	18.92	**23.17**	**44.89**	**58.98**
+Design	0.673	0.887	0.424	**18.77**	23.44	46.12	59.57
Count	19118	6046	8880	19118	14333	4887	2268
				CASP14			
SCWRL	1.062	1.331	0.677	33.50	33.05	51.61	55.28
FASPR	1.048	1.304	0.696	33.04	32.49	50.15	54.82
RosettaPacker	1.006	1.384	0.716	31.79	33.00	50.54	56.16
DL-Packer	0.929	1.197	0.562	29.01	31.69	53.98	72.88
TFN-Transformer	0.895	1.139	0.544	27.18	29.13	49.73	51.97
AttnPacker	0.823	1.067	0.502	25.34	28.19	**48.77**	**51.92**
+Design	**0.815**	**1.058**	**0.466**	**24.75**	**27.56**	48.96	55.06
Count	17693	4979	8476	17693	13588	4613	2194

Table 2.7: Overall RMSD and $\chi_1 - \chi_4$ MAE results for the CASP13 and CASP14 targets while native backbones are given. For RMSD, columns divide results by residue degree centrality (All, Core, and Surface). Residue counts for each category are shown as a separate row for each dataset

only method that generates fewer clashes than what is found in experimental structures. We remark that Rosetta also includes the capability to perform side-chain minimization after packing, which could remove many of the clashes in RosettaPacker's output, but we did not perform this as a post-processing step.

	χ_{1-4}Acc. (%)↑			Clash↑	
Method	All	Surface	Core	$r = 0.9$	$r = 0.8$
			CASP13		
SCWRL	56.2%	45.2%	71.2%	20.6	4.6
FASPR	56.4%	45.5%	70.3%	23.3	5.6
RosettaPacker	58.6%	45.9%	75.3%	10.3	2.6
DL-Packer	58.8%	47.3%	73.9%	7.3	2.0
TFN-Transformer	60.1%	47.4%	73.7%	1.2*	0.3*
AttnPacker	**62.1%**	**51.5%**	**75.9%**	1.4*	0.3*
+Design	61.2%	50.7%	75.5%	1.2*	0.2*
Count	19118	6046	8880		
			CASP14		
SCWRL	45.4%	35.1%	62.3%	24.6	6.5
FASPR	46.3%	36.3%	62.3%	29.5	8.7
RosettaPacker	47.5%	35.3%	66.1%	17.1	5.2
DL-Packer	48.0%	36.3%	66.8%	10.7	3.0
TFN-Transformer	50.0%	38.7%	67.8%	2.8*	0.9*
AttnPacker	50.9%	39.1%	68.2%	3.0*	1.0*
+Design	**51.6%**	**39.8%**	**70.6%**	2.5*	0.8*
Count	17693	4979	8476		

Table 2.8: χ_{1-4} accuracy and Clash Results for the CASP13 and CASP14 targets while native backbones are given. For accuracy, columns divide results by residue degree centrality (All, Core, and Surface). Residue counts for each category are shown as a separate row for each dataset. The average number of clashes for each target using a 90% and 80% fraction of the van der Waals radius is given in the rightmost columns. An asterisk indicates an average clash value below that of native structures; 5.9, and 0.4 for CASP13 and 7.9, and 2.5 for CASP14. Results for AttnPacker+Design are obtained with the native sequence provided.

Non-Native Backbones

Comparing PSCP methods directly to protein structure prediction methods is difficult because the predicted tertiary structure can deviate far from that of the native. Side-chain MAE and RMSD statistics also lose interpretability as the predicted backbone structure deviates from the ground truth and native contacts are not conserved. Huang et al. [HPZ20] noticed that χ_{1-4} recovery rates decrease significantly for non-native side chain packing when

backbone RMSD is larger than 2.38Å. Cao et al. also find a strong correlation between *de novo* designability and backbone $C\alpha$ RMSD in protein binder design. In an attempt to fairly compare structure prediction methods to PSCP methods, we restrict to predicted backbones with $C\alpha$ RMSD at most 2.5Å from native in the main text, and include a comparison of side-chain RMSD and χ_1 MAE against a broader range of backbone RMSDs in Figure 2.8. Since non-native backbones can deviate from ground truth, we compute a transformation to align backbone heavy atoms N, $C\alpha$, and C for each residue in the predicted structure to the corresponding residue in the native structure. After applying this per-residue transformation, we can compute side-chain RMSD and dihedral MAE as usual.

As a consequence of choosing low RMSD backbones, we remark that the comparison may be biased in favor of PSP methods since we consider only those targets for which AlphaFold2 already produces accurate backbone conformations. Furthermore, the results may be biased against OmegaFold and RosettaFold, since target selection was based exclusively on AlphaFold2 accuracy.

For simplicity, we exclude FASPR and SCWRL and include only the top-performing traditional method RosettaPacker. As shown in Table 2.9, discarding MSA information from AlphaFold2 input causes a drastic reduction in performance. RosettaFold (with MSA) performs slightly better than AlphaFold2 (without MSA), having a slightly higher average RMSD but better dihedral accuracy. The relatively poor performance of these methods is likely attributed to inaccurate backbone predictions, where the average RMSDs are 13Å and 15 Å for RosettaFold and AF2+Temp, respectively. It is likely that MSA information also contributes to AlphaFold2's success on CASP14 targets, where the nearly identical architecture used in OmegaFold falls short in terms of dihedral accuracy. Part of this reduction can be attributed to OmegaFold predicting more accurate backbone conformations for the CASP13 targets, with an average RMSD of 4Å increasing to 6Å for the CASP14 targets. The two methods perform similarly in terms of χ_1 MAE and RMSD on both data sets, with OmegaFold having a slight edge in terms of χ_1 prediction, despite predicting backbone

	RMSD↓	MAE° ↓	Acc.↑	Clash (%VdW)↓		
	All	χ_1	χ_{1-4}	100%	90%	80%
Method	**CASP13 Non-Native (N=7490)**					
AF2+Temp.	1.401	45.00	32.9%	80.0	30.5	13.2
AF2+MSA+Temp.	0.944	30.12	54.8%	39.7	1.1*	0.0*
OmegaFold	0.926	28.15	54.6%	52.3	10.7	2.8
RosettaPacker	0.953	30.16	54.6%	45.9	3.9	0.6
DLPacker	0.925	29.56	53.3%	39.4	3.3	0.4*
AttnPacker	**0.871**	28.00	**55.0%**	28.8*	0.4*	0.0*
+Design	0.876	**27.87**	54.6%	30.7*	0.4*	0.0*
	CASP14 Non-Native (N=4327)					
AF2+Temp.	1.431	47.40	28.4%	77.2	29.4	12.3
AF2+MSA+Temp.	0.948	30.36	**52.2%**	38.8*	1.7*	0.0*
OmegaFold	0.949	29.32	49.3%	54.7	10.8	3.0
RosettaFold	1.51	48.45	34.8%	40.6	1.3*	0.0*
RosettaPacker	0.980	30.85	51.2%	46.7	3.3	0.4*
DLPacker	0.955	30.39	49.9%	38.8*	4.4	0.7*
AttnPacker	0.902	29.14	51.2%	30.8*	0.44*	0.0*
+Design	**0.884**	**28.18**	51.1%	32.6*	0.48*	0.0*

Table 2.9: Results for packing non-native backbone structures predicted by AlphaFold2. Values are derived from a total of 47 CASP13 and 27 CASP14 targets, all having at most 600 residues and $C\alpha$ RMSD less than 2.5Å from native. An asterisk indicates an average clash value below that of corresponding native structures - 34.6, 2.2, and 0.5 for CASP13 and 40.0, 2.7, and 0.7 for CASP14. Results for AttnPacker+Design are obtained with the native sequence provided.

conformations with higher RMSD on average.

For PSCP methods, AttnPacker achieves top performance in terms of RMSD and χ_1 MAE, with significantly lower overall RMSD and a smaller improvement in χ_1 MAE over OmegaFold. For several targets, AttnPacker produces packings with nearly 30% lower RMSD than AlphaFold2, an example of which is given in Figure 2.7B. For non-native backbones, RosettaPacker is more competitive with our method, having dihedral accuracy within a percentage point of the best score on both test sets and a slightly smaller gap in terms of

RMSD.

Overall, we see fewer steric clashes on non-native backbones than on native backbones. Looking into this, we found a correlation between protein length and the average number of clashes. This difference is likely attributed to non-native backbones being restricted to less than 600 amino acids in length.

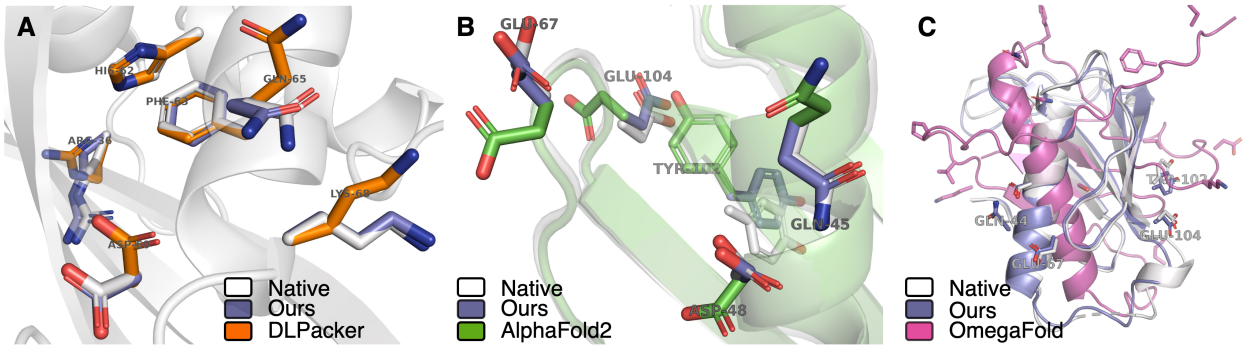

Figure 2.7: **Example backbone packing where AttnPacker yields correct results.** (**A**) The backbone of CASP13 target T0968s1 is shown in cartoon representation, and selected side chains are shown using stick representation. Amino acid names and sequence positions are shown for select residues. AttnPacker 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.

OmegaFold and non-MSA-based AlphaFold2 have the highest per-target average. This may contribute to OmegaFold's accuracy in terms of χ_1 MAE and RMSD, as it frequently predicts packings with large amounts of steric hindrance. This highlights the importance of considering steric clashes alongside traditional metrics when analyzing modeling performance. AttnPacker has fewer steric clashes than MSA-based AlphaFold2 and corresponding native structures, indicating that our method is effective at producing physically realistic packings. It is well known that AlphaFold2 structures can contain considerable steric hindrance between side-chain atoms. According to the AlphaFold Structure Database [Var+21], this typically occurs only in low-confidence regions. Since we selected only accurately modeled backbones, these numbers may be biased. This could also explain why the average

number of clashes is much higher when excluding MSA information. We also note that AlphaFold2 is fine-tuned with steric clash loss, likely contributing to lower clash values.

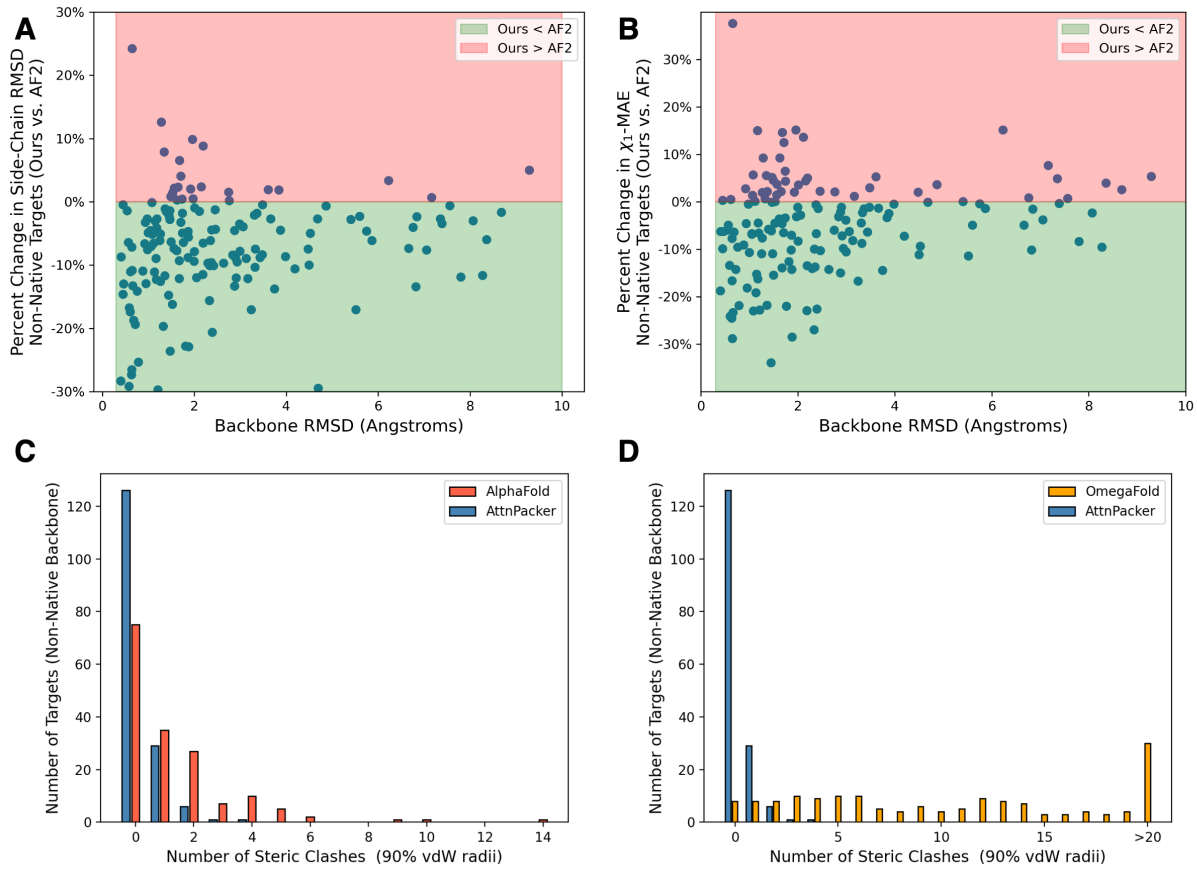

Figure 2.8: (**A**) Per-target scatter plot comparing AttnPacker and AlphaFold2 (AF2) on non-native backbones for CASP13 and CASP14 targets. The x-axis shows backbone $C\alpha$ RMSD between AlphaFold2 predicted backbones and native structures. The y-axis shows the percentage difference in average side-chain 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.

RMSD for Bulky Amino Acids

In terms of residue-level RMSD, the most significant improvements are achieved for Arg and His, each of which have large positively charged side-chains, as well as bulky hydrophobic amino acids Phe, Trp, and Tyr.

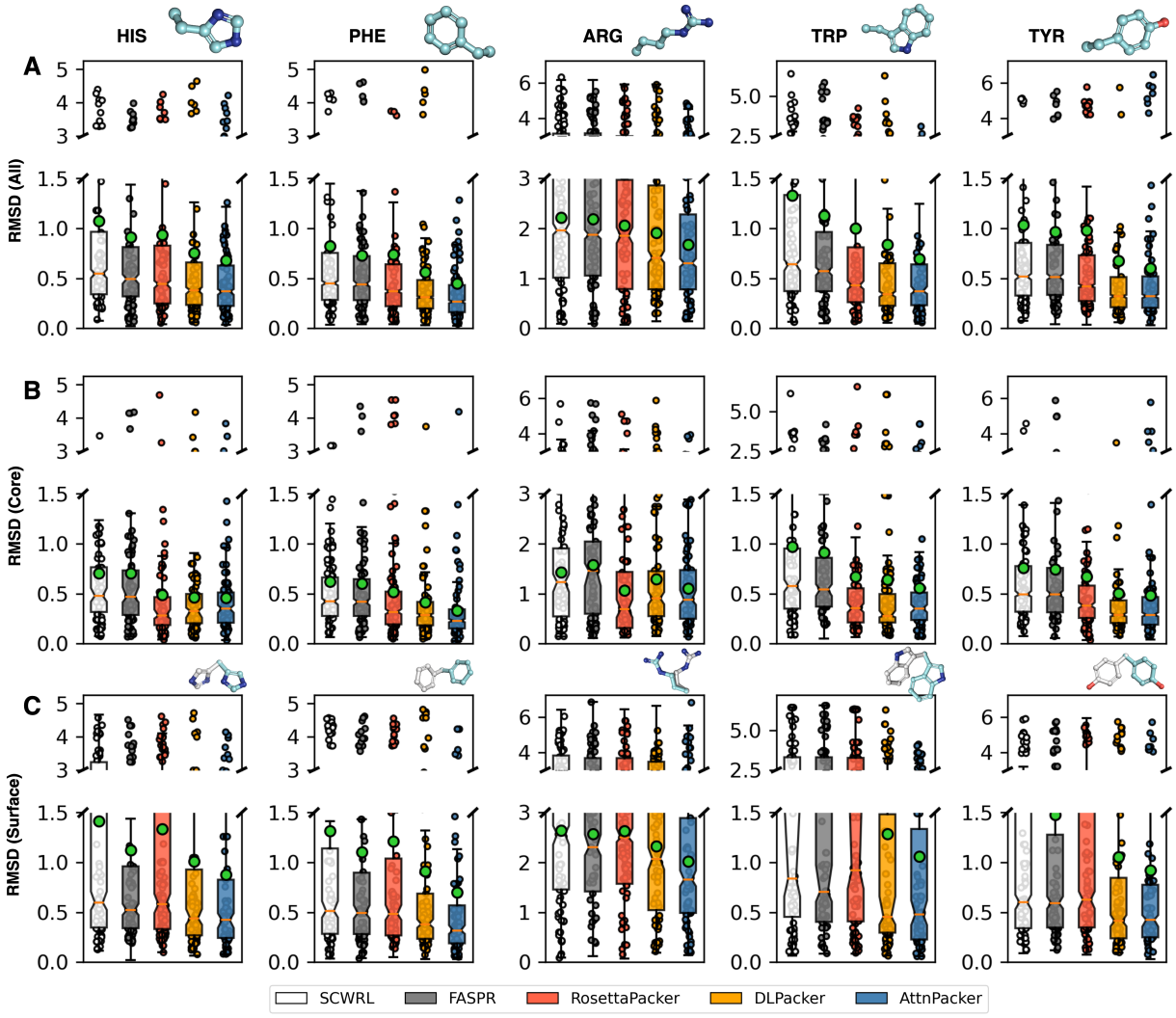

Figure 2.9: **RMSD box plots for bulky amino acids on the CASP13 targets.** Each box plot shows the average RMSD(Å) (y-axis) of each method (x-axis) on His, Phe, Arg, Trp, and Tyr. (**A**) shows RMSD values over all centrality categories, (**B**) restricts to the protein core, and (**C**) to the protein surface. Each box extends from the data's lower to upper quartile values, with an orange line at the median. 95% confidence intervals around the median are shown with a notch, and the mean is shown with a green circle. For each residue type, a random sample of 80 RMSD values is plotted vertically with each bar.

In Figure 2.9, we break the y-axis for all residues except Arg but maintain the split-axis

formatting for consistency. Aside from Arg, all RMSD distributions have two major peaks around 0.5 Å, corresponding to correct predictions, and around 4-5 Å, corresponding to mirrored conformations.

As shown in Figure 3A, AttnPacker improves the overall RMSD of each amino acid type by 10%-20%, In Figure 2.9C, we show that part of this reduction can be attributed to more accurate modeling of surface residues, where deep learning methods outperform traditional approaches by 20% for these residue types. In the protein core, deep learning methods lose their edge on His, Arg, and Trp. Incorrect modeling of bulky amino acids in the protein core could cause adverse effects on neighboring amino acids, leading to inaccurate downstream predictions. The density of atoms in the protein core also provides more packing constraints, which is favorable for traditional energy-based methods like RosettaPacker. A complete overview of residue-level RMSD on the CASP13 and CASP14 targets is given in Tables A.3 and A.4.

Dihedral Accuracy Large Amino Acids with High Degrees of Freedom

To better understand the instances where our method loses its advantage to traditional PSCP algorithms, we evaluate dihedral predictions on four large amino acids with high side-chain dihedral degrees of freedom. We consider dihedral prediction for charged and polar amino acids Lys, Arg, Glu, and Gln in Table 2.10.

For these residue types, we restrict the comparison to the top-performing classical method RosettaPacker. For these amino acids, our method is still competitive with RosettaPacker, achieving top performance in $\chi_1 - \chi_4$ MAE for all but χ_3 of Lys. Surprisingly, low dihedral MAE does not always translate to high dihedral accuracy. Although AttnPacker almost always obtains lower MAE and RMSD values (see Tables A.3 and A.4) than RosettaPacker, the physics-based method, achieves top-1 χ_{1-4} accuracy for both Arg and Lys. In Figure 2.10, we consider accuracy conditioned on average B-factor and show that inaccuracies in 3D models are unlikely to be the cause of this discrepancy. We hypothesize that this

Method	MAE° ↓				Acc.(%)↑	
	χ_1	χ_2	χ_3	χ_4	χ_{1-2}	χ_{1-4}
				Arg		
Ros. Pack.	32.0	32.21	59.3	63.3	**52.5%**	**22.2%**
DLPack.	28.6	32.5	61.1	67.2	49.1%	14.5%
AttnPack.	**26.2**	**30.1**	**57.0**	**61.8**	50.5%	13.3%
				Lys		
Ros. Pack.	31.3	37.9	**45.0**	56.5	51.5%	**21.6%**
DLPack.	27.6	37.6	51.8	72.7	46.1%	9.6%
AttnPack.	**24.1**	**33.8**	45.9	**56.3**	**54.2%**	16.0%
				Gln		
Ros. Pack.	32.2	42.4	54.0		50.9%	29.7%
DLPack.	30.5	39.9	50.4		51.9%	26.0%
AttnPack.	**24.8**	**34.6**	**43.0**		**56.3%**	**34.0%**
				Glu		
Ros. Pack.	38.9	43.7	31.1		43.9%	24.9%
DLPack.	34.6	42.4	36.5		42.9%	22.0%
AttnPack.	**29.0**	**38.1**	**27.9**		**49.2%**	**30.1%**

Table 2.10: **χ-Dihedral MAE° and accuracy on the CASP13 targets for charged and polar Amino Acids Arg, Lys, Gln, and Glu.**

is a side-effect of using RMSD as a training objective, which places more importance on accurately predicting lower-order dihedrals. This suggests that new training methods or loss functions which improve higher-order dihedral accuracy are important areas for future research.

2.3.4 Sequence Design

We find that AttnPacker's sequence designs strongly encode native backbones, achieving native-like scores for both scTM and plDDT metrics (Figure 2.11(B and C)). Extended *in silico* analysis of AttnPacker and ProteinMPNN is provided in Section 2.3.4. Results for native sequence recovery and perplexity in various design settings are given in Tables 2.11 and 2.12.

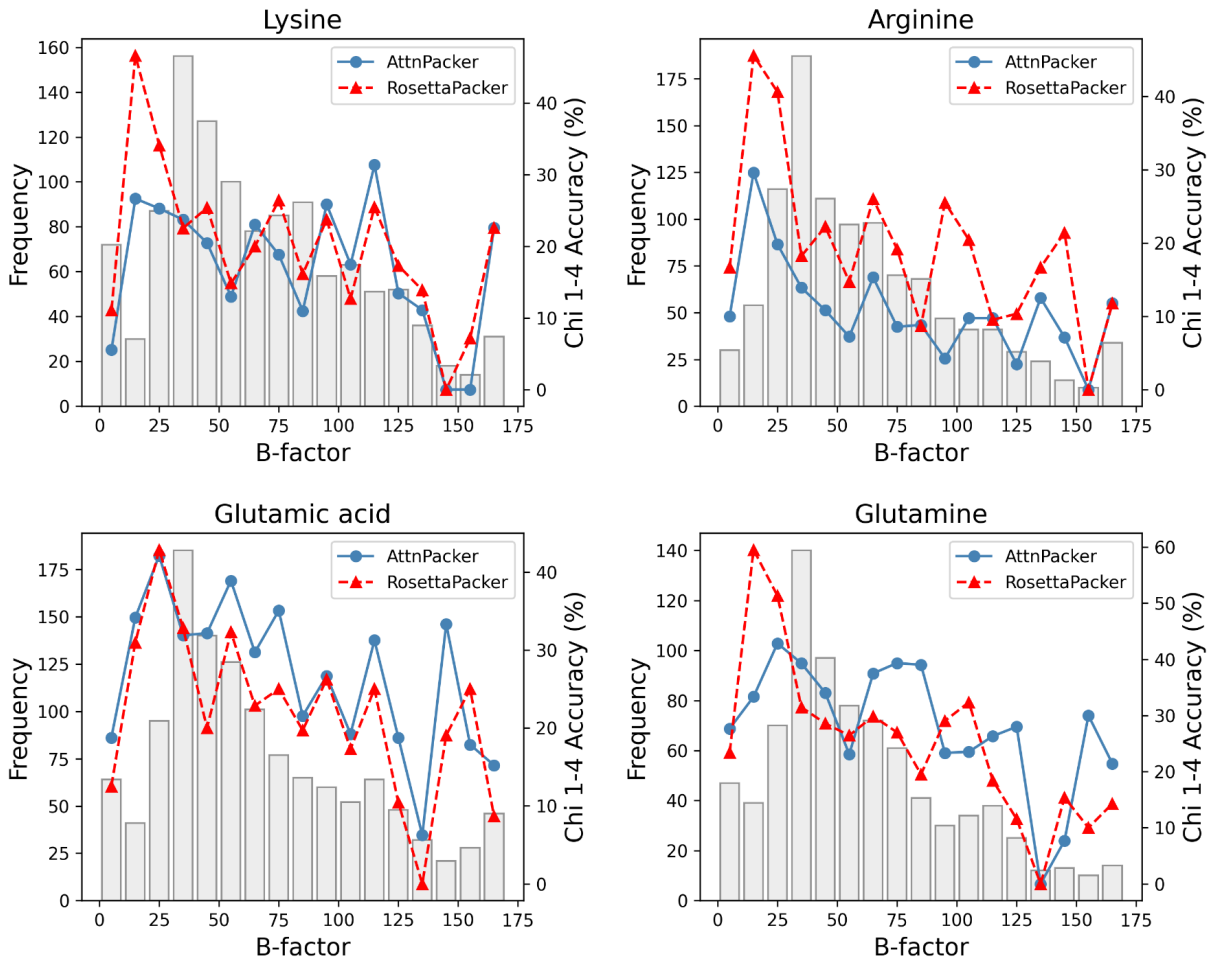

Figure 2.10: χ_{1-4} **accuracy conditioned on average side-chain atom B-factor for CASP13 targets**. Dihedral accuracy for AttnPacker (blue) and RosettaPacker (red) are shown for four amino acid types. Gray bars indicate the frequency of the corresponding residue type and b-factor bin, as indicated 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.

Co-Design

In Figure 2.11A, we evaluate AttnPacker's ability to co-design sequence and rotamers by considering Rosetta energy values. 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 2.6.

Figure 2.11A shows the energy of AttnPacker fixed-backbone designs and Rosetta gener-

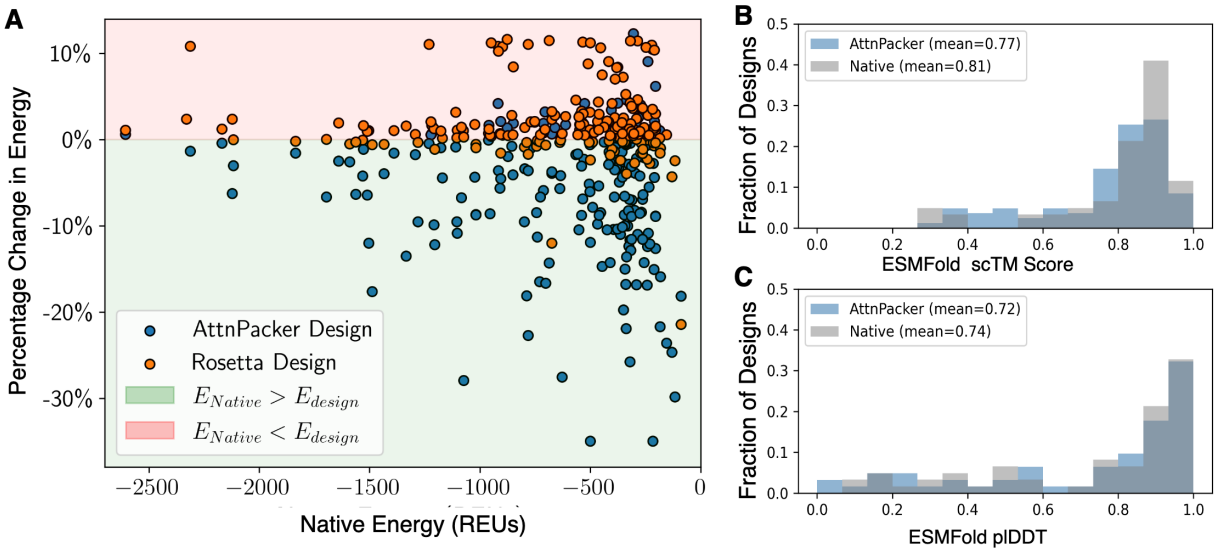

Figure 2.11: **In silico evaluation of AttnPacker generated sequence designs for CASP13 targets.** (**A**) Scatter Plot of Rosetta energy for Rosetta-relaxed native structures (x-axis) against percentage change in Rosetta energy for designed sequences and side-chain conformations (y-axis). Orange dots correspond to Rosetta designs, and blue dots correspond to designs with AttnPacker. Energy scores were computed after running Rosetta's FastRelax protocol (outlined in Appendix A.1).Plots on the right show ESMFold scTM (**B**) and plDDT scores (**C**) for native (gray) and AttnPacker-generated (blue) sequences.

ated designs using corresponding native proteins from the CASP13 and CASP14 test set as a baseline. Rosetta designs were obtained using the FastDesign protocol [CLG10; Lem+20], and energy was calculated with the Rosetta *ref_2015* energy function [Alf+17]. A detailed overview of data collection for this process is provided in Appendix A.1. We remark that other programs have been developed for this task, e.g. Osprey [Gai+13], protCAD [SD02], and SCADS [FAK07] but they lack documentation and have not been rigorously benchmarked. On average, designs from AttnPacker yield lower energy than both native structures and Rosetta designs, indicating that our method captures biologically meaningful aspects of sequence-structure compatibility.

In Table 2.11, 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

	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 2.11: **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 each subset as a fixed percentage of the protein's length, using values 10%,25% and 50%.

Results in Table 2.11 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 [Ing+19]. 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.

In Table 2.12, 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

Method	NSR ↑		Perplexity ↓	
	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 2.12: **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 reported by Jing et al.[Jin+20] but differ on the training set used - CATH4.2, and BC40, respectively.

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 detailed in Appendix A.1.

As shown in Table 2.12, AttnPacker+Design performs similarly 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 could not significantly improve performance, likely because of the well-known over-smoothing issue [CW20] with graph neural networks.

ESMFold Evaluation

To understand how well our generated sequences encode native backbone structure, we used ESMFold [Lin+22] to predict structures from sequence designs for CASP13 and CASP14 targets. In Figure 2.12, 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 [Joh+21], 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. However, 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 ([Dau+22], Supplementary Material, "Further *in silico* analysis"). Our results in Figure 2.12 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).

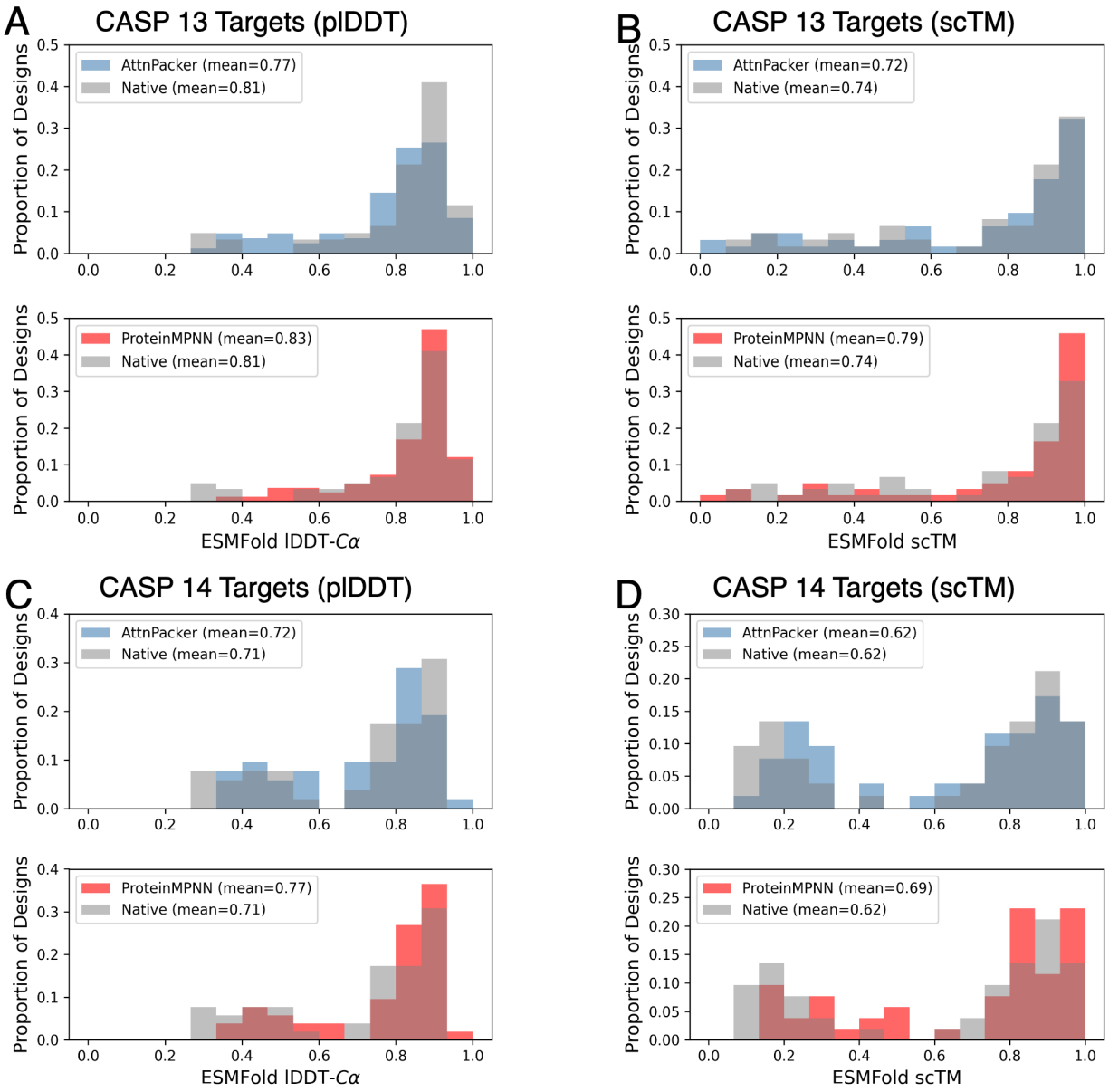

Figure 2.12: **ScTM and plDDT distributions for CASP13 and CASP14 targets.** Distributions for AttnPacker, ProteinMPNN, and native sequences are shown in blue, gray, and red, respectively.

Zero-shot Protein Mutation Effect Prediction

For this task, we trained another variant of AttnPacker using a higher weight for native sequence recovery and a lower weight for side-chain RMSD loss; Equation (2.17) is modified to

$$\mathcal{L} = 0.15 \cdot \mathcal{L}_{\text{coord}} + 0.2 \cdot \mathcal{L}_{\text{dist}} + 0.15 \cdot \mathcal{L}_{\text{plDDT}} + 1 \cdot \mathcal{L}_{\text{seq}}. \tag{2.19}$$

Following Ingraham et al. [Ing+19], we first assess our model's performance on predicting protein stability in response to single-point mutations using deep mutational scanning data from [Roc+17]. In Table 2.13 we observe that conditioning on the partial input sequence (in addition to backbone conformation) leads to improvements in stability prediction for eight of the ten *de novo* designed mini-proteins. Moreover, our model improves over the Structure Transformer for nine of the ten targets.

Topology	$\beta\beta\alpha\beta\beta$				$\alpha\beta\beta\alpha$				$\alpha\alpha$		avg.
ID	37	1498	1702	1716	779	223	726	872	134	138	-
Structure Trans.	0.47	**0.45**	0.12	0.47	0.57	0.36	0.11	0.21	0.24	0.33	0.33
Ours (given S_{-i}, C)	**0.74**	0.43	**0.28**	**0.58**	0.58	0.51	**0.33**	**0.42**	**0.51**	**0.58**	**0.50**
Ours (given C)	0.70	0.39	0.26	0.52	0.48	0.49	**0.33**	**0.42**	**0.51**	0.52	0.46
GVP-GNN	0.53	0.39	0.26	0.57	0.48	0.47	0.19	0.39	0.44	0.44	0.42
GVP-Transformer+AF2	0.70	0.33	0.22	**0.58**	**0.64**	**0.55**	0.26	**0.42**	0.50	**0.58**	0.48

Table 2.13: **Predicted likelihoods correlate with mutation effects in de novo - designed mini proteins**. Pearson Correlation between high-throughput mutation effect data from a systematic design of mini proteins. Each design (column) includes 775 experimentally tested mutant protein sequences. Results are split by fold topology (first row) and design model (second row) as referenced in [Roc+17]. Results for the Structure Transformer are generated on rigid backbones, and taken from Table 5 of [Ing+19]. Results for GVP-GNN and GVP-Transformer+AF2 are taken from [Hsu+22], table C.2.

Moreover, our model outperforms protein language models TAPE [Rao+19] and UniRep [All+19] trained on large sequence databases across 12 DMS datasets in zero-shot protein mutation effect prediction regardless of sequence evolutionary information as shown in Figure 2.13A. This result indicates that AttnPacker captures the underlying functional interaction between the 3-dimensional conformation and the amino acid sequences of a given protein

which suggests structure information may facilitate the characterization of protein mutation effects. We also noticed in Figure 2.13(B) that the performance of our method correlates with the similarity between the DMS target structure and our training set but independent of the sequence identity between the DMS target and our training set. This further confirms that our zero-shot mutation effect prediction is structure-aware.

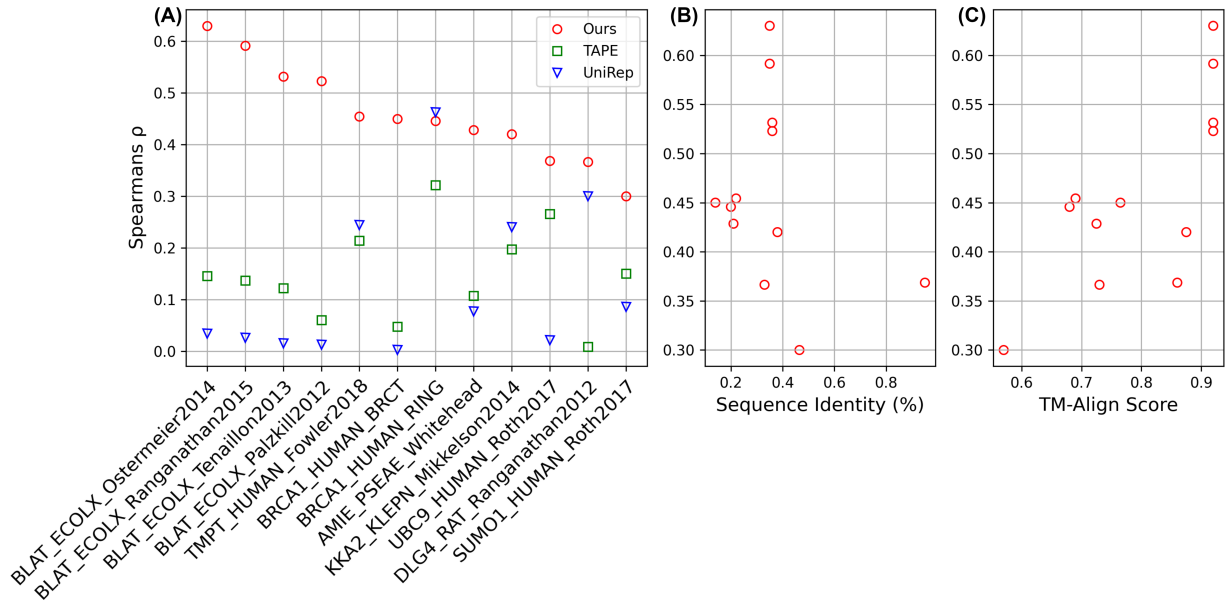

Figure 2.13: **Zero-shot prediction performance** (**A**) Per task performance of our model compared with two other language models, UniRep and TAPE, on 12 DMS datasets measured by rank correlation. (**B**) Performance of our model against the top-1 TM-Align score (right) and sequence identity (left) among the training protein and DMS targets.

2.3.5 Confidence Predictions

Illustrated in Figure 2.14, AttnPacker's side-chain lDDT predictions correlate strongly with ground truth lDDT for both native ($\rho = 0.85$) and non-native ($\rho = 0.92$) input backbones (Figure 2.14D). Furthermore, show that this metric is consistent with per-target side-chain RMSD and χ_1 MAE (Figure 2.14E,F), suggesting that confidence predictions can be used to accurately assess rotamer quality.

Figure 2.14A-C shows how predicted sequence and side-chain lDDT scores can be used to

predict design quality. Interestingly, we see a strong correlation between predicted side-chain lDDT and native sequence recovery (Figure 2.14C). This strongly suggests that the model is able to reason about sequence-structure compatibility.

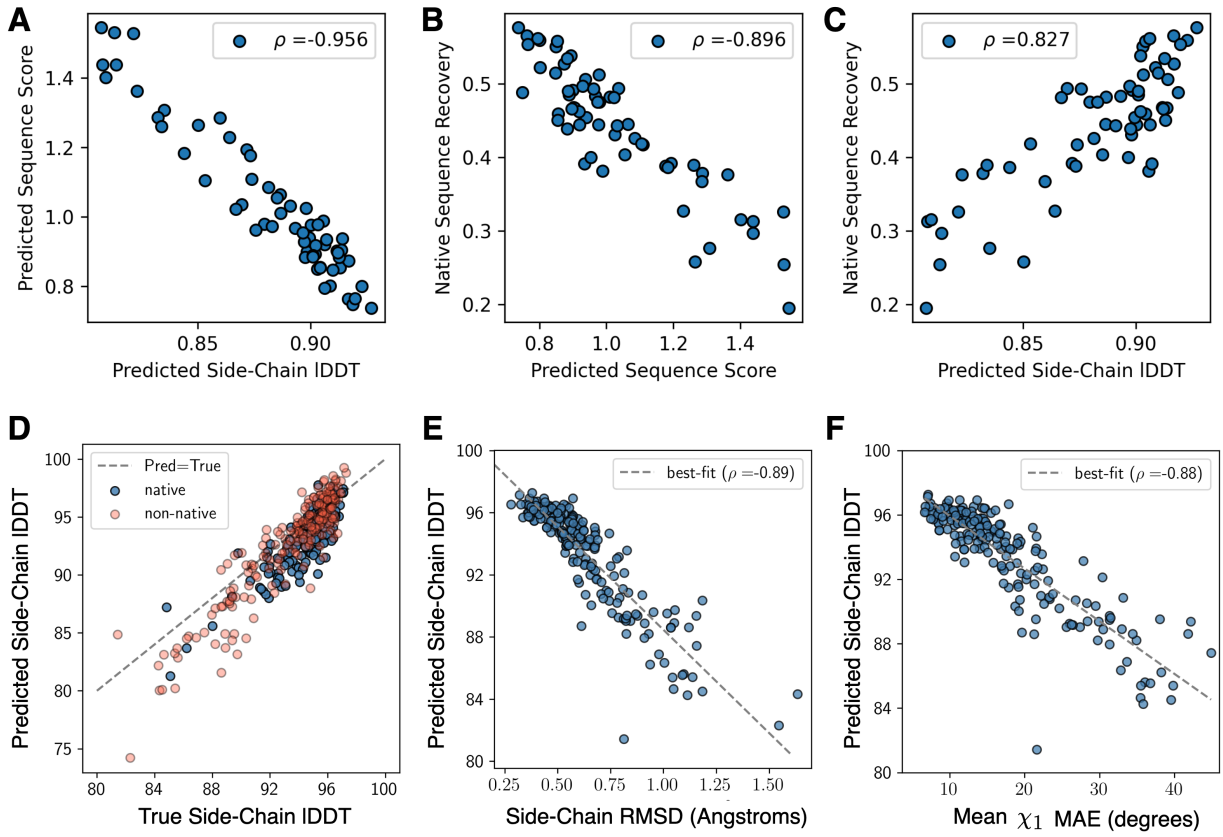

Figure 2.14: **Model Confidence Calibration**. (**A-C**) Scatter plots between predicted side-chain lDDT, sequence score, and native sequence recovery for CASP13 targets. To generate results, AttnPacker+Design was run five times for each target with no sequence information provided. Each metric was averaged over decoys for each target to produce the plots. Plots (**D-E**) were generated using the native sequence as input. (**D**) shows a scatter plot of predicted side-chain lDDT (y-axis) against ground-truth side-chain lDDT for native and non-native CASP targets. Spearman correlations are $\rho = 0.92$ and $\rho = 0.85$ respectively. (**E** and **F**) show predicted side-chain lDDT (y-axis), against average side-chain RMSD and χ_1-MAE for CASP13 and CASP14 native targets. The Design variant of AttnPacker was also used for plots (D-F), but full sequence information was provided.

2.4 Ablation Studies and Architecture Assessment

Much of AttnPacker's success can be attributed to novel architectural components, described in Section 2B. Our stand-alone TFN-Transformer outperforms DLPacker with respect to both overall RMSD and dihedral angle accuracy on native backbones. Integrating local triangle updates boosts accuracy further, especially for core residues, as shown in Figure 2.15, and Table 2.16. Some of this improvement may come from increased depth, which effectively increases the network's receptive field. This is supported by the fact that performance differences are minimal for residues on the protein surface. Compared to full triangle updates, as implemented in AlphaFold2, our local approach roughly matches the performance while considering only the 30 nearest neighbors of each residue. These results and several other ablation studies are shown in Section 2.4. We consider several variants of SE(3)-equivariant

	Similarity
Dot Product	$\sigma_{ij}^{\ell} = \left(q_i^{\ell}\right)^{\top} k_{ij}^{\ell} + b_{ij}^{\ell}$
Distance	$\sigma_{ij}^{\ell} = \begin{cases} \left(q_i^{\ell}\right)^{\top} k_{ij}^{\ell} & \ell = 0 \\ \left\| T_{ii}\left(q_i^{\ell}\right) - T_{ij}\left(k_{ij}^{\ell}\right) \right\|_2^2 & \ell > 0 \end{cases}$
	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 2.14: **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.

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 2.14. Following the conventions of ([Tho+18; Fuc+20]), we use a superscript ℓ to denote type-ℓ features of dimension $2\ell + 1$.

We trained four TFN-Transformer models differing only by attention and similarity type. Hyperparameters were chosen to match those in Table A.1. The average RMSD and dihedral accuracy results of the four attention variants are shown in Table 2.15.

		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 2.15: **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 2.15 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. [Fuc+20], this may be because 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 2.16, 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 Dist.	**0.698**	**0.475**	**0.892**	$\theta = 12$	0.679	0.438	0.888
+ No coord. attn.	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 2.16: **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 2.16.

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.

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

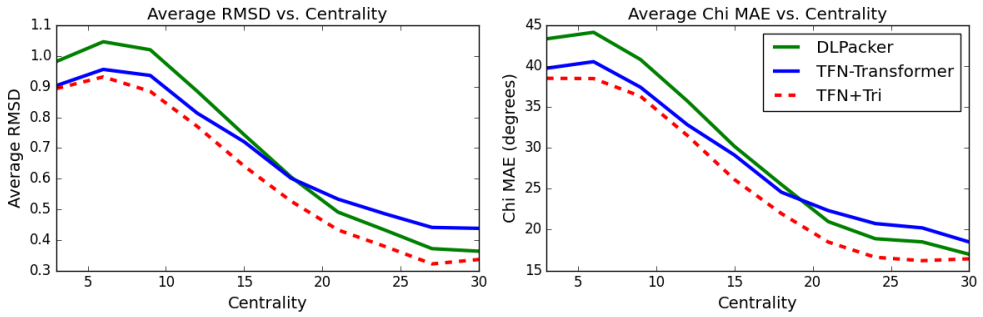

Figure 2.15: **Average RMSD (Å) and χ_{1-4}-MAE° against centrality**. The y-axis shows average RMSD (**left**) and average χ-MAE (**right**) for methods DLPacker, TFN-Transformer (TFN), and AttnPacker against residue centrality (x-axis). Values were computed using all targets in both CASP13 and CASP14 data sets. Values were disregarded if the number of residues with the corresponding centrality was less than 30. We remark that RMSD was computed before performing post-processing for our methods.

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.

2.5 Concluding Discussion

We have developed AttnPacker, an SE(3)-equivariant model for the direct prediction of sequence and side-chain coordinates. AttnPacker uses spatial information derived from protein backbone coordinates to efficiently model residue and pairwise neighborhoods. This, coupled with an SE(3)-equivariant architecture, allows for the simultaneous prediction of all side-chain rotamers without conformational sampling or discrete rotamer selection.

Components of our deep learning model were inspired by AlphaFold2 and the SE(3)-Transformer. By generalizing and carefully evaluating ideas from these architectures, we were

able to achieve comparable or better accuracy while drastically improving efficiency. Specifically, we generalized two components of AlphaFold2 to spatial graphs specific to protein backbones. We also modified the attention heads of the SE(3)-Transformer to incorporate pair bias, distance information between hidden points, and shared attention weights.

Even without mechanisms from AlphaFold2, our modified TFN-Transformer outperforms all other PSCP methods on native backbones in terms of average RMSD on the CASP13 and CASP14 targets. On the other hand, the baseline SE(3)-Transformer of Fuchs et al. falls short of DLPacker for residues in the protein core (see Section 2.4). Part of this improvement is achieved by augmenting edge features with pairwise distance information between hidden points. We hypothesize that this information is especially important for deeper TFN-based architectures because information about relative distances between hidden coordinates is lost as a result of using spherical basis functions.

Method	Ours	DLPack	RosPack	FASPR	SCWRL4
Rel. Time	1.0	124.4	151.7	0.5	14.7

Table 2.17: **Time comparison of PSCP methods**. Relative times for reconstructing the side-chain atoms of all 83 CASP13 targets. We exclude the time for running our post-processing procedure, as this can be done in parallel on CPU. DLPacker (DLPack) and AttnPacker (Ours) were run on a single RTXA6000 GPU. RosettaPacker (RosPack), SCWRL4, and FASPR were run on a single AMD EPYC 7742 processor. With local triangle updates, our method reconstructs all side-chain atoms for all the CASP13 targets in 68 seconds.

On top of outperforming other popular methods, our model presents several other advantages. First, it is extremely fast. Table 2.17). shows the cumulative and relative time spent by each method for reconstructing side-chains of all the CASP13 targets. We can predict all side-chain conformations for a 600-residue protein in less than a second using a single Nvidia RTXA6000 GPU. On the other hand, DLPacker must be run iteratively for each amino acid side-chain creating a strong dependence on protein length at inference time. Our model is also very simple to use - it requires only a protein data bank (PDB) file. In contrast, OPUS-Rota4 [XWM21] requires voxel representations of atomic environments derived from DLPacker, logits from trRosetta100, secondary structure, and constraint files derived from

the output of OPUS-CM. Obtaining the requisite input data was too burdensome to provide a comparison.

Furthermore, since our method directly predicts side-chain coordinates, the output is fully differentiable which benefits downstream prediction tasks such as refinement or protein-protein interaction. This also circumvents the use of engineered energy functions and rotamer libraries and emphasizes architectural innovations and better loss functions.

AttnPacker also succeeds in efficiently modeling residue-level local environments by using locality-based graph attention during feature and structure generation stages. On the other hand, DLPacker and OPUS-Rota4 use 3D-voxelized representations of each amino acid's microenvironment - requiring space $O\left(v^3cd\right)$, where v is the voxelized width (40 in the case of DLPacker and OPUS), c is the number of channels, and d is the channel dimension. Although this choice of representation has helped facilitate good performance for each method, the memory requirements prevent simultaneous modeling of all amino acid side-chains which could ultimately hinder reconstruction accuracy. We hypothesize that simultaneously modeling all side-chains helps contribute to our method's success.

Although AttnPacker yields significant improvements in residue-level RMSD, we remark that traditional PSCP methods SCWRL, FASPR, and RosettaPacker still perform comparably in terms of χ_3 and χ_4 angle prediction. We guess that incorporating dihedral information into our model - either directly or through an appropriate loss function - could help improve performance.

In addition to side-chain packing, the design variant of AttnPacker is able to generate sequences with both high recovery (mean NSR of 47.8%, CASP13), and high in-silico success rates (mean scTM of 0.77, CASP13). These results are competitive with the inverse-folding method ProteinMPNN, which achieves a mean NSR of 48.5% and ESMFold scTM of 0.83 for the CASP13 targets. Unlike AttnPacker, ProteinMPNN does not directly model coordinates and is primarily featurized by pairwise distances between backbone atoms. In training ProteinMPNN, backbone atoms are perturbed with a small amount of Gaussian noise (SD

= 0.02Å). This data augmentation strategy is found to increase in-silico success rates and may explain some of the performance differences between the two methods. Furthermore, ProteinMPNN is trained on a combination of monomeric and multimeric inputs. The training set is clustered at 30% sequence identity and contains 23k clusters of protein assemblies in the PDB as of August 2021. At every training epoch, representatives from each cluster are re-sampled, which may also favorably impact performance.

CHAPTER 3

FLEXIBLE DOCKING

Protein complexes are vital to many biological processes, and their understanding can lead to the development of new drugs and therapies. Although the structure of individual protein chains can now be predicted with high accuracy, determining the three-dimensional structure of a complex remains a challenge. Protein docking, the task of computationally determining the structure of a protein complex given the unbound structures of its components (and optionally binding site information), provides a way to predict protein complex structure. Traditional docking methods rely on empirical scoring functions and rigid body simulations to predict the binding poses of two or more proteins. However, they often make unrealistic assumptions about input structures and are not effective at accommodating conformational flexibility or binding site information. In this chapter, we present DockGPT (Generative Protein Transformer for Docking), an end-to-end deep learning method for flexible and site-specific protein docking that allows conformational flexibility and can effectively make use of binding site information. Tested on multiple benchmarks with unbound and predicted monomer structures as input, we significantly outperform existing methods in accuracy and running time. Our performance is especially pronounced for antibody-antigen complexes, where we predict binding poses with high accuracy even without binding site information. Finally, we highlight our method's generality by extending it to simultaneously dock and co-design the sequence and structure of antibody complementarity-determining regions targeting a specified epitope.

This chapter presents work with Jinbo Xu. The main content was recently released as a pre-print in April 2023 [MX23].

3.1 Introduction

The bound configuration of two or more proteins helps regulate many biological processes, including signal transduction [LY16; Hub01], membrane transport [Süd95; Jia+22], and cell metabolism [DW08; Luz+21]. The process by which unbound protein chains bind together to form a complex is often controlled by more general protein-protein interactions (PPIs) [SB21; BMP13; Zou+18], and accordingly, aberrant PPIs are associated with various diseases, including cancer, infectious diseases, and neurodegenerative diseases [Lu+20]. The role of PPIs in protein complex formation makes selective targeting of PPIs an essential strategy for drug design and already forms the basis for several established cancer immunotherapies such as monoclonal antibodies [WSW10; Sin+19]. Although most proteins interact with partners to form a complex, experimental methods for determining the structures are often expensive and technically difficult to administer [Man13; RS02]. As a result, protein complexes account for only a small fraction of entries in the Protein Data Bank (PDB) [Ber+00], highlighting the need for effective *in silico* methods.

Although it is possible to infer protein complex structure from primary sequence information alone, in many cases, the three-dimensional structures of constituent (unbound) chains have already been experimentally determined. Moreover, extra information such as target binding sites or inter-chain contacts is readily available in many applications or can be derived through experimental methods such as cross-linking mass spectrometry [Orb+18]. In these scenarios, protein docking methods can be used to predict a complex structure. Despite having many practical applications [Kac+18; Bar+17; Ben+21], the efficacy of *in silico* protein docking or design methods is ultimately hindered by unrealistic assumptions about input structures, and failure to effectively utilize PPI information such as binding sites and inter-protein contacts.

Current computational methods for protein docking and design typically impose backbone and side-chain rigidity constraints and are trained to utilize specific side-chain interactions or protein backbone placements derived from native complexes which are already

optimal for binding [Cao+21; Tor+19]. Training computational models on only bound structures – in which binding interfaces match perfectly – is, in a sense "starting with the answer." In the real world, unbound chains typically lack shape complementarity because proteins tend to deform substantially upon binding [Mar+08; TB05], even for small-molecule ligands [MD09]. Accounting for backbone and side-chain flexibility can significantly increase the number of sequences that fold to the structure while maintaining the general fold of the protein [MK09], and is especially important for protein design because mutations in sequence often result in small changes to the backbone structure, [SHD22], and potentially large changes to surrounding side-chain conformations [Eya+03].

In addition to overlooking conformational flexibility, current methods tend to either ignore or ineffectively incorporate PPI information. For many applications, it is important to consider interactions as a particular binding site, such as targeting catalytic sites of enzymes or designing therapeutics to block a specific protein-protein interaction. A salient example is the design of neutralizing antibodies targeting the SARS-CoV-2 S protein, which initiates infection upon binding to the human angiotensin-converting enzyme 2 (ACE2) receptor [Cha+21; DYZ21]. In most cases, PPIs such as binding sites or inter-chain contacts are utilized only as a post-processing step to re-rank or filter out incompatible predictions.

Flexible docking and the design of protein complexes present several challenges for machine learning. First, the 3D geometry of multiple proteins is inherently difficult to represent. The difficulty arises from the fact that spatial relationships between receptor and ligand structures are ambiguous at the input level, yet inter-protein interactions must still be modeled jointly by the learning algorithm. Although several geometric deep-learning approaches offer a way to directly model 3D point clouds, so far, only one end-to-end machine-learning method has been proposed for general protein docking [Gan+22]. This method does not take into consideration backbone flexibility or bindings site information and suffers from excessive steric clashes in its predictions. Finally, sufficient training data is also scarce. Currently, there is no large dataset consisting of both protein complexes and their unbound components.

In this work, we introduce DockGPT, an end-to-end deep-learning approach to site-specific flexible docking and design. In developing DockGPT, we hypothesized that neural networks could accurately recover protein 3D coordinates from coarse or incomplete descriptions of their geometry. After affirming this capability, we approached flexible docking in a manner analogous to matrix completion followed by multidimensional scaling. In the matrix completion step, missing entries loosely correspond to inter-chain quantities such as distance and orientation. The imputed representation is then converted to 3D geometry to recover the bound complex. This framing allows us to naturally incorporate PPI information as input through residue-level binding interfaces or interfacial contacts. In addition, removing some intra-chain geometry allows us to simultaneously dock and design protein segments while still targeting specific binding sites.

To better incorporate flexibility into our predictions, we provide only a coarse description of intra-chain geometry, presenting distance and angle information within a resolution of at least 2Å and 20°, respectively. On top of this, we attempt to approximate the unbound state of each training example by applying Rosetta's FastRelax protocol [Lem+20] to individual chains.

To validate our approach, we perform an extensive comparison against four other protein docking methods on unbound chains from Antibody Benchmark (Ab-Bench)[Gue+21b], and Docking Benchmark Version 5 [Vre+15]. We also show that DockGPT performs well in docking protein structures predicted by AlphaFold2 [Jum+21] with high success rates. Finally, we demonstrate how to extend DockGPT to perform simultaneous docking and *de novo* design by docking antibody-antigen partners while concurrently predicting both the sequence and structure of all heavy chain complementarity-determining regions (CDRs).

3.2 Related Work

Traditional Methods for Protein Docking Protein docking is traditionally performed in three steps: (1) sampling of candidate conformations, (2) score-based ranking of candi-

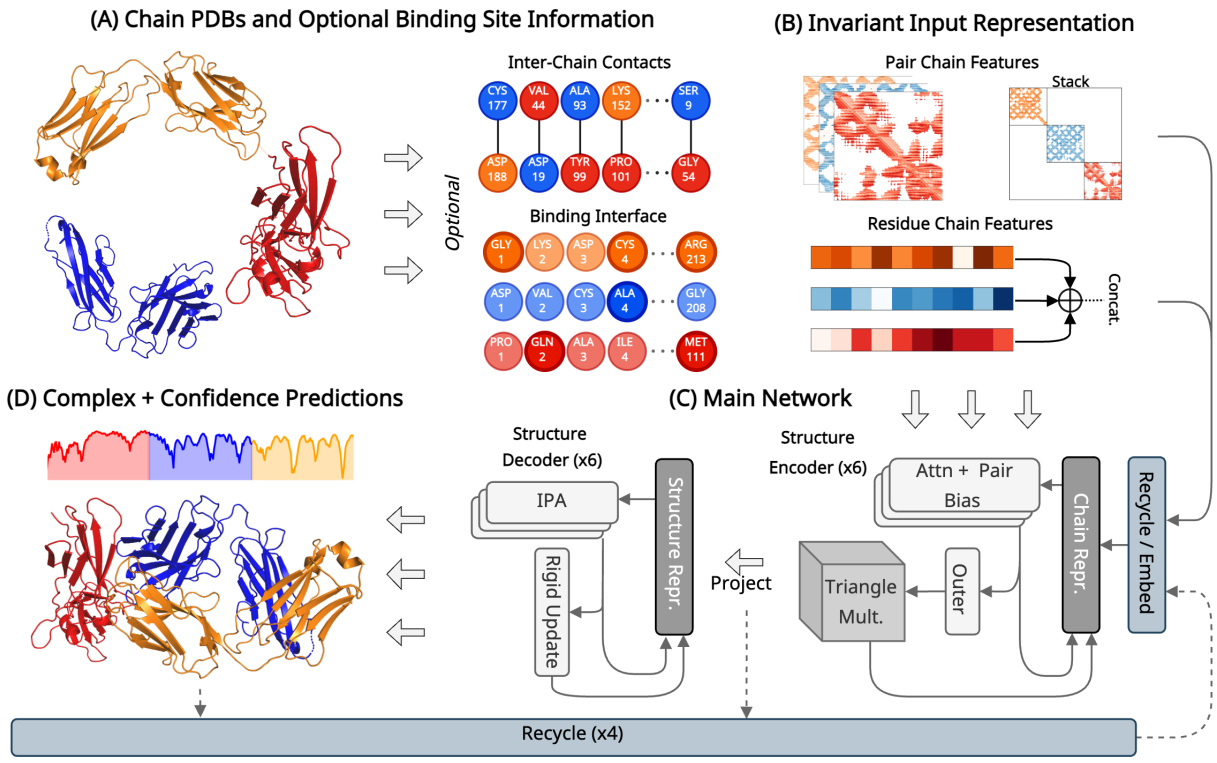

Figure 3.1: **Approach overview**. (**A**) Unbound chain sequence, coordinates, and (optional) information regarding binding interface(s) are given as input. (**B**) An invariant representation of 3D geometry is constructed for each chain from quantities such as pairwise atom distances and orientations. If interface residues or contacts are provided, this information is added to the respective residue and pair features. Other features are discussed in Section 3.3.1. (**C**) The main network consists of two submodules. The *structure encoder* develops a joint representation of the input chains and the *structure decoder* infers the 3D geometry. (**D**) The output of the main network is the complex 3D coordinates and per-residue confidence predictions. Steps (**C**) and (**D**) are repeated four times, with output residue, pair, and distance features recycled from the previous iteration.

dates, and (3) refinement of top-ranking complex structures. These algorithms primarily differ in either of the first two steps. Holding the position of the receptor fixed, each candidate conformation can be described by a 3-dimensional rotation and translation of the input ligand. Although the search space has relatively few degrees of freedom, the size of the effective candidate space can still total into the millions, even for small ligands [Ben+21]. In addition, the choice of score function usually induces a rugged energy landscape which is difficult to optimize over.

Within this paradigm, methods such as HDock [Yan+17; Yan+20a], PatchDock [Sch+05], ZDock [Pie+14], Attract [Vri+15], ClusPro [Koz+17], RosettaDock server [LG08], and Haddock [van+16], have been developed and made available for public access. Among these methods, PatchDock is one of the most widely used and computationally efficient. PatchDock avoids brute-force search over transformation space by matching protein surface patches based on "shape complementarity." Ligand transformations that align favorable patches bolster wide binding interfaces and avoid steric clashes resulting in favorable energy scores. HDock, ClusPro, and ZDock all make use of the Fast Fourier Transform algorithm to efficiently perform a global search on a 3D grid. The methods differ in how they post-process each candidate. ClusPro clusters candidates by root-mean-squared deviation and attempts to find a cluster with favorable energies. ZDock uses a combination of shape complementarity, electrostatics, and statistical potential terms for scoring. HDock, ranked as the number one docking server for multimeric protein structure prediction in the community-wide critical assessment of structure prediction 13 (CASP13-CAPRI) experiment in 2018, uses an iterative knowledge-based scoring function to discern the native complex. For a more complete review of traditional docking methods, available software, and accomplishments, we refer the reader to the comprehensive reviews [Agr+19; Wan+16; PST17].

The majority of these methods incorporate backbone and side-chain flexibility only as a post-processing step, e.g. through molecular dynamics simulations. In order to incorporate inter-chain contacts or binding site residues, traditional docking methods typically alter their score function, or restrict search or results to ligand transformations matching these criteria. For example, ZDock allows users to specify "undesirable" residue contacts and penalizes these interactions via the score function, and HDock applies post-processing to filter out predictions lacking target interactions.

Machine Learning for Protein Docking In the past, machine learning has been used outright or combined with physics-based methods for scoring docked complexes [Gue+21a;

AM15; McN+21]. Recently, end-to-end machine learning methods EquiDock [Gan+22] and EquiBind [Stä+22] were proposed for protein docking and docking drug-like molecules. In particular, EquiDock makes use of an SE(3)-equivariant graph matching network to output a single rigid rotation and translation, which, when applied to the ligand, places it in a docked position relative to the receptor. This is done by matching and aligning predicted *keypoints*, which roughly correspond to the centroid of the binding interface. Although this method provides favorable theoretical guarantees, it does not perform well in practice. On top of this, the independent SE(3)-equivariant graph matching network and training procedure are relatively complicated. Training EquiDock requires solving an optimal transport plan which matches predicted interface key points to ground truth positions for each example. Custom loss functions are developed to back-propagate gradients through alignments and to penalize surface intersections. Moreover, it is unclear how to extend this method to account for conformational flexibility or more than two interacting chains. In contrast, our framework is conceptually very simple, utilizes standard architectural components and losses, allows for flexibility, and is straightforward to extend to three or more chains.

***De novo* Binder Design** Recently, there has been a spate of interest in *de novo* protein design using deep learning, especially the design of small protein binders. AlphaDesign [JKS21] introduces a framework for *de novo* protein design which uses AlphaFold2 inside an optimizable design process, and [Ben+22] uses both AlphaFold2 and RosettaFold to improve the experimental success rate of their designs. Wang et al. [Wan+21] describe a method for *de novo* design of proteins harboring a desired binding or catalytic motif based on modifying the input and training of the RosettaFold network and augmenting the loss function. Here we show that our docking method can be easily extended to *de novo* design a protein that may bind a specific target site.

Geometric Deep Learning Complementary to this work, several geometric deep learning methods tailored explicitly towards modeling symmetries of point clouds have been proposed

[Sch+17; Tho+18; Fuc+20; Jin+21; SHW21b]. These methods have helped facilitate significant improvements in protein-related molecular modeling tasks such as protein structure prediction [Bae+21; Jum+21; Eva+22; Wu+22; Lin+22; Ahd+22], inverse folding [Jin+20; Hsu+22; MLX22], and *de novo* design [Wan+21; AA22; Jin+22; Ben+22; LMX22].

3.3 Methods

We now give an overview of our input representation, architecture, loss, training procedure, and training datasets.

Notation

To distinguish between multiple chains, we use $\mathcal{C}_1, \ldots, \mathcal{C}_k \subseteq \{1, \ldots, n\}$ to denote the partition of residue indices into chains $1, \ldots, k$. We also use $\mathcal{C}(i)$ to denote the chain containing residue i, i.e. $\mathcal{C}(i) \in \{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$.

3D rigid transformations $T = \left(R, \vec{\boldsymbol{t}}\right)$ are represented by a rotation $R \in SO(3)$, and translation $\vec{\boldsymbol{t}} \in \mathbb{R}^3$. We use the operator $\circ$ to denote the composition of rigid transformations

$$T \circ T' \triangleq \left(RR', \vec{\boldsymbol{t}} + R\vec{\boldsymbol{t}}'\right). \tag{3.1}$$

We use

$$T(\vec{\boldsymbol{x}}) \triangleq R\vec{\boldsymbol{x}} + \vec{\boldsymbol{t}} \tag{3.2}$$

to denote the action of the rigid transformation T on a vector $\vec{x} \in \mathbb{R}^3$. For notational convenience and as a visual aid, we adopt the notation

$$[\vec{\boldsymbol{x}}]_T \triangleq T^{-1}(\vec{\boldsymbol{x}}) \tag{3.3}$$

to denote the vector of coordinates $\vec{\boldsymbol{x}}$ in the *local frame* defined by T.

3.3.1 Input Features

The input to our network is a complete graph $G = \big(V = \{\boldsymbol{x}_i\}, E = \{\boldsymbol{e}_{ij}\}\big)$ where V consists of residue features $\boldsymbol{x}_i$ and E consists of pair features $\boldsymbol{e}_{ij}$ between residues i and j. The bulk of our input features are generated independently for each input chain. We refer to those features which do not depend on the input complex as *intra-chain features* and those which do as *inter-chain features*. In the interest of clarity, we first describe intra-chain features, which are independent of the protein complex being predicted.

Intra-Chain Features

Residue Features We generate residue features for each chain and join them by concatenating along the sequence dimension. The input feature x_i associated with residue i consists of four encodings:

$$\boldsymbol{x}_i^{\text{chain}} = \Big(E_{\text{AA}}\left(s_i\right), E_{\text{pos}}\left(i, |\mathcal{C}(i)|\right), E_{\text{cen}}(i, \vec{\boldsymbol{x}}_i^{C\beta}), E_{\text{dih}}\left(\theta_i\right)\Big). \tag{3.4}$$

The first, $E_{\text{AA}}\left(s\right)$, is a one-hot encoding of the amino acid type s using 20 bins for each naturally occurring amino acid. The next, E_{pos}, encodes the residue relative sequence position into ten equal-width bins. As a proxy for estimating whether a residue is on the protein's surface, we use a centrality encoding, E_{cen}, which corresponds to the number of $C\beta$ atoms in a ball of radius 10Å around the query residue. We encode this feature with six radial basis functions equally spaced between 6 and 40 and only consider residues in the same chain as the query atom. Last, E_{dih}, encodes the angle $\theta \in [-180^\circ, 180^\circ]$ into 18 bins of width 20°. The input $\theta_i \in \{\phi_i, \psi_i\}$ are the phi and psi backbone torsion angles of residue i. For residues before and after chain breaks or at the N and C terminus of a chain, we set the phi and psi angles to 0.

Pair Features Pair features are made up of low-resolution descriptions of pairwise distance and orientation and relative sequence information. The features for each chain are stacked to form a block-diagonal input matrix. A separate learned parameter is used to fill the missing off-diagonal entries. For a pair of residues i and j, in a common chain, the corresponding feature $\boldsymbol{e}_{ij}$ consists of three one-hot encodings

$$\boldsymbol{e}_{ij}^{\text{chain}} = \left(E_{\text{dist}}\left(\|\vec{\boldsymbol{x}}_i^{C\alpha} - \vec{\boldsymbol{x}}_j^{a}\|_2\right), E_{\text{ori}}\left(\theta_{ij}\right), E_{\text{sep}}\left(i-j\right)\right). \tag{3.5}$$

E_{dist} is an encoding of the distance d into six equal-width bins between 2Å and 16Å, with one extra bin added for distances greater than 16Å. We include distances between $C\alpha$ and each atom type $a \in \{N, C\alpha, C, C\beta\}$. E_{ori}, encodes the angle θ performed in the same manner as the backbone dihedral encoding for residue features. The input angles $\theta_{ij} \in \left\{\phi_{ij}, \psi_{ij}, \omega_{ij}\right\}$ are pairwise residue orientations defined in [Yan+20b]. Note that all pairwise distances and angles are known only within a resolution of at least 2Å and 20°, respectively. The last feature, $E_{\text{sep}}\left(\cdot\right)$, is a one-hot encoding of signed relative sequence separation into 32 classes, in the same manner as [MLX22].

Inter-Chain Features

We add three additional features to encode information about the target protein complex and PPIs.

Inter-Chain Interface (Residue) $f_i \in \{0, 1\}$ is an optional binary flag indicating whether the $C\alpha$ atom of residue i is within 10Å of a $C\alpha$ atom belonging to a residue in a different chain. This flag is 0 if this criterion does not hold.

Inter-Chain Contact (Pair) $f_{ij} \in \{0, 1\}$ indicates whether two residues in separate chains are in contact. This occurs when the distance between $\vec{\boldsymbol{x}}_i^{C\alpha}$ and $\vec{\boldsymbol{x}}_j^{C\alpha}$ is less than 10Å. This flag is 0 if this criterion does not hold.

Relative Chain (Pair) A one-hot encoding of the relative chain index for residues i and j into three classes. Let $c_i,\ c_j \in \{1, \ldots, k\}$ denote the chain index of residues i and j, then $f_{ij} = \text{OneHot}\left(\text{sign}\left(c_i - c_j\right)\right)$, where $\text{sign}\,(x) \in \{-1, 0, 1\}$.

The interface and contact flags provide context for residues on the binding interface for each chain, restricting the effective search space during inference. In real-world applications, knowledge of the binding interface may be limited or unknown. In light of this, we provide only a limited number of contacts or binding residues, chosen randomly for each training example. Specifically, for each input, we include no contacts or no residue flags independently, with a probability of $1/2$. This means that during training, the method sees 25% of examples without any interface or contact information, 50% with one or the other, and 25% with both features provided, on average. If interface features are included, we randomly sub-sample a number of interface residues $N_{\text{int}} \sim \text{geom}\,(1/5)$ to include, meaning five residues are selected on average. Similarly, we sub-sample $N_{\text{con}} \sim \text{geom}(1/3)$ inter-chain contacts when this feature is used, resulting in three provided contacts on average.

The relative chain encoding provides a way to distinguish between intra-chain and inter-chain pair features. By taking a signed difference, pair features e_{ij} and e_{ji} receive different encodings when i and j are in distinct chains. This not only discriminates the endpoints as belonging to different chains but also breaks symmetry.

3.3.2 Architecture and Hyperparamter Details

Figure 3.2 shows a schematic overview of our model architecture and loss. We do not explicitly show our structure encoder module since it differs only slightly from the structure decoder; the rigid update is removed, and IPA block is replaced with a pair-biased attention block. Though not displayed in the figure, we follow the Pre-LayerNorm scheme described in [Xio+20] where layer normalization [BKH16] is placed inside the attention and transition residuals. In addition (pun intended), we use ReZero [Bac+20] for all residuals. Each feed-forward transition consists of one hidden layer having a dimension four times that of

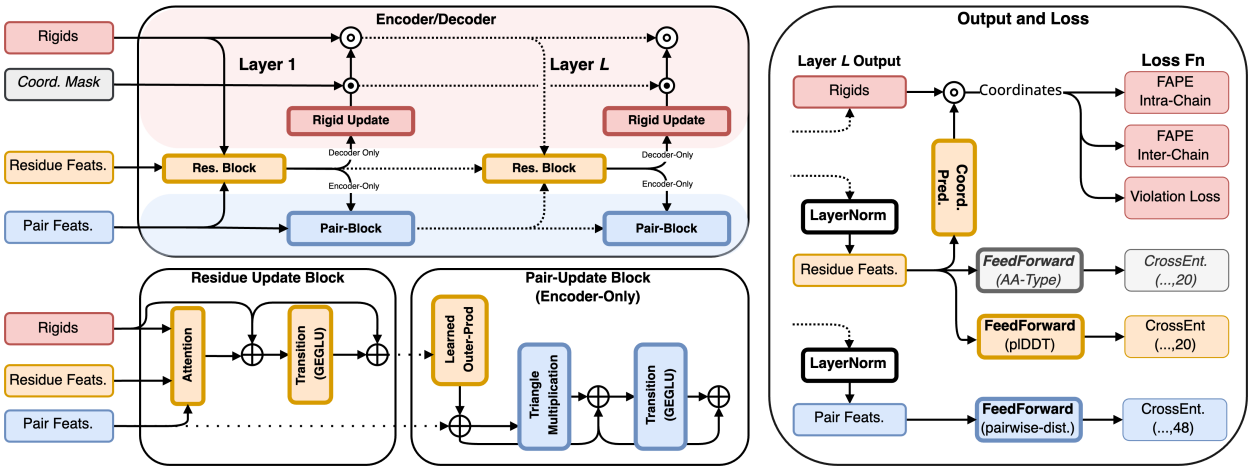

Figure 3.2: **DockGPT architecture and loss**. Learnable modules are shown with bold text and bold borders. Modules operating on residue features are shown in orange, and those operating on pair features are shown in blue. Modules making direct or indirect use of coordinates are shown in red. Optional input and sequence and structure co-generation modules are shown in light gray. We use $\oplus$ to denote residual operations, $\odot$ to denote element-wise multiplication and $\circledcirc$ to denote the element-wise composition of two rigid transformations. Labeling of IPA and pair blocks with the index of their respective layer is omitted, as block weights may be shared across multiple layers. We highlight with a blue/red background those modules which are encoder/decoder specific (i.e., pair updates are omitted in the decoder, and rigids are omitted in the encoder). The structure encoder uses pair-biased multi-head attention for residue updates, and the structure decoder uses invariant point attention. Finally, layer normalization is applied but not displayed here, except in deriving the logits for loss.

the input dimension. For pointwise nonlinearity, we use gated GELU (GeGLU) based on success in other sequence modeling tasks [HG16; Sha20]. The Learned Outer-Prod module is nearly identical to the outer-product mean module described in [Jum+21], except we use a smaller intermediate dimension ($c = 16$ vs. $c = 32$) and skip the mean operation. The rigid update maps residue features to a per-residue rigid rotation and translation. This is implemented as a learned linear projection preceded by layer normalization. The composition of rigid transformations is implemented in the same manner as the backbone update in AlphaFold2 (see [Jum+21] Supp. Material, Algorithm 23). In some settings, incorporating prior coordinate information may be useful. Specifically, a subset of coordinates can be held fixed by replacing the corresponding rigid rotation and translation updates with the

respective identity transformations (i.e., I_3 and $\vec{\mathbf{0}}$). More details on this are provided in Section 3.6.1.

We use a hidden dimension of 256 for residue features and 128 for pair features in both submodules. All triangle multiplication updates use four heads of dimension 32 for queries and values. In the encoder submodule, we use eight attention heads of dimension 32 for each residue update block. Decoder IPA uses 12 heads per block, with dimensions 16 for scalar features and four/eight for point queries/values.

3.3.3 Network Architecture and Training

We design a two-stage network making use of triangle multiplication, pair-biased attention, and invariant point attention (IPA). Our first module, which we refer to as the "structure encoder," produces an invariant representation of the protein complex, which is subsequently converted to 3D coordinates by the second module, the "structure decoder." Our encoder uses pair-biased attention to update residue features and triangle multiplication to update pair features. The decoder updates only residue features using IPA. We also make use of feature recycling during training and inference. We note that although our architecture modifies or extends some components in AlphaFold2, the two architectures are functionally quite distinct. We do not make use of multiple sequence alignments (MSAs), templates, global attention, self-distillation, or other elements contributing to the success of AlphaFold2. In contrast, we hope to learn the principles governing protein binding from sequence and structure alone and develop a more specialized architecture to do so.

Network Architecture

Here, we provide a general overview of our architectural components. A schematic overview of the architecture and loss can be found in Figure 3.2. Complete implementation details and more thorough descriptions of each submodule can be found in Section 3.3.2.

Structure Encoder Layer Our encoder produces a joint representation of the input chains. Since inter-chain features are mostly missing from the input, we hypothesized that a network that updates pair features would facilitate more successful docking. Consequently, we chose to update pair features using incoming and outgoing triangle-multiplicative updates [Jum+21].

$$\boldsymbol{x}_i^{(\ell+1)} = \textbf{Pair-Bias-Attn-Block}^{(\ell)}\left(\boldsymbol{x}_i^{(\ell)}, \boldsymbol{e}_{ij}^{(\ell)}\right) \tag{3.6}$$

$$\boldsymbol{e}_{ij}^{(\ell+1)} = \textbf{Pair-Block}^{(\ell)}\left(\boldsymbol{x}_i^{(\ell+1)}, \boldsymbol{e}_{ij}^{(\ell)}\right) \tag{3.7}$$

Each layer has two update blocks. The first block updates the residue features using multi-head attention with pair bias. The next block transforms the updated residue features into an update for the pair representation using a learned outer product and then applies triangle multiplication and a shallow feed-forward network to the result.

Structure Decoder Layer The decoder module converts the encoder representation to 3D Geometry. Since we do not make direct use of coordinate information in our input (although we show that this can be done for special cases in Section 3.3.3), we sought an invariant architecture specialized for coordinate prediction and ultimately settled on IPA. At each layer, node features are transformed according to

$$\boldsymbol{x}_i^{(\ell+1)} = \textbf{IPA-Block}\left(\boldsymbol{x}_i^{(\ell)}, \boldsymbol{e}_{ij}^{\text{enc}}, T_i^{(\ell)}\right) \tag{3.8}$$

$$T_i^{(\ell+1)} = T_i^{(\ell)} \circ \textbf{RigidUpdate}\left(\boldsymbol{x}_i^{(\ell+1)}\right), \tag{3.9}$$

where $\boldsymbol{e}^{enc}$ denotes the pair feature representation at the last encoder layer.

We use a total of six decoder layers, sharing the same weights for all six layers. We perform recycling during training and inference, allowing us to execute our model multiple

times on the same example. This is done by embedding the previous iteration's outputs in the next iteration's inputs. Our best-performing model uses the same scheme as described in the AlphaFold2 implementation ([Jum+21], supplementary section 1.10). In concurrence with AlphaFold2 and OpenFold [Ahd+22], we find that recycling significantly improves prediction quality while incurring only a constant increase in inference and training time. We experimented with recycling features from the structure decoder rather than the encoder. Since the decoder residue features encode plDDT information, we hypothesized that this information could better inform future iterations. This ablation and others are shown in Section 3.5.

Coordinate Prediction In predicting residue-wise atom coordinates, we deviate from the strategy of AlphaFold2 and simply compute the local-frame coordinates for each atom using a learned linear projection. The coordinates for $C\alpha$ are taken as the translation component of the per-residue predicted rigid transformation, and the remaining atom coordinates are predicted as

$$\vec{\boldsymbol{x}}_i^a = T_i^{(L)} \circ \text{Linear3D}\Big(\text{LayerNorm}\left(\boldsymbol{x}_i^{(L)}\right)\Big) \tag{3.10}$$

where L denotes the index of the last decoder layer, and Linear3D is a learned projection into dimension $|\mathcal{A}| \times 3$. Note that only one rigid transformation is used to produce all atom coordinates for a given residue.

Handling Coordinates as Input Although we do not explicitly use coordinates in our docking model, for certain tasks, it may be important to incorporate this information as part of the input. This is especially salient in antibody loop design, where the framework region remains mostly rigid upon binding. In Section 3.6.1, we show how to modify Equation (3.9) and Equation (3.10) to easily incorporate rigid, flexible, and missing coordinates as part of the input while still maintaining SE(3)-equivariance. We also provide empirical results for

designing CDR loops with this modification in Section 3.6.2.

3.3.4 Loss

Our network is trained end-to-end with gradients coming from frame-aligned point error (FAPE), pairwise distance, per-residue lDDT (plDDT), and a few other auxiliary losses. We remark that our implementation of FAPE differs from that in AlphaFold2, as we use a different method for predicting coordinates. Other modifications were made in the clamping procedure of FAPE loss to facilitate faster convergence. A complete overview is provided throughout the remainder of this section.

We use the same loss function for pre-training and fine-tuning:

$$\mathcal{L} = 1.0\mathcal{L}_{\text{FAPE}} + 1.0\mathcal{L}_{\text{aux-FAPE}} + 0.5\mathcal{L}_{\text{dist}} + 0.5\mathcal{L}_{\text{plDDT}} + 0.2\mathcal{L}_{\text{viol}}. \tag{3.11}$$

Each term in the loss (Equation (3.11)) is described in the remainder of this section.

Similar to AlphaFold2, we also apply an averaged FAPE loss $\mathcal{L}_{\text{aux-FAPE}}$ on the intermediate structures produced by the shared-weight layers of our decoder module. For the results in Sections 3.6 and 3.6.2, we add an additional term which is an averaged cross-entropy loss for amino acid identity, given weight 0.5

When writing equations for loss functions, we assume that we have output residue features $\{\boldsymbol{x}_i\}$, pair features $\{\boldsymbol{e}_{ij}\}$, rigid transformations $\{T_i\} = \left\{\left(R_i, \vec{\boldsymbol{t}}_i\right)\right\}$, and coordinates $\{\vec{\boldsymbol{x}}_i^a\}$ for each atom type $a \in \mathcal{A}$ as described in Section 3.3.3.

Per Residue lDDT

Residue output features are used to predict per-residue local distance difference test scores (plDDT). We compute lDDT as in Equation (2.14). In defining the ground-truth labels, there are two reasonable approaches. The first approach directly uses predicted coordinates,

$$\text{plDDT}_i \quad = \quad \frac{1}{|\mathcal{N}(i)|} \cdot \sum_{j \in \mathcal{N}(i)} \text{lDDT}\left(\text{abs}\left(\left\|\vec{\boldsymbol{t}}_i - \vec{\boldsymbol{t}}_j\right\|_2 - \left\|\vec{\boldsymbol{t}}_i^* - \vec{\boldsymbol{t}}_j^*\right\|_2\right)\right), \tag{3.12}$$

where $\mathcal{N}(i) = \left\{j : \left\|\vec{\boldsymbol{t}}_i^* - \vec{\boldsymbol{t}}_j^*\right\|_2 < 12\text{Å}\right\}$, and

The alternative approach compares coordinates as they are seen in the predicted local frames of each residue. For this, we use predicted rigid transformations $T_i = \left(R_i, \vec{\boldsymbol{t}}_i\right)_{i=1,\ldots,n}$ and true rigids $T_i^* = \left(R_i^*, \vec{\boldsymbol{t}}_i^*\right)_{i=1,\ldots,n}$ obtained from the native conformation to compute the local pLDDT score as:

$$\begin{aligned}
&\text{plDDT-Local}_i = \\
&\frac{1}{|\mathcal{N}(i)|} \cdot \sum_{j \in \mathcal{N}(i)} \text{lDDT}\left(\text{abs}\left(\left\|\left[\vec{\boldsymbol{t}}_i\right]_{T_i} - \left[\vec{\boldsymbol{t}}_j\right]_{T_i}\right\|_2 - \left\|\left[\vec{\boldsymbol{t}}_i^*\right]_{T_i^*} - \left[\vec{\boldsymbol{t}}_j^*\right]_{T_i^*}\right\|_2\right)\right) \\
&= \frac{1}{|\mathcal{N}(i)|} \cdot \sum_{j \in \mathcal{N}(i)} \text{lDDT}\left(\text{abs}\left(\left\|\left[\vec{\boldsymbol{t}}_j\right]_{T_i}\right\|_2 - \left\|\left[\vec{\boldsymbol{t}}_j^*\right]_{T_i^*}\right\|_2\right)\right).
\end{aligned} \tag{3.13}$$

Ultimately, we use the standard plDDT to train our model. Although local-frame coordinates and distances are compared in each IPA head, we found that the local plDDT produces less accurate confidence estimates, and is also more difficult to optimize. Nevertheless, we include the alternative definition as it may be of interest to some readers.

To compute plDDT loss, we pass our output residue features $\boldsymbol{x}_i$ through a shallow feed-forward network with output representing 20 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.

Pairwise Distance

We predict pairwise distances for four atom pairs $(C\alpha, X)$, where $X \in \{N, C\alpha, C, C\beta\}$ from 2-20Å using a bin width of 0.4Å. An extra bin is added for distances beyond 20Å. We do not separate inter and intra-chain atom pairs. Cross entropy loss is applied to compare the prediction to the ground truth.

Violation Loss

Unlike AlphaFold2, we predict only a single rigid transformation for each input residue. This means that intra-residue bond lengths and angles must be learned in the linear projection used to obtain predicted atom coordinates. We find that violation loss is very important for generating physically realistic conformations, and also for avoiding unfavorable steric interactions such as surface intersection. Here we use the same violation loss as defined in AlphaFold-Multimer; bond angle, bond length, and one-sided flat bottom steric penalty. We omit the "Center of Mass" loss [Eva+22, eq.1] as it had no empirical effect on performance.

FAPE

Here we describe a slight modification of the frame-aligned point error (FAPE) loss described in [Jum+21; Eva+22]. We reiterate that only a single rigid transformation is predicted for each residue and thus rigid transformations for each output atom type cannot be directly compared.

Given predicted atom coordinates $\vec{\boldsymbol{x}}_j^a$ for each atom $a \in \mathcal{A}_j$ of residue j, we compute the per-residue FAPE, (pFAPE) for residue i as

$$\text{pFAPE}\left(T_i, \vec{\boldsymbol{x}}_j^a; \theta\right) = \underset{j, a \in \mathcal{A}_j}{\text{mean}} \left(\min \left(\left\| \left[\vec{\boldsymbol{x}}_j^a\right]_{T_i} - \left[\vec{\boldsymbol{x}}_j^{a,*}\right]_{T_i^*} \right\|_2, \theta \right) \right) \tag{3.14}$$

the FAPE loss over all residues is then

$$\text{FAPE}\left(T_i, \vec{\boldsymbol{x}}_j^a; \theta\right) = \frac{1}{\theta} \cdot \underset{i}{\text{mean}}\left(\text{pFAPE}\left(T_i, \vec{\boldsymbol{x}}_j^a; \theta\right)\right) \tag{3.15}$$

Our network employs two FAPE loss terms, each with equal weight. The first, FAPE_{intra} is intra-chain FAPE which restricts the computation to pairwise relative coordinates within the same chain. The second is Inter-chain FAPE which applies the loss between atom coordinates in separate chains. Formally,

$$\begin{aligned}\mathcal{L}_{\text{FAPE}} = \frac{1}{k} \cdot \sum_{\mathcal{C} \in \{\mathcal{C}_1, \ldots, \mathcal{C}_k\}} \Bigg(& \underbrace{\text{FAPE}\left(\{T_i\}_{i \in \mathcal{C}}, \left\{\vec{\boldsymbol{x}}_j^a\right\}_{j \in \mathcal{C}}; \theta_{intra}\right)}_{\text{intra-FAPE}} \\ & + \underbrace{\text{FAPE}\left(\{T_i\}_{i \in \mathcal{C}}, \left\{\vec{\boldsymbol{x}}_j^a\right\}_{j \notin \mathcal{C}}; \theta_{inter}\right)}_{\text{inter-FAPE}} \Bigg) .\end{aligned} \tag{3.16}$$

Following AlphaFold-Multimer, we use $\theta_{intra} = 10$, and $\theta_{inter} = 30$ with probability 0.9 and randomly set $\theta = \infty$ for each FAPE type with probability 0.1.

3.3.5 Training

For general protein docking, model training is split into two stages. In the first stage, we pre-train on a mix of complexes and monomers, randomly selecting a monomer or a complex to train on with equal probability. This repeats for five epochs. The rationale for this decision is described in Section 3.3.5. Afterward, monomers are removed, and we train exclusively on complexes. For all complexes in our training sets, we relaxed each individual chain using Rosetta's FastRelax protocol [Lem+20] with all default settings (antibody heavy and light chains were relaxed jointly when applicable). For antibody-antigen docking results, we fine-tuned the model on a dataset consisting of only antibody-antigen complexes.

All models were trained on 48Gb Nvidia RTX A6000 GPUs and optimized using Adam

[KB15] with default parameters ($\beta_1 = 0.9,\ \beta_2 = 0.999,\ \epsilon = 10^{-8}$), with learning rate 10^{-3} during pre-training, and $5 \cdot 10^{-4}$ afterwards. We apply per-example gradient clipping by global norm as described in [Jum+21, supplementary material, section 1.11.3], and scale the loss of each example by the log of the total number of residues to up-weight larger complexes. We validate our model every 500 mini-batches, using a minibatch size of 24. We train our model for at most 15 epochs and apply early stopping with patience of eight validation steps. Since ReZero is used for residuals, we do not use any learning rate warm-up.

During the mixed monomer/multimer pre-training phase, we crop complex chains so that the total number of residues does not exceed 500. We also remark that during the pre-training stage, we append a binary flag to each residue and pair feature indicating whether the input corresponds to a single chain – in which case the chain should be treated as rigid. For general multimer training and antibody fine-tuning, we place the encoder and decoder modules on separate GPUs and increase the crop size to 800 amino acids. Any complex containing a chain with more than 550 residues is removed from our training datasets. When cropping antibody-antigen complex chains, we randomly sample a contiguous subset of antigen residues so that the total number of resulting residues is 800. We follow the same strategy for general proteins but choose a chain to crop at random. We note that no cropping was performed at inference time for any of the results in this paper.

Rationale for Monomer Pre-Training While developing this model, we first ran experiments to understand how well our architecture performed on multidimensional scaling tasks. For this, we sought to recover the $C\alpha$ trace of protein chains given only distance and inter-residue orientation. We found that our deep model was able to recover the original $C\alpha$-trace with sub-angstrom RMSD using a 2Å resolution for distances, and 20° resolution for angles after around 4k mini-batches (approximately 1.5 epochs).

We attempted to apply the same model to learn rigid docking, providing the same intra-chain information but excluding all inter-chain features. In these experiments, the model

struggled to reconstruct the conformations of the respective chains with reasonable accuracy and showed a tendency to favor auxiliary loss terms such as pairwise distance. This behavior persisted even after significantly more gradient updates.

Considering this, we decided to separate FAPE loss into inter and intra-chain components, similar to what is done in [Eva+22], and pre-train our model on a 50-50 split of protein complexes and monomers. This resulted in significantly faster convergence in FAPE loss and far more accurate 3D models. We remark that a single float (1 or 0) is appended to each residue and pair feature to indicate if the input is a complex or monomer.

Training Datasets

Monomers For pre-training, with single chains, we randomly sample chains from the publicly available BC40 dataset, consisting of roughly 37k chains filtered to 40% nonredundancy. Proteins with greater than 40% sequence similarity to any chain in our test datasets are removed.

General Protein Complexes We use a subset of the publicly available Database of Interacting Protein Structures (DIPS)[1] [Tow+18]. The training set is generated to exclude any complex that has any individual protein with over 30% sequence identity when aligned to any protein in the Docking Benchmark Version 5 test set (described in Section 3.3.6). We follow the training and validation splits for DIPS used in [Tow+18], with 33159 and 829 complexes, respectively.

Antibody-Antigen Complexes For fine-tuning on antibody complexes, we use the publicly available Structural Antibody Database (SAbDab), which consists of 4994 antibody structures renumbered according to the Chothia numbering scheme [CL87; Cho+89; ALC97].

1. Downloaded from https://github.com/drorlab/DIPS.

Various papers from Chothia have conflicting definitions of heavy-chain CDRs [2]. In light of this, we use the most recent definitions from [ALC97]. We generate train and test splits based on antigen sequence similarity, filtering out examples where antigen chains have more than 40% sequence identity using mmseqs [HSS16]. Before generating clusters, we removed all targets with overlap in our test sets using the same criteria. We remark that no filtering is performed against antibodies. This results in roughly 3k complexes, for which we use an 8:1:1 split for training, validation, and testing.

3.3.6 Evaluation Criteria

To measure docking prediction quality, we report interface root-mean-square deviation (I-RMSD), ligand root-mean-square deviation (L-RMSD), DockQ score, and DockQ success rate (SR) as reported by the DockQ algorithm[3] [BW16]. DockQ score is a single continuous quality measure for protein docking models based on the Critical Assessment of PRedicted Interactions (CAPRI) community evaluation protocol. In Figure 3.4, we provide some visual examples of low, acceptable, and medium-quality complex predictions according to the DockQ metric. For antibody chains, we sometimes report CDR-RMSD, which is calculated after superimposing the $C\alpha$ atoms of the heavy and light chain framework regions using the Kabsch algorithm [Kab76]. Finally, we sometimes include complex root-mean-square deviation (C-RMSD), which is derived by simultaneously superimposing all $C\alpha$ atoms between two protein complexes. We visually illustrate the difference between C-RMSD, L-RMSD and I-RMSD in Figure 3.3.

2. See here for a nice summary of CDR numbering schemes and changes in corresponding CDR loop definitions over time.

3. DockQ is publicly available for download at http://github.com/bjornwallner/DockQ/

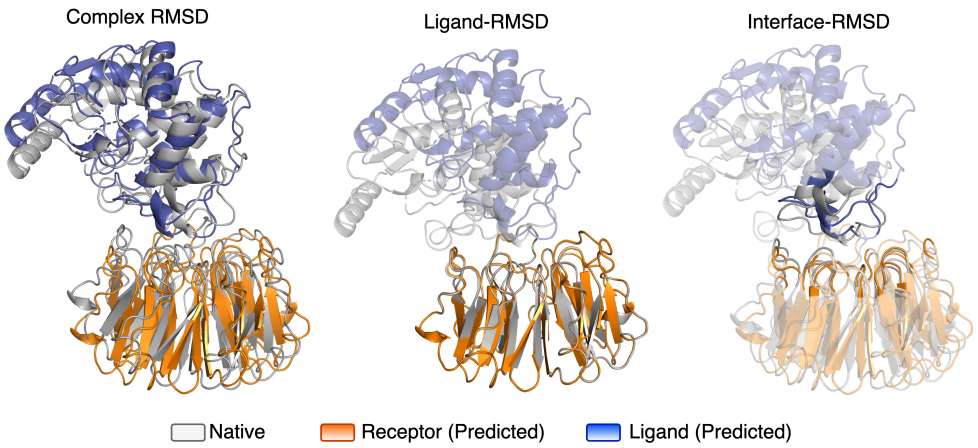

Figure 3.3: **Illustration of common RMSD types used to assess protein complex predictions (PDB entry 1JTD)**. The regions which are aligned for each RMSD type are set as opaque.

When assessing top-k performance, we take the best score over the top-k ranked predictions of each target. When interface residues or contacts are specified, the information is randomly sampled from the native complex. If binding site information is given, each method is run fifteen times for each target, each run with different random samples. For energy-based methods, outputs are ranked by predicted energy. For our method, we use the predicted interface lDDT to rank each prediction. (see Section 3.4.3 for details).

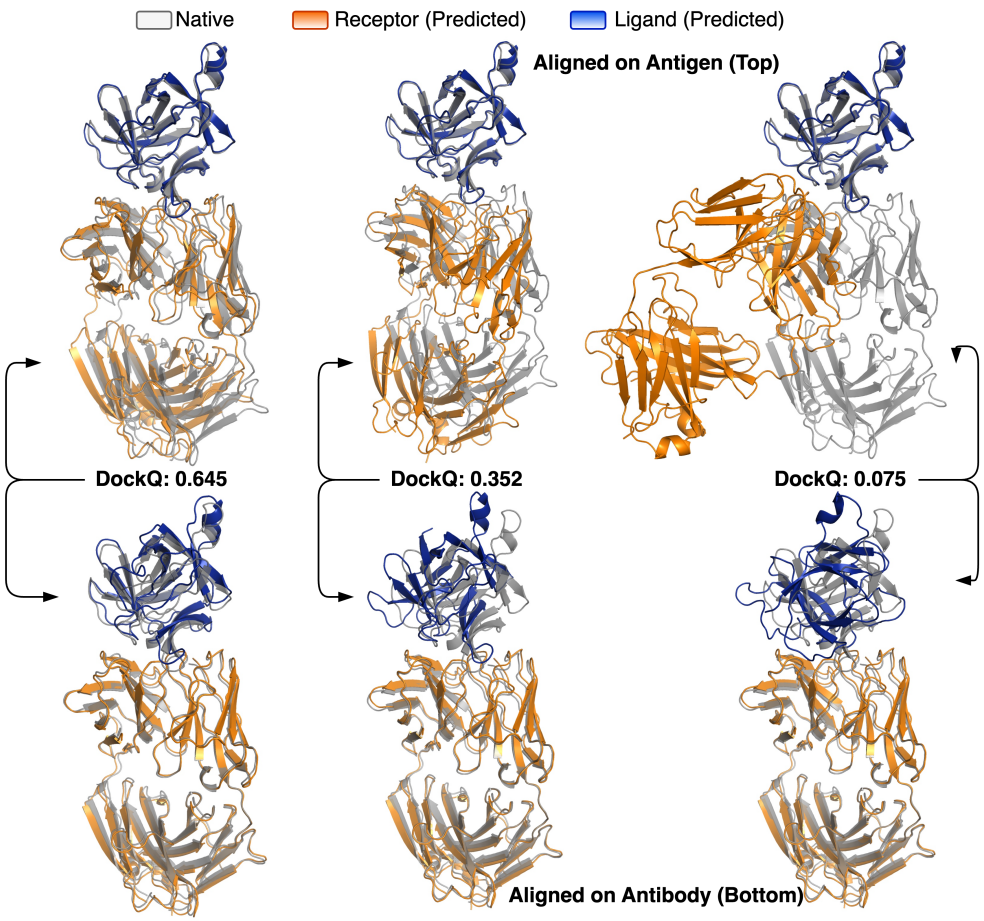

Figure 3.4: **Illustration of complex predictions with varying DockQ scores**. Three predictions are shown for PDB entry 4G6M. Distinct predictions are shown in each column. The bottom row shows the predicted complex aligned to the ground-truth antigen (receptor), and the top row shows the same predictions aligned to the ground-truth antibody (ligand).

Docking Paradigms

In this work, we are primarily concerned with predicting the bound conformation of a protein complex, given only unbound conformations of constituent chains. This is easily confused with *redocking*, the task of predicting a protein complex given *bound* conformations of each chain as input. Redocking is considerably easier than docking. For this task, traditional score-based methods are able to accurately predict most protein complexes. We verify this claim in Section 3.4.1, where we consider redocking antibody-antigen complexes.

When assessing docking performance, we sometimes condition on information about PPIs,

such as interacting residues. Traditionally, amino acids are defined as interacting if any heavy atoms are within 6Å from one another. In this work, we used a more relaxed definition, where residues are defined as interacting if the distance between their $C\alpha$ atoms is less than 10Å. This definition applies to downstream protein design tasks, where knowledge of sequence or side-chain conformations may be missing or incomplete. In some cases, we provide the identity of select residues on the binding *interface* of a complex. In other settings, we provide *contacts*, which correspond to interacting residue pairs. We refer to the setting where neither interface residues nor contacts are specified as *blind* docking.

Benchmarks

For each benchmark, we include only receptor-ligand pairs having at least four contacts and maximum chain-wise RMSD less than 10Å from the bound state. We note that some of the baselines might have used part of the DB5 test set in validating their models, and thus the scores may be optimistic. In addition to bound and unbound structures, we include comparisons using receptor and ligand structures predicted by AlphaFold2 or AlphaFold-Multimer[Eva+22]. The same filtering criteria are applied to predicted structures.

Antibody Benchmark (Ab-Bench) [Gue+21b] A non-redundant set of 46 test cases for antibody-antigen docking and affinity prediction. This set contains both bound and unbound structures with diverse CDR-loop conformations between the bound and unbound states, ranging from $\leq$ 1Å to $\geq$ 4Å for CDR-H3. When AlphaFold-predicted structures are used as input, 26 test cases are used.

Docking Benchmark Version 5 (DB5) Test [Vre+15] To the best of our knowledge, DB5 [Vre+15], which contains 253 structures, is the largest dataset containing both protein complexes and the unbound structures of their components. We use 42 complexes from the DB5 test set, which are held out by the DIPS training split. For predicted structures, we also gathered 26 receptor-ligand pairs meeting the filtering criteria.

Rosetta Antibody Design (RAbD) [Ado+18] A set of 46 κ and 14 λ antibody-antigen complexes. The benchmark contains antibodies with high CDR diversity and a wide range of length classes. All structures have experimental resolution ≤ 2.5Å, buried surface area in the antibody-antigen complex of $\geq 700^2$, and contacts with CDRs in both the light and heavy chain regions. These structures were used to assess the performance of docking algorithms in the bound input context, and results are given in Table 3.1.

3.4 Docking Results

We compare DockGPT against ZDock [Pie+14], HDock [Yan+20a], PatchDock [Sch+05], and EquiDock [Gan+22]. We downloaded their code and ran them locally. More details can be found in Appendix B.1.

In addition to docking software, we compare with AlphaFold-Multimer [Eva+22] in Section 3.4.5, fig. 3.14, and table 3.2. We do not do so in the main text, as the focus of this manuscript is protein docking and assessing the ability of docking programs to target specific binding sites. In general, current complex prediction algorithms such as AlphaFold-Multimer do not explicitly make use of binding site information, although this may be derived implicitly via multiple sequence alignments or templates. That is, they lack the ability to target specific binding modes, which further highlights the importance of effective docking methods.

3.4.1 Antibody Docking

We now compare methods on docking antibody-antigen unbound and predicted structures from the Antibody Benchmark dataset. As shown in Figure 3.5, for all but a few cases, our performance on docking AlphaFold2-predicted structures roughly matches that on unbound inputs. In the interest of brevity, we report statistics for unbound inputs unless otherwise specified. Additional results and tables with RMSD and DockQ statistics are provided in Appendix B.2.1.

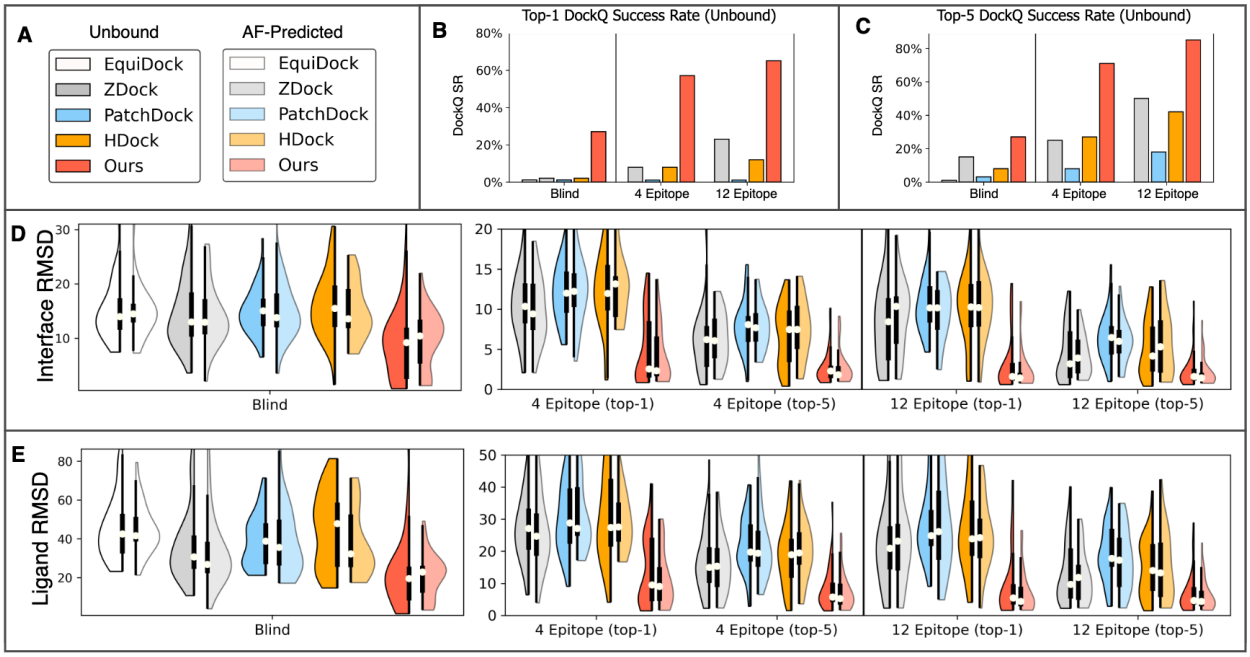

Figure 3.5: **Results for Antibody Benchmark predicted and unbound inputs**. (**A**) Legends to distinguish between the five methods and the target type (predicted or unbound) in plots (B–E). (**B**) and (**C**) show top-1 and top-5 success rates for each method on unbound targets, with no epitope residues provided (blind) and 4 or 12 epitope residues provided. (**D**) Split violin plots showing interface RMSD distributions for docking unbound (left half) and predicted (right half) chains given 0, 4, or 12 epitope residues. Each violin plot marks the median value with a white dot and shows the interquartile range with a bold vertical line. Both top-1 and top-5 distributions are shown when 4 and 12 epitope residues are provided. (**E**) Ligand RMSD distributions, in the same manner as (D).

In Figure 2C and 2D, our method obtains top performance in blind docking (i.e., no interfacial contacts or residues are provided as input), with considerably lower interface and ligand RMSD values than others. This holds regardless of whether unbound or predicted structures are used as input. This carries over to DockQ success rate where our method exceeds 25% for both input types. Since our method is deterministic, we only make a single prediction in the blind setting; thus, top-1 and top-5 success rates are the same. In an attempt to improve blind-docking results, we developed a genetic algorithm that exploits our method's ability to target different binding modes and predict docking quality. Details are given in Section 3.4.4 and examples are shown in Figure 3.12. This procedure increases both top-1 and top-5 success rates to 37.0% and 45.7%, respectively.

For blind docking, traditional methods improve significantly when top-5 predictions are considered. ZDock's top-1 predictions are successful for only one target, but this increases to 8 targets (17.4%) for top-5. Similarly, HDock improves median interface RMSD by roughly 5Å, from 15.6Å for top-1 to 10.8Å for top-5. EquiDock performs the worst of all five methods, with no DockQ successes for unbound or predicted targets. The method's poor performance is likely a result of excessive steric clashes. On average, EquiDock has 581 backbone atom clashes between antibody and antigen chains. In contrast, our method does not produce more than three atom clashes for any target. Clash distributions for our method and EquiDock, along with some example predictions, can be found in Figure B.1.

Compared to the blind setting, performance for all methods improves significantly when information of the antigen binding interface (epitope) is included. When four epitope residues are given, we reduce top-1 median interface RMSD from 9.2Å to 3.1Å. Top-1 ligand RMSD decreases accordingly, from 19.5Å to 9.5Å. For traditional methods, the RMSD reduction is less dramatic. The best-performing traditional method, ZDock, decreases top-1 interface RMSD from 12.8Å for blind docking to 10.4Å when 4 epitope residues are given. Even when binding interfaces are accurately predicted, traditional methods often fail to orient protein backbones properly. When 12 epitope residues are provided, the lower-quartile interface RMSD of ZDock is 3.6Å, but the same quartile ligand RMSD is 14.4Å for top-1 predictions. On the other hand, our method obtains 1.2Å and 3.5Å RMSDs, respectively.

Parallel to blind docking performance, top-5 predictions of the traditional methods yield significantly higher DockQ success rates than top-1 when epitope residues are included. Furthermore, traditional methods see substantial improvements on all metrics when more epitope residues are provided. HDock and ZDock obtain top-5 DockQ success rates of 47.8% and 56.5% with 12 epitope residues, but only 30.4% and 28.3% with four residues. This is likely a side-effect of the post-processing procedure, as increasing the number of epitope residues limits the size of the effective candidate space. In contrast, our method achieves a top-5 DockQ success rate of 71.7% with four epitope residues, increasing to 87.0% with 12.

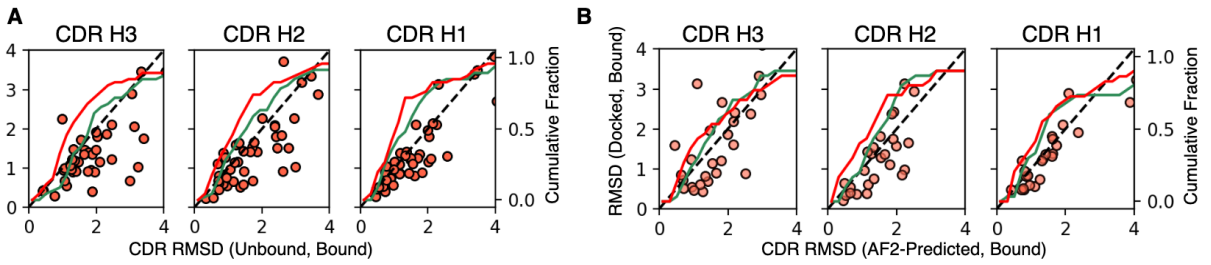

Figure 3.6: **RMSD of CDR Loop Regions**. (**A**) and (**B**) Scatter plots of heavy chain CDR RMSD between DockGPT predictions and the ground truth (bound) complex structure. Here, the x-axis shows RMSD between the input (unbound or AF2-predicted) heavy chain CDRs and corresponding segments in the ground truth complex structure. The y-axis shows RMSD between our predicted heavy chain CDRs and corresponding segments in the ground truth. Points below the $y = x$ axis correspond to targets having lower RMSD for our predicted complex structures. The cumulative fraction of targets with CDR-RMSD less than the corresponding x value is also plotted on a secondary axis using a red line for our predictions and a green line for unbound or predicted inputs. We provided our method with 12 residues on the antigen epitope for these plots.

This suggests that our method has learned to use binding site information as more than just a search filter.

Considering the relationship between binding interface quality and prediction accuracy, in Figure 2E, we consider the heavy chain CDR-RMSD distribution of our docked antibody structures. Here, we see that CDR loop conformations predicted by our method are closer to the ground truth than that of the unbound or AF2-Predicted input. Predicted heavy-chain CDR conformations have median RMSD 1.55Å, 1.39 Å, and 1.81 Å compared to 1.82 Å, 1.67 Å, and 1.94Å for the unbound input. The outcome is similar when starting from AF2-predicted input structures. This implies that our method goes beyond multidimensional scaling and actually learns to incorporate backbone flexibility in its predictions.

Results for docking bound antibody-antigen structures are radically different than those shown in Figure 3.5. When blind-docking bound chains HDock and PatchDock achieve DockQ success rates of 79% and 25%, respectively, for RAbD targets (see Table 3.1). If we fine-tune and evaluate our model on bound antibody-antigen chains, the blind-docking success rate increases more than two-fold to 62%. This implies that important information about

antibody-antigen binding interfaces is captured in the bound structures and highlights the importance of comparing docking methods on benchmarks containing unbound structures. When training and analyzing on bound inputs, we still provide only a coarse description of backbone geometry and do not consider input side-chain conformations. We hypothesize that more fine-grained features would significantly improve performance for bound targets. Interestingly, although EquiDock was trained on bound structures, the approach still underperforms on bound targets, with a success rate of 1.2%.

		Rosetta Antibody Design (Bound Targets)									
		I-RMSD↓			**L-RMSD↓**			**DockQ↑**			
Epitope	**Method**	Med.	Mean	Std.	Med.	Mean	Std.	SR	Med.	Mean	Std.
0	EquiDock	14.76	15.47	3.60	40.70	41.89	12.05	1.7%	0.02	0.03	0.03
	ZDock	5.43	8.35	8.01	14.22	22.05	24.01	50.0%	0.29	0.43	0.40
	PatchDock	11.33	10.33	6.78	26.96	29.00	22.95	25.9%	0.04	0.24	0.34
	HDock	**0.32**	**4.56**	9.01	**1.01**	**13.56**	26.52	**79.3%**	**0.98**	**0.77**	0.39
	Ours	1.79	5.75	7.60	5.0	16.7	22.0	61.7%	0.44	0.45	0.37
2		1.55	2.42	2.99	4.0	8.6	11.4	78.6%	0.62	0.56	0.32
4	Ours	1.20	1.87	2.10	3.5	6.9	8.3	82.1%	0.66	0.59	0.30
All		1.17	1.78	1.95	3.3	7.1	12.2	87.5%	0.73	0.64	0.27

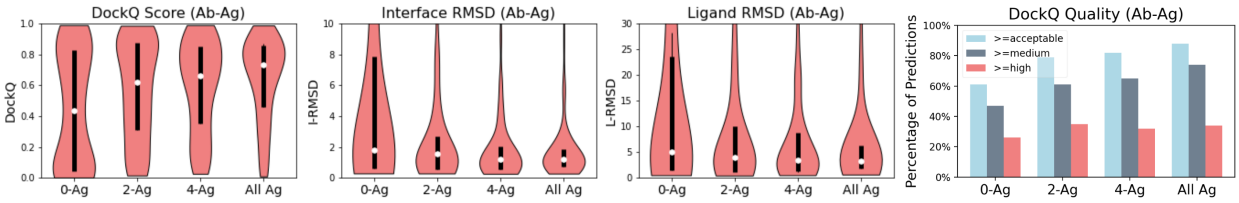

Table 3.1: **Docking results for Rosetta Antibody Design bound targets**. Results on the RAbD test set using bound chains as input to each docking method. Results for our method are generated after fine-tuning on bound antibody-antigen chains. The x-axis in the below four pictures shows the number of epitope residues provided to the docking methods. DockQ score cutoffs for acceptable, medium, and high-quality predictions are ≥ 0.23, ≥ 0.49, and ≥ 0.80

3.4.2 Results for DB5 Unbound and Predicted Targets

Results for DB5 targets are shown in Figure 3.7. Here, we focus mainly on performance when residue contacts are provided but also consider providing a limited number of interface residues on one or more chains. We chose to provide at most $C = 3$ inter-chain contacts

because, in theory, the number of rotational degrees of freedom for the ligand chain should be roughly $\max(0, 3-C)$ if the contacts are well distributed. More results for DB5 predicted and unbound targets are shown in Figures 3.7, 3.10 and 3.11 and appendix B.2.1, including tables with RMSD and DockQ statistics and performance on blind docking.

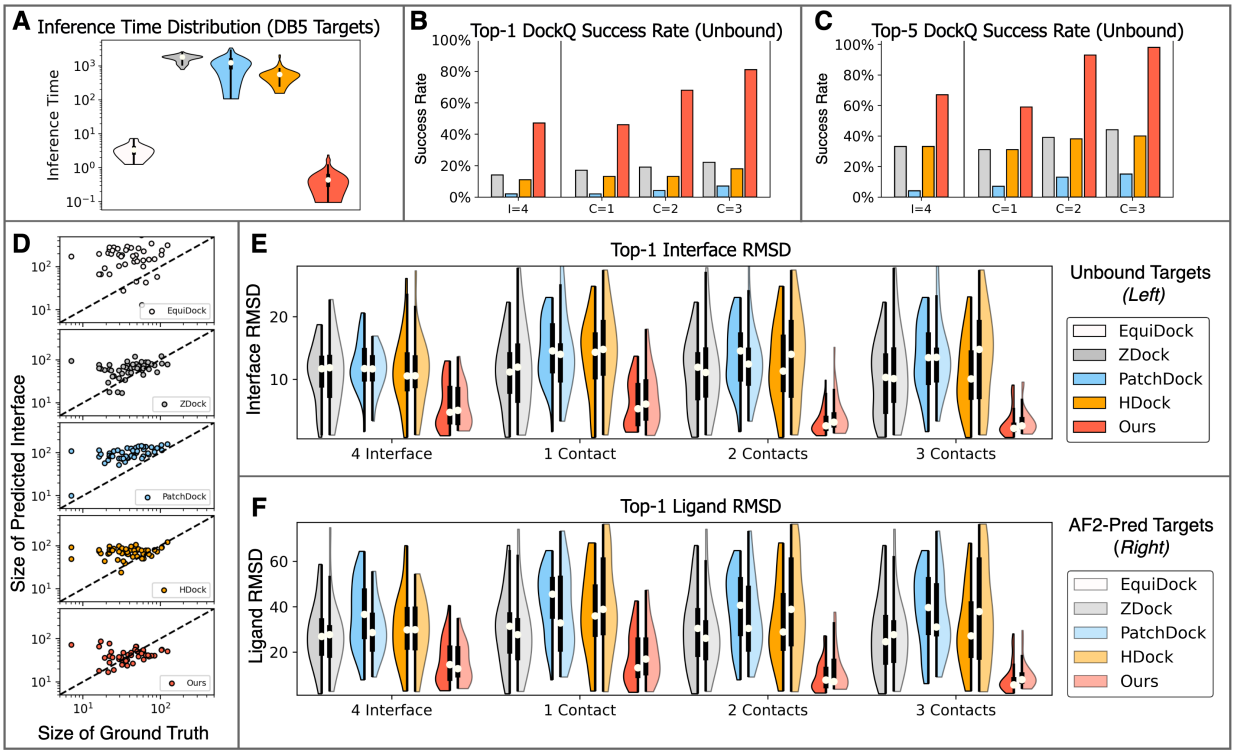

Figure 3.7: **Results for DB5 with AlphaFold2 predicted, and unbound monomer structures as input**. (**A**) Per-method distribution of inference times for docking DB5 unbound targets. (**B**) Bar plot of Top-1 DockQ success rates for DB5 unbound targets. Each method was given one, two, or three contacts ($C = 1, 2, 3$) or no contacts and four residues distributed over both the receptor and ligand binding interfaces ($I = 4$). (**C**) Bar plot of top-5 DockQ success rates, analogous to (B). (**D**) Scatter plot of the number of interacting residues in predicted complexes (y-axis) compared to the ground truth complex (x-axis). Blind docking predictions were made for all DB5 unbound targets, and interacting residues include only $C\alpha$ atoms, with a cutoff distance of 10Å. (**E**) and (**F**) Split violin plots of Interface-RMSD and Ligand-RMSD distributions as in Figure 3.5. In (B, C, E, F), we exclude Equidock because this method does not accept interface or contact information as input. Legends to distinguish between the five methods and target type are shown alongside RMSD distributions in (E) and (F).

For both unbound and AF2-predicted targets, supplying our model with a single contact

generates better top-1 median RMSD scores than traditional methods supplied with up to three contacts. When one contact is given, DockGPT achieves a top-1 DockQ success rate of 45.2%, and 59.5% for top-5. In contrast, ZDock and HDock have less than 20% success for top-1 and 33.3% for top-5. When two contacts are provided, DockGPT's top-5 predictions are correct for all but three targets. All targets have a correct top-5 prediction when three contacts are provided. On the other hand, the success rate of traditional methods improves only moderately, with a maximum top-5 success rate of 45.2% for ZDock across all settings.

On top of performance, our method also achieves significantly faster inference times than others, averaging inference times more than three orders of magnitude faster than ZDock,HDock, and PatchDock, and approximately six times faster than EquiDock.

As shown in Figure 3.7D, blind docking predictions for methods EquiDock, HDock, and PatchDock tend to have large binding interfaces, even when there are few contacts in the ground truth complex. The tendency is most pronounced for EquiDock, which regularly predicts receptor-ligand poses with large surface overlap. On average, EquiDock, predicts a binding interface size that is 5.4× larger than the ground truth. PatchDock, HDock average 2.9× and 2.3×, that of the native complex, respectively. In contrast, ZDock and our method are the least biased, averaging 1.9× and 1.4×, respectively.

The tendency to predict large binding interfaces may be explained by considering the objective functions of each method. PatchDock explicitly rewards large binding interfaces and high shape complementarity. HDock and ZDock both rank decoys by summing pairwise interfacial energy terms, and larger binding pockets may offer more potential for weak yet statistically favorable interactions. EquiDock, is trained to predict key points corresponding to the binding interface of each chain. It may be preferable from a loss perspective to predict key points near the chain's center of mass when the binding interface is hard to discern. In theory, a chain's center of mass offers a low-variance estimation of the true binding region. Finally, our method is trained with clamped FAPE loss, and thus all predictions that deviate beyond the clamp value are equally "bad" in a gradient sense. Adding violation loss also

ensures that the model is heavily penalized if surfaces intersect.

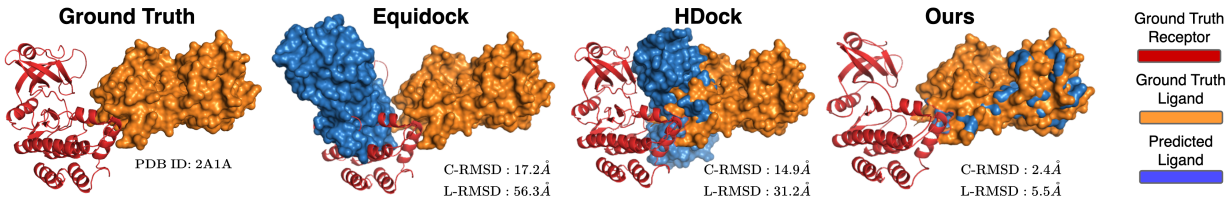

Figure 3.8: **Predictions for one DB5 target with unbound structure as input.** Ground-truth and docking predictions for PDB entry 2A1A. For each method, we show the surface of the predicted and ground truth ligand relative to the ground truth receptor. For this example, we selected traditional method HDock as it performed similarly or better than ZDock and PatchDock.

An example of the behavior described in the previous paragraph is shown in Figure 3.8. Although EquiDock's prediction is physically unrealistic, it still compares similarly to HDock in terms of interface and complex RMSD. EquiDock's prediction has an interface RMSD of 14.8Å, and a complex RMSD of 17.2Å, whereas HDock obtains 20.5Å and 14.9Å respectively. It is also clear that HDock predicts a large binding interface for this target, even though the true binding interface is relatively small. This example also highlights the importance of assessing ligand RMSD in addition to complex and interface RMSD.

3.4.3 Decoy Ranking with Predicted lDDT

In Section 3.3.6, we mention that predicted complexes are ranked by predicted interface lDDT (I-plDDT). By selecting the target with the highest I-plDDT score, we are able to improve DockQ success rate, and RMSD scores compared to random sampling (see Figures 3.9, 3.10 and 3.11). We now describe this procedure in more detail. We define the predicted binding interface as the set of residues having at least one predicted inter-chain contact with $C\alpha$ distance between predicted coordinates less than 10Å. Formally, it is the set:

$$\{i \in \mathcal{C}_1 \cup \cdots \cup \mathcal{C}_k \ : \exists j \notin \mathcal{C}(i), \|\vec{\boldsymbol{x}}_i^{C\alpha} - \vec{\boldsymbol{x}}_j^{C\alpha}\|_2 < 10\}, \tag{3.17}$$

where $\vec{\boldsymbol{x}}_i^{C\alpha}$ are predicted $C\alpha$-coordinates.

The ground-truth binding interface is defined analogously with respect to coordinates taken from the ground truth complex. To rank decoys for a given target, we average the per-residue plDDT for those residues on the predicted binding interface. This differs from the standard predicted lDDT, which averages over the entire complex. For a given residue, we compute the predicted plDDT score as an expectation over the bin labels used for lDDT loss (Section 3.3.4) with respect to the softmax of predicted logits.

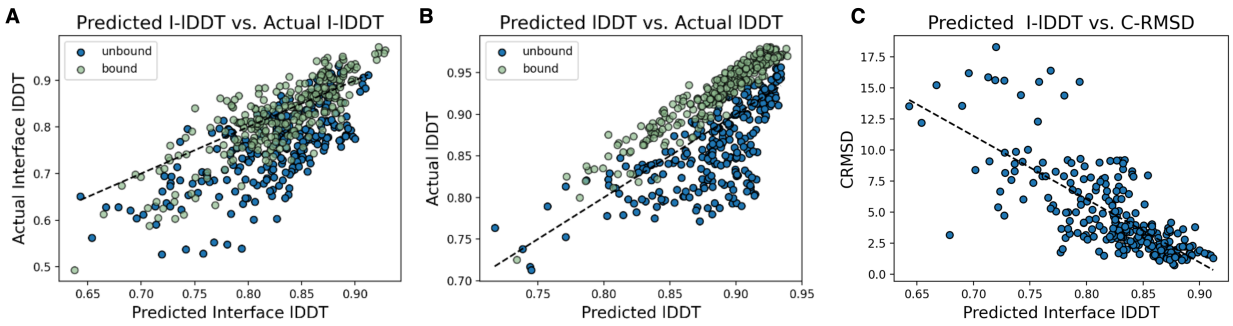

Figure 3.9: **Analysis of lDDT predictions**. Each plot shows results for predictions made on DB5 bound or unbound input chains, providing the model with four interface residues and four contacts sampled at random from the ground truth complex. Each dot represents a decoy generated from bound or unbound input chains. A total of five decoys were generated for each target. Correlation coefficients for predictions derived from unbound and bound targets are denoted with ρ_u and ρ_b, respectively. (**A**) scatter plot of predicted lDDT (x-axis) for the predicted binding interface against actual lDDT (y-axis) for the ground truth binding interface. Unbound targets are shown in blue ($\rho_u = 0.69$), and bound targets are shown in green ($\rho_b = 0.83$). We remark that the predicted and actual interfaces may differ. (**B**) Scatter plot of predicted lDDT (x-axis) and actual lDDT (y-axis) for bound and unbound targets ($\rho_u = 0.70$, $\rho_b = 0.95$). (**C**) Scatter plot of predicted lDDT using the predicted binding interface against the complex RMSD of the predicted structure ($\rho_u = -0.74$).

Figure 3.9 shows scatter plots of plDDT and i-plDDT for DB5 bound and unbound targets. In plot (C), we find a strong correlation between I-plDDT and complex RMSD for unbound targets, suggesting that this quantity is effective for ranking decoy structures. We explore this further in Figure 3.10, which compares the complex RMSD (A) and interface RMSD (B) distributions of decoys selected by plDDT (orange) and the same distributions computed over all decoys (blue). In this figure, we generate five decoys per target and assess

them across 12 binding site settings, varying the number of provided contacts or interface residues in each setting. Mean, and median RMSD scores for selected decoys are lower across all binding site contexts. RMSD distributions of decoys selected by interface plDDT are also consistently more concentrated at lower values.

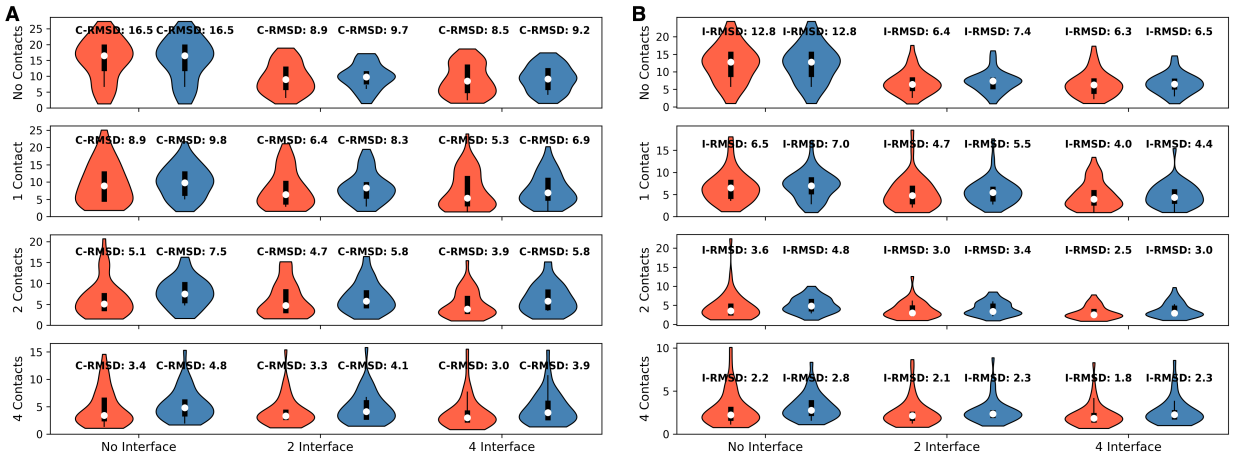

Figure 3.10: **Selection overview for DB5 unbound targets (without recycling iterations)**. For this experiment, we generate five decoys for each target using a reduced model (no side-chain prediction, no recycling). Each row/column corresponds to a number of provided contacts/ interface residues. This information is derived as a random sample from the native conformation. For each violin plot, we compare the complex RMSD (C-RMSD, (**A**)) or Interface RMSD (I-RMSD, (**B**)) of all predictions (blue) against the prediction for each target having the highest predicted interface plDDT (orange). We remark that results in the two plots use only $C\alpha$ atoms to compute each RMSD type, and as such, may differ slightly from the results reported in other sections.

Last, we consider our model's ability to predict conformation changes upon binding. In Figure 3.11 (A, B), we see that the chain-wise RMSD between predicted and unbound structures is similar for all but a handful of targets. In terms of interface RMSD, predicted structures are slightly more similar to the bound conformation, especially when there are larger discrepancies in the interface of aligned bound and unbound structures.

Unfortunately, the conformation similarity between DB5 bound and unbound structures is relatively high, and more diverse structures should be examined before drawing conclusions from these results; nevertheless, in Figure 3.11 (C and D), we consider a case study on

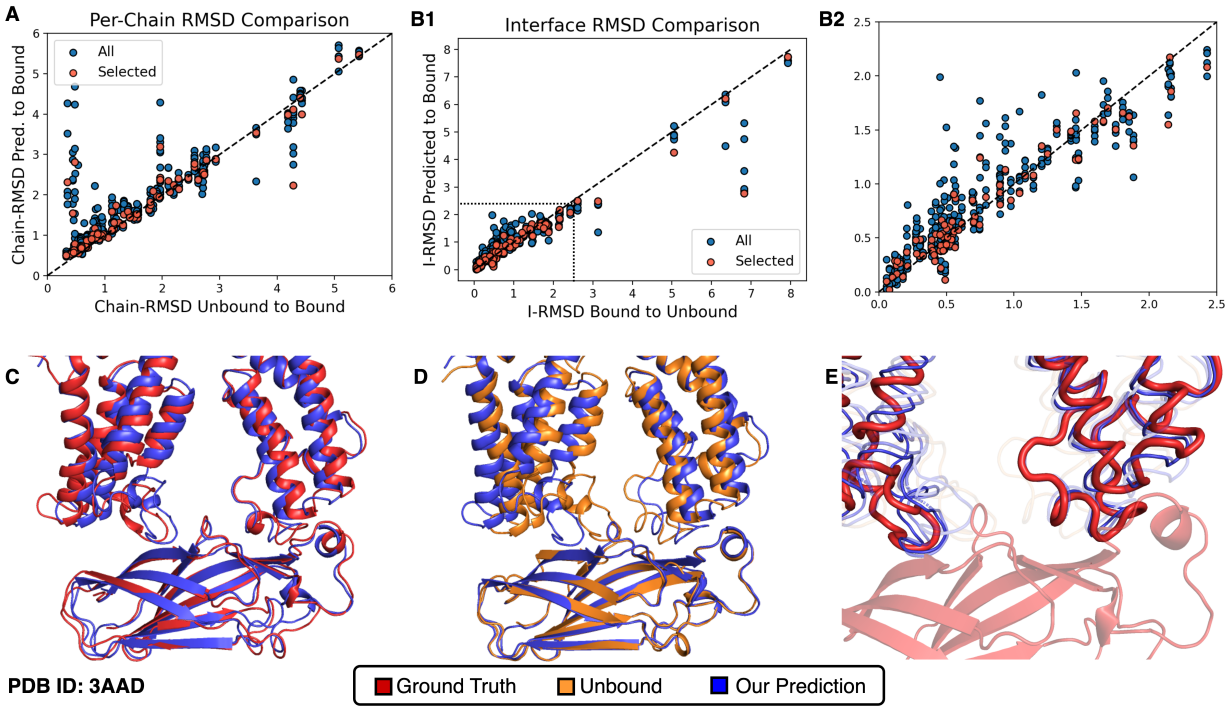

Figure 3.11: **Examination of conformational flexibility for DB5 unbound targets**. As in Figure 3.9, we generate five predictions per DB5 target, using unbound chains as input to our model. For each prediction, we provide our model with three contacts sampled at random. (**A**) Scatter plot of receptor/ligand chain-wise RMSD between bound and unbound chains (x-axis) against predicted and bound chains (y-axis). Red dots show the decoy with the highest predicted interface pLDDT for each target. (**B1**) Shows the interface RMSD in the same manner as (A). (**B2**) zooms in on the 0-2.5Å range of (B1). (**C**–**E**) Cartoon representations of our prediction, bound, and unbound chains for DB5 target 3AAD. (**C**) Our top-ranked prediction for DB5 target 3AAD using unbound chains as input is shown in blue, and the bound conformation in red. (**D**) Cartoon representations of our top-ranked prediction (blue) and unbound chains (orange) for target 3AAD. For this image, unbound chains are optimally aligned to respective bound chains using a chain-wise Kabsch alignment. (**E**) Our model's top-3 ranked predictions for 3AAD are colored by predicted interface lDDT. Lower transparency is used to denote lower predicted interface-LDDT. For this target, the RMSD between bound and unbound receptor chains (top, helices) is 4.18Å, and 2.05 Å for the ligand chain (bottom, sheets). When bound and unbound chains are optimally aligned, the interface RMSD is $\approx$ 6.8Å. Our top-ranked prediction obtains an interface RMSD of 2.6Å.

PDB entry 3AAD, where our model predicts a conformation diverging significantly from the unbound state. For this target, our model with highest predicted interface lDDT has interface RMSD 2.6Å, whereas an optimal alignment mapping the unbound chains to the

bound complex has interface RMSD 6.8Å. Moreover, our model predicts a conformation for the helical receptor chain that is only 2.2Å from that of the bound conformation, compared to 4.2 for unbound-bound conformation. We remark that the maximum sequence identity between target 3AAD and any training example is only 9%.

3.4.4 Genetic Algorithm for Protein-Protein Docking

Although our method is deterministic, sampling can still be performed by providing different subsets of inter-chain contacts or interface residues for the same example. If confidence predictions correlated perfectly with complex RMSD (see Figure 3.9), we could discern the native complex by enumerating all subsets of inter-chain contacts as input to our algorithm. Inspired by this idea, we develop a genetic algorithm to guide complex predictions toward high-confidence binding modes. This approach significantly improves Antibody-Antigen prediction quality and enables us to predict ensembles of conformations in the absence of binding site information. We describe this approach throughout the remainder of this section.

To sample conformations in the absence of interfacial residue and contact information, Any genetic algorithm consists of four main components: (1) a genetic representation of the solution domain, (2) a "fitness" function to assess population quality, (3) a mutation function that alters representations, and (4) a crossover function that combines two representations. Given an initial population of solution candidates, the algorithm then proceeds to produce new "generations" by assessing the fitness of each candidate and stochastically selecting those with favorable fitness to combine or mutate. We describe each component of our algorithm below.

Solution Representation Solutions are represented as a binary vector of interface residues. The length of this vector is $L_{rec} + L_{lig}$ where L_{rec} is the length of the receptor chain, and L_{lig} is the length of the ligand chain. Each position of the vector corresponds to a residue in one of the chains, and a one at position i is meant to indicate that this residue i is part

of the binding interface.

Initial Population To generate initial candidates $\{X_0^{(0)}, \ldots, X_{n^{(0)}}^{(0)}\}$, we randomly sample a single residue on the surface of receptor and ligand chains, and provide these two residues as the "interface-residue" feature. For antibodies, we restrict the sampling to residues in CDR H1-3 loops. Random surface residues are chosen by scaling a 3-dimensional Gaussian (direction), to the maximum distance between any two residues in the protein and then choosing the residue closest to this point.

Fitness Function To evaluate the fitness of each candidate, we use the candidate solution as the binding interface feature and then compute a function of predicted interface-pLDDT on the output. We choose $f(X, t) = \exp[(t{+}1){\cdot}(\text{I-pLDDT}(X))]$ where t is a scaling parameter (chosen ad hoc as one plus the index of the current iteration).

Mutation Function Given a set of solution candidates, $\{X_1^{(1)}, \ldots, X_{n^{(1)}}^{(1)}\}$ and corresponding structures generated at time t, we select a subset of $n = n^{(t+1)}$ with replacement according to the fitness function $f(\cdot, t)$, and randomly sub-sample six residues on the predicted binding interface. We choose to sample a fixed number here because we empirically found that predicted interface lDDT scores have a modest correlation with the number of interface residues provided as input.

Example predictions for PDB target 2YVJ are shown in Figure 3.12. For this target, the decoy with the highest predicted interface lDDT has a low DockQ score, but the third ranked prediction has a DockQ score above 0.7. Results for Ab-Bench and DB5 targets are given in Tables 3.2 and B.3 to B.8. To generate results, we run this procedure for ten generations, using an initial population size of 50 and subsequent population sizes of 25.

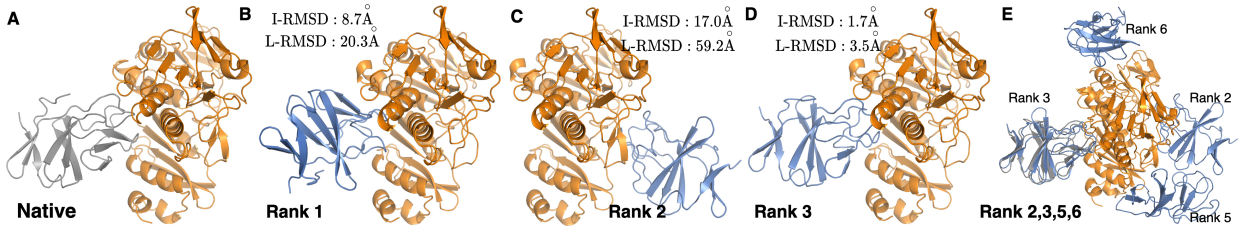

Figure 3.12: **Genetic Algorithm Explores Diverse Binding Modes** Ground truth and example predictions from our genetic algorithm for DB5 target 2YVJ. In all sub-figures, the ground truth receptor is shown in orange, the bound ligand in gray, and our predictions in blue. (**A**) The bound complex of DB5 Target 2YVJ. (**B**–**D**) the top three ranked predictions using our genetic algorithm. (**E**) Rank 2, 3, 5, and 6 predictions from our genetic algorithm. Rank 1 and 4 predictions are omitted for visual clarity, as they clash with other predictions. The bound ligand is also shown in gray. Although our method fails to generate an accurate top-1 prediction, our third-ranked prediction successfully docks to the same interfacial region.

3.4.5 Comparison to AlphaFold-Multimer

We compare our method with AlphaFold-Multimer in the blind docking setting on DB5 and Ab-Bench benchmarks described in Section 3.3.6. In addition to comparing the two methods directly, we also include a hybrid approach (Ours + AF). For this approach, we provide our method with up to three randomly sampled residues from antibody-antigen binding interfaces predicted by AlphaFold-Multimer. No information on native complexes is used. We generated 100 decoys for each target and selected the decoy with highest predicted interface lDDT as our final prediction (selection as described in Section 3.4.3). The results are shown in Table 3.2.

Motivating the hybrid approach, we analyzed binding site information extracted from AlphaFold-Multimer predictions (Figure 3.13). As expected, AlphaFold-Multimer recovers the antibody paratope with high precision. Perhaps more surprisingly, we see that at least part of the antigen epitope is recovered with relatively high precision but lower recall. Noticing this, we conjectured that our results might be improved by sampling a limited number of predicted binding modes and ranking predictions.

Our blind docking (i.e., our deep learning plus our genetic algorithm) greatly outperforms

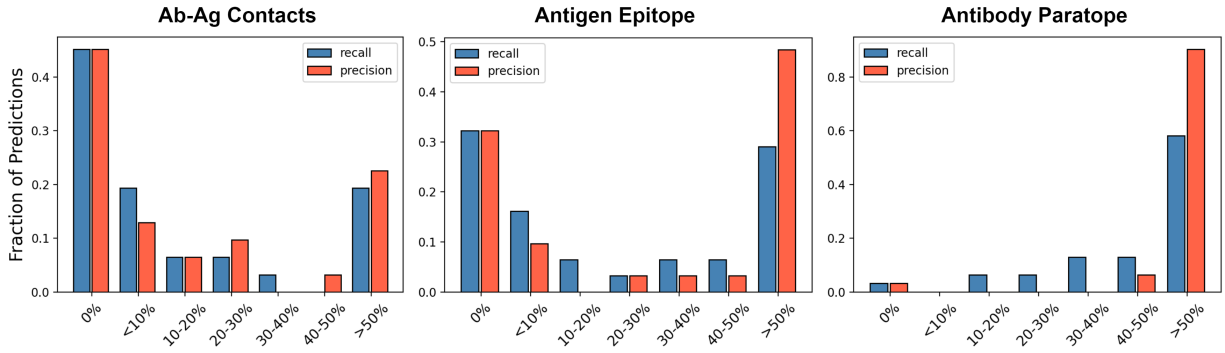

Figure 3.13: **Binding site precision and recall for AlphaFold-Multimer on Ab-Bench targets**. Histograms of binding site precision and recall for AlphaFold-Multimer predicted structures on Ab-Bench targets. Recovered contacts, antigen binding interface (epitope), and antibody binding interface (paratope) are shown from left to right.

AF-Multimer on antigen-antibody complex structure prediction without using any binding site information. But on general protein targets, our method performs poorly. Adding AF-Multimer predicted interface or contact information significantly improves prediction quality since this indirectly makes use of MSA information. We hypothesize that directly including MSA information could significantly improve prediction quality for general proteins, especially in conjunction with our genetic algorithm, as model confidence predictions correlate strongly with predicted interface plDDT, but we leave this study for future work.

Recently, Yin et al. [Yin+22] benchmarked AlphaFold-Multimer and other docking programs on antibody-antigen and general protein targets using the sequences or structures of unbound chains. This study found that AlphaFold-Multimer performs very poorly for antibodies, successfully predicting only 11% of targets. In their study, the authors identified sequence and structural features associated with the lack of AlphaFold success and attributed the performance gap to a lack of co-evolutionary signal. For antibody-antigen complexes, they found that the success rate of AlphaFold-Multimer was not much different when the model was given only templates and no MSA information. In this setting, AlphaFold-Multimer is similar to our model. We hypothesize that our performance improvement for antibody-antigen targets comes from (1) fine-tuning and (2) no MSA inputs. Since

	DockQ↑	I-RMSD↓			L-RMSD↓		
	SR (%)	**25th**	**50th**	**75th**	**25th**	**50th**	**75th**
	Antibody Benchmark (Top-1)						
AF-Mult.	28.3%	1.9	9.3	14.7	12.2	22.6	36.0
Ours	26.1%	2.5	9.2	**12.1**	8.2	19.5	25.4
Ours+GA	**37.0%**	**1.8**	**8.3**	12.4	**5.5**	**19.2**	26.4
Ours+AFM	28.3%	1.9	10.1	13.3	5.7	20.1	27.8
	Antibody Benchmark (Top-5)						
AF-Mult.	34.8%	1.8	5.8	13.1	9.2	18.3	26.4
Ours+GA	**45.7%**	**1.7**	**4.0**	**7.3**	**4.9**	**11.5**	**19.8**
Ours+AFM	37.0%	**1.7**	4.4	8.9	5.4	11.3	19.0
	Docking Benchmark Version 5.5 (Top-1)						
AFM	**50.0%**	**0.9**	7.9	16.4	**2.8**	19.6	35.2
Ours	7.1%	8.9	13.3	17.4	24.2	35.4	49.5
Ours+GA	9.5%	9.7	14.0	17.5	23.1	33.4	47.5
Ours+AFM	42.8%	2.7	**5.7**	**14.3**	6.6	**17.4**	**28.7**
	Docking Benchmark Version 5.5 (Top-5)						
AFM	50%	**0.9**	**4.7**	13.0	**2.6**	**12.2**	30.2
Ours+GA	16.7%	5.4	8.8	13.6	12.9	20.7	34.3
Ours+AFM	**52.4%**	2.0	5.1	**12.7**	5.2	12.4	**24.5**

Table 3.2: **Comparison of DockGPT and AlphaFold-Multimer on two docking benchmarks** Results for AlphaFold-Multimer (AFM), our method (ours), our method with genetic algorithm (Ours+GA), and our method using AlphaFold-Multimer predicted interfaces (Ours+AFM) for Ab-Bench and DB5 benchmarks. AlphaFold-Multimer outperforms our method on blind docking general protein targets from DB5. Our method does not make use of MSA information, which is especially important for general proteins where binding interfaces are harder to discern. For antibody complexes, the paratope is limited to CDR loops, and our method has an easier time predicting the complex.

we do not train with MSA information, our model is forced to learn sequence and structural features which facilitate good binding modes. This is particularly useful for immunoglobulin targets, as antibody-antigen interfaces are less likely to have co-evolving sequences available for MSA generation [Yin+22].

While AlphaFold-Multimer often predicts correct conformations for antibody and antigen chains, the predicted complex can deviate far from the ground truth. For example, Figure 3.14A shows that although the complex structure is far from the ground truth, the antibody and antigen structures are highly similar to their respective bound counterparts, with less than 2Å complex-RMSD between predicted and unbound antibody chains, and

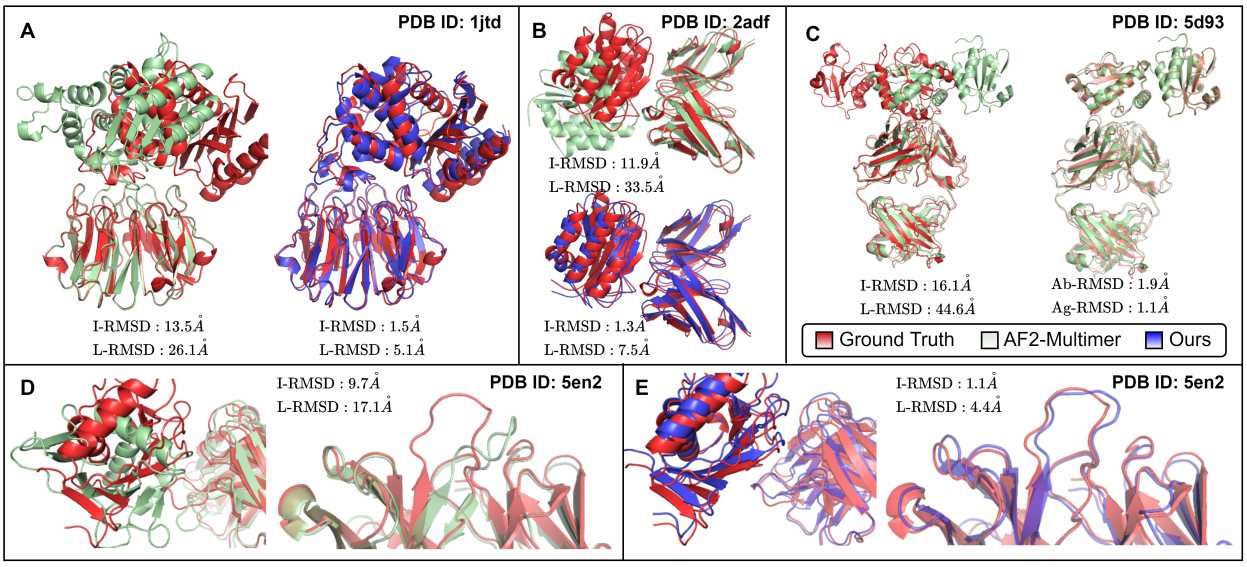

Figure 3.14: **Comparison of structure predictions between DockGPT and AlphaFold-Multimer**. In this figure, all predictions from our model were made with AlphaFold2 or AlphaFold-Multimer predicted structures as input. (**A**) Predictions for DB5 target 1JTD. Our method uses one random contact. (**B**) Predictions for RAbD target 2ADF. Our prediction uses four randomly chosen epitope residues. (**C**) Example of high interface and ligand RMSD for an antibody-antigen complex predicted by AlphaFold-Multimer (left). Alignment of predicted chains to the ground truth structure (right). (**D**,**E**) Another example where AlphaFold predicts correct chain conformations but an incorrect complex. Supplying our method with antigen epitope residues predicted by AlphaFold improves complex prediction quality (left) and CDR loop RMSD (right)

1.1Å RMSD between predicted and bound antigen chain. In Figure 3.14(A,B,D), we provide more examples illustrating this and also show how our model can be used in conjunction with AlphaFold to improve prediction quality when binding site information is known.

3.5 Ablation Study

We trained several ablated models to identify how different components of our architecture and training procedure contribute to docking performance. We show results for four additional models in Table 3.3.

We find that removing the shared weight layers and auxiliary FAPE loss from our structure decoder leads to the largest degradation in performance. We also remark that ablating

the centrality encoding or adding a secondary structure encoding to our input residue features had an insignificant impact on performance. We remark that including ESM1b [Riv+19] encodings (+ ESM1b) of each chain did not noticeably improve performance in the blind docking setting. We obtain DockQ scores ≥ 0.23 for three targets when ESM1b encodings are used and two targets when the encodings are removed. These encodings do not significantly improve performance, so we opted for the simpler model. Interestingly, the variant of our model that does not use recycling can still obtain competitive top-5 performance but suffers in top-1 performance. Recycling decoder residue features is also competitive with the baseline recycling implementation but does not result in significantly better performance.

	Top-1			**Top-5**		
	I-RMSD	L-RMSD	SR(%)	I-RMSD	L-RMSD	SR(%)
			4 Interface			
Baseline	**4.7**	14.5	**47.6%**	**3.0**	**8.6**	69.0%
No Recycle	8.4	20.0	29.7%	3.9	8.5	64.9%
+ ESM1b	5.1	13.8	**47.6%**	3.3	8.8	**73.3%**
No Share Wts.	4.9	16.6	42.9%	3.9	9.6	54.8%
Recycle Dec.	5.3	**13.7**	42.9%	3.4	8.8	**73.3%**
			1-Contact			
Baseline	5.8	**13.6**	**45.2%**	3.5	10.2	59.5%
No Recycle	8.0	20.0	33.3%	4.1	11.5	51.4%
+ ESM1b	**5.6**	18.1	37.7%	3.6	**9.5**	**62.2%**
No Share Wts.	7.2	22.1	31.0%	3.7	11.4	45.2%
Recycle Dec.	5.7	18.0	37.7%	**3.3**	**9.5**	59.5%

Table 3.3: **Ablation Study** We consider the top-1 and top-5 performance of model variants on DB5 unbound targets using one contact or four interfacial residues as input. This information is randomly sampled independently for each variant, and 15 decoys are generated for each target. Predicted IplDDT is used to rank each decoy. The baseline model is described in the main text. For the four variants, we considered removing recycling (No Recycle), adding ESM1b encodings of chain sequences as input (+ESM1b), learning separate weights for each decoder block (No Share Wts), and recycling decoder residue features, rather than encoder residue features (Recycle Dec.). When learning separate weights for decoder layers, we also remove auxiliary FAPE loss.

3.6 CDR-Loop Design

We now describe how DockGPT can be extended to perform sequence/structure co-design. As a proof of concept, we consider the task of CDR-loop generation, focusing on heavy chain CDRs H1-H3. We note that our method also designs light chain CDRs, but we omit this for brevity.

First, we describe the modifications made to DockGPT, which enable *de novo* design. We then show how context such as pairwise atom distances, or even atomic coordinates can be incorporated into the design task. Finally, we compare our approach to four other CDR loop design algorithms and analyze the factors contributing to our success.

Approach To perform joint imputation of sequence and complex structure, we re-frame the design task as a data imputation problem and develop a masked autoencoder framework. Unlike the original framework proposed by He et al. [He+21], we do include mask tokens in our encoder. This provides a more natural way to incorporate *partial context* such as partial structure and missing sequence, or known secondary structure and missing tertiary structure. It is also easy to incorporate partial context with our input featurization (Section 3.3.1) which consists almost exclusively of one-hot encodings. To enable masking, we simply augment each encoding with an extra bin, $\langle \mathsf{MASK} \rangle$, for mask values.

Implementation We use the same features as described in Section 3.3.1, except for residue degree centrality. We exclude degree centrality since this information is unlikely to be available for most *de novo* design tasks. We add one additional residue feature; a one-hot encoding of secondary structure using three classes for sheets, helices, and loops. We encode all CDR residues as loops during inference and do not mask this feature during training. We found that the secondary structure encoding improved convergence when transitioning from pre-training to fine-tuning on antibody structures. During pre-training, we masked linear segments of a randomly selected chain, sampling the segments based on proximity to the

chain's binding interface. The process is detailed in Algorithm 8, where we consider the case of two interacting chains, though it is straightforward to extend to general complexes. We remark that the receptor and ligand labels in Algorithm 8 are arbitrary. We randomly select one chain as the receptor and the other as the ligand before performing this procedure. The length r of the masked segment is selected from a geometric distribution as geom $\left(\frac{1}{15}\right)$. For each residue in the chosen segment, we replace the corresponding features with separate $\langle\mathsf{MASK}\rangle$ parameters except for relative sequence position and relative sequence separation. To be clear, no inter-atom distance or orientation information is given for masked residues.

Algorithm 8 Select Masked Region

Input
  $\{\vec{\boldsymbol{x}}_i\}$: $C\alpha$ coordinates for protein dimer
  ℓ: Length of region to mask
  $\{\mathcal{C}_k\}$: Chain indices for receptor and ligand ($k = 1, 2$)
Output
  r: $[r, r+\ell)$ segment to mask in ligand chain.

function SelectRegionMask($\{\vec{\boldsymbol{x}}_i\}, \ell, \mathcal{C}_1, \mathcal{C}_2,\ \sigma = 4$) :
  $\{\vec{\boldsymbol{x}}_i^L\}, \{\vec{\boldsymbol{x}}_j^R\} = \{\vec{\boldsymbol{x}}_i : i \in \mathcal{C}_1\}, \{\vec{\boldsymbol{x}}_j : j \in \mathcal{C}_2\}$
  $D_{ij} = \|\vec{\boldsymbol{x}}_i^L - \vec{\boldsymbol{x}}_j^R\|_2$
  $S_{ij} = \frac{1}{1+D_{ij}^2/4}$
  $s_i^{(L)} = \sum_j S_{ij}$
  choose $r \in [0, |\mathcal{C}_1| - \ell]$ w.p. $\propto \exp\left[\frac{\sum_{c=1}^{\ell} s_{r+c}^{(L)}}{\sigma}\right]$
  return r

The methods we compare with (see Table 3.4) are trained, validated, and tested on different datasets. Code has not been released for all of these methods, and because of this, we tried to replicate their training and testing procedures as accurately as possible. To generate our data, we use the scheme proposed in Jin et al. [Jin+22], using the SAbDab dataset described in Section 3.3.5, generating CDR-clusters at 40% sequence identity and using an 8:1:1 split for training, validation, and test sets, respectively. Some example generations are shown in Figure 3.15

3.6.1 Incorporating Coordinates

Here, we describe how fixed, flexible, and rigid coordinates can be incorporated into the masked-design framework while still maintaining SE(3)-equivariance. We note that, by using IPA, our decoder is already equivariant to any global rigid transformation applied to per-residue local frames [Jum+21, Suppl. Material, 1.8.2]. Moreover, the same proof shows that scalar node features are invariant to any global rigid transformation. Thus, setting rigids $T^{(0)}$ in the decoder submodule to those derived from a complete set of backbone coordinates results in an SE(3)-equivariant update to the rigid frames and an invariant update to the scalar features.

We now argue a more general claim: that IPA can be made SE(3)-equivariant even when some input coordinates are rigid (fixed), flexible, or missing. Let $C = C_{\text{fixed}} \cup C_{\text{flexible}} \cup C_{\text{missing}}$ be a partition of the input residues $i = 1, \ldots, L$ denoting those residues with coordinates which should remain fixed, those which are flexible, and those with coordinates that are missing. Without loss of generality, assume that the coordinates which are not missing have mean $\vec{\mathbf{0}}$. We propose to initialize all missing coordinates at the origin.

Intuitively, mean-centering the complex implies that missing coordinates are viewed equivalently in the global and local frames of every fixed and flexible residue at initialization; since they are initialized at the origin. Note that any global rotation applied to the input points will leave the origin fixed, and thus only the fixed or flexible coordinates will change position. Since IPA is equivariant to global rotations, the claim follows directly from the equivariance of IPA. That is, applying a global rotation to all residue coordinates - while keeping the scalar embeddings fixed - will result in an equivalent update to the local frames.

To keep the coordinates corresponding to residues in C_{fixed} static, we modify the update in Equation (3.9) to

$$T_i^{(\ell+1)} = \begin{cases} T_i^{(\ell)} & i \in C_{\text{fixed}} \\ T_i^{(\ell)} \circ \textbf{RigidUpdate}\left(\boldsymbol{x}_i^{(\ell+1)}\right) & \text{otherwise} \end{cases} \tag{3.18}$$

From the equation above, it's clear that transformations corresponding to fixed positions ($i \in C_{\text{fixed}}$) remain fixed in the output. This implies that

$$\left[\vec{\mathbf{0}}\right]_{T_i^{(\ell)}} = \left[\vec{\mathbf{0}}\right]_{T_i^{(0)}} = \vec{\boldsymbol{t}}_i^{(0)} = \vec{\boldsymbol{x}}_i^{C\alpha,\text{input}} \qquad \forall i \in C_{\text{fixed}}. \tag{3.19}$$

This maintains local consistency and allows the network to learn the placement of missing atoms with respect to a fixed set of reference frames. To fix coordinates, we simply replace the prediction of $\vec{\boldsymbol{x}}_i^a$ ($i \in C_{\text{fixed}}$) in Equation (3.10) with the coordinates given as input.

For practical reasons, mean-centering *all of the input coordinates* does not actually result in an equivariant update – this is because the rigid frames use a specific atom (e.g., $C\alpha$) to initialize their translation. Thus, in practice, only the rigid translations should have mean zero.

3.6.2 Results

Evaluation Metrics

We provide a comparison with four other protein design frameworks tailored towards antibodies, RefineGNN[JX21], CoordVAE[LMX22], AR-GNN [JBJ20], and LSTM [Sak+21; Akb+22]. In Tables 3.4 and 3.5, we report native sequence recovery (NSR) and perplexity as described in Chapter 2, Section 2.3.2. When reporting CDR RMSD, we use the same methodology as described in Section 3.3.6. To generate our method's results in Tables 3.4 and 3.5, we provide four native antibody-antigen contacts and produce five decoys per target. The decoy with the highest predicted interface plDDT is selected for comparison.

Loop Design with Framework Distance as Context

In Table 3.4, we show design results for our method using coarse distance and orientation information as input. Despite receiving low-resolution descriptions of antibody and antigen

Method	Structure Prediction				Sequence Prediction				
	RMSD↓				PPL↓			NSR↑	PPL↓
	H1	H2	H3	Fr	H1	H2	H3	H1-H3	
(Ours+FT)	<u>1.11</u>	1.02	**1.88**	0.72	**4.46**	<u>6.71</u>	10.68	**39.7%**	7.67
CoordVAE	**0.96**	<u>1.00</u>	<u>1.95</u>	–	–	–	–	–	–
Refine-GNN	1.18	**0.87**	2.50	–	<u>6.09</u>	<u>6.58</u>	**8.38**	35.4%	–
AR-GNN	2.97	2.27	3.63	–	6.44	6.86	<u>9.44</u>	23.9%	–
LSTM	–	–	–	–	6.79	7.21	9.70	22.5%	–

Table 3.4: **Results for CDR-loop design.** Performance of our method and four others, CoordVAE [LMX22], Refine-GNN [Jin+22], AR-GNN [JBJ20; Jin+22] and LSTM [Sak+21; Akb+22], on the task of predicting CDR H1-H3 loop conformation and sequence. For our method, "FT" denotes fine-tuning on antibody structures. The columns H1-H3 show the $C\alpha$-RMSD of heavy chain CDR H1-H3 between predicted and native structures. For our method, we also list the RMSD of the predicted and bound framework regions under column "Fr". Perplexity (PPL) of sequence predictions for each CDR loop are shown in separate columns. Finally, overall perplexity and native sequence recovery across all loop regions are shown in the rightmost columns. We note that AR-GNN and Refine-GNN predict sequence and structure for each CDR loop region separately while conditioning on the native sequence and structure of the other CDR regions. This may result in slightly lower perplexity for these models.

chains, we can still recover antibody framework regions with sub-Angstrom RMSD. Furthermore, none of the four other methods can design CDR loop regions in the presence of an antigen; for these methods, the sequence and structure generation results in Table 3.4 are generated on bound heavy-light chains, with the bound antigen omitted. In contrast, our method simultaneously docks and designs all six heavy and light chain CDR loops and sequences simultaneously.

Loop Design with Coordinates as Context

In settings such as CDR-loop generation, fixing the heavy and light chain framework regions may be practically useful. To enable *de novo* design of loop regions, the CDR L1-L3 and

H1-H3 segments can simply be treated as missing. To test whether this approach works in practice, we fine-tuned the same pre-trained model from Section 3.6 while supplying the coordinates of the heavy and light chain framework regions to the structure-decoder module. The framework coordinates are treated as rigid during inference, and the rest of the procedure is implemented exactly as described in Section 3.6. Of course, it is also possible to provide the coordinates of the docked antigen complex and the framework. For example, coordinates on or surrounding the epitope may be treated as flexible and the others as rigid depending on the use case. We omit this setting here as the manuscript focuses primarily on protein docking, and the former is a significantly simpler task.

Our Method	**Structure Prediction**				**Sequence Prediction**				
	RMSD↓				**PPL↓**			**CDR H1-3**	
	H1	H2	H3	Fr	H1	H2	H3	NSR	PPL
No FT	1.43	1.53	2.49	0.55	4.84	7.53	11.17	37.5%	8.41
FT	1.11	1.04	1.88	0.82	4.46	6.71	10.68	39.7%	7.67
FT + Fr-Coord	1.03	0.98	1.78	–	4.27	6.50	10.36	40.6%	7.18

Table 3.5: **CDR-loop design with framework coordinates** Results from our method without framework coordinates and fine-tuning (No FT), without framework coordinates and with fine-tuning (FT) and with fine-tuning and coordinates for antibody heavy and light chain framework regions (**FT + Fr-Coord**). The same criteria and results from our method described for Section 3.6 are used here.

Fine-tuning our model on SAbDab reduces overall sequence perplexity ($p = 0.086$) and CDR-RMSD ($p < 0.005$ for CDR H1-H3). We remark that including framework coordinates reduces median CDR H1-H3 RMSD and sequence perplexity, but hypothesis tests comparing our fine-tuned models with and without framework coordinates do not support this claim ($p = 0.41,\ p = 0.43,\ p = 0.86$ for CDR H1, H2, and H3 RMSD). Nevertheless, this outcome provides further empirical justification for our results in Section 3.6.1 and acts as a robust proof of concept for integrating coordinate information into docking or *de novo* design tasks.

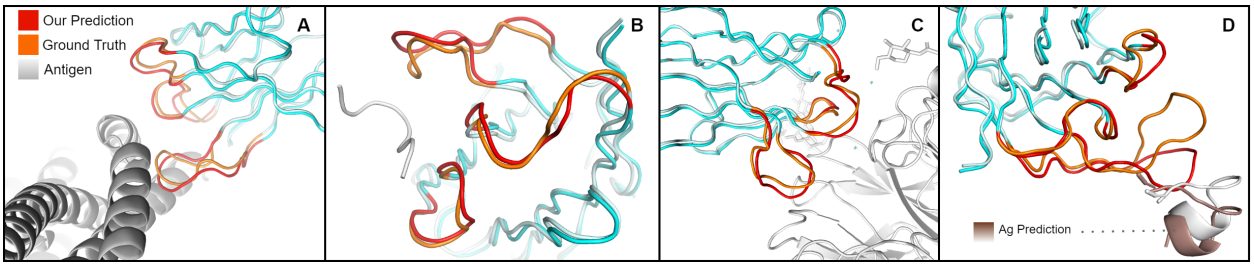

Figure 3.15: **Examples of antibody docking and CDR-loop design** Example docking and designs comparing our predictions with native structures. For each example, we give the length (L) of CDR H3 and the RMSD between the predicted (red) and ground truth (orange) conformations. For simplicity, only heavy chains are displayed. Only the bound antigen (gray-white) is shown when the prediction L-RMSD is less than 2 Å. (**A**) Fab of mAb 3E9 in complex with Plasmodium vivax reticulocyte-binding protein 2b (PvRBP2b) (PDB: 6BPA, $L = 11$, $RMSD = 1.49$). (**B**) Fab of IgG B13I2 bound to synthetic 19-amino acid peptide homolog of the C helix of myohemerythrin (PDB: 2IGF, $L = 11$, $RMSD = 1.19$). (**C**) Fab of mAb B10 heavy chain in complex with A(H3N2) influenza Virus (PDB: 6N6B, $L = 9$, $RMSD = 1.21$). (**D**) Fab of igG 7B2 bound to 13-residue HIV-1 GP41 peptide (PDB: 4YDV, $L = 17$, $RMSD = 2.86$)

3.7 Concluding Remarks

In this work, we developed DockGPT, a deep learning architecture for flexible protein docking with applications to *de novo* design of protein-binding proteins. Unlike other methods, our approach circumvents explicit training on bound structures and offers a natural approach to modeling conformational flexibility in complex prediction. By comparison across multiple benchmarks, we show that DockGPT meets or exceeds state-of-the-art methods on rigorous quality metrics while also making better use of binding site information when it's available. With significantly reduced inference times and explicit confidence estimates, we anticipate that our model will find further applications to machine-learning-based virtual screening and *de novo* design platforms.

Despite our success, there are several limitations and extensions of our approach left open for future investigation. We use only a single atom type and threshold to supply our model with interface and contact information. Although it is straightforward to incorporate more fine-grained binding site information, we did not explore this here. Parallel to this, supplying

noisy or probabilistic binding site information could potentially improve performance and generalization.

Our training procedure enables the generation of diverse conformations by enumerating random contacts. We show in Section 3.4.4 how this can be used to rank and generate diverse binding modes and ultimately improve blind docking. We suspect that this approach can be refined or extended to achieve even better performance. Specifically, (1) MSA information should significantly improve confidence predictions, (2) predicted distances should be used to generate a better initial population, and (3) Considering more candidates should help improve results. It is also important to maintain a *diverse* set population at any given time step. Conformations should be clustered by complex RMSD and only one representative from each cluster should be considered when sampling for the next population. We remark that we did not perform this clustering.

Although some of our deep network components were drawn from AlphaFold2, we do not incorporate any MSA information. We expect MSA embeddings to be especially helpful in the blind docking setting. Finally, for *de novo* design tasks, we only analyzed our model on CDR loop design and do not include estimates of binding affinity or free energy. Evaluation across more rigorous criteria and a broader range of design tasks must still be performed. We hope that future work will address some of these issues and develop this approach further.

APPENDIX A

SUPPLEMENT TO CHAPTER 2

A.1 Data Collection

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[Mir+22] 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[SS17], 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 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[1]. The first RosettaFold model for each target was selected for comparison.

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

Inverse Folding

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.[Ing+19]. 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.

A.2 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[KB15] 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[Bac+20], for every residual connection in our transformer blocks. We also apply gradient clipping by global norm[PMB13] 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 2.2.2. 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.

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 A.1. 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).

	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 A.1: **Hyperparameters used for fixed backbone design models**

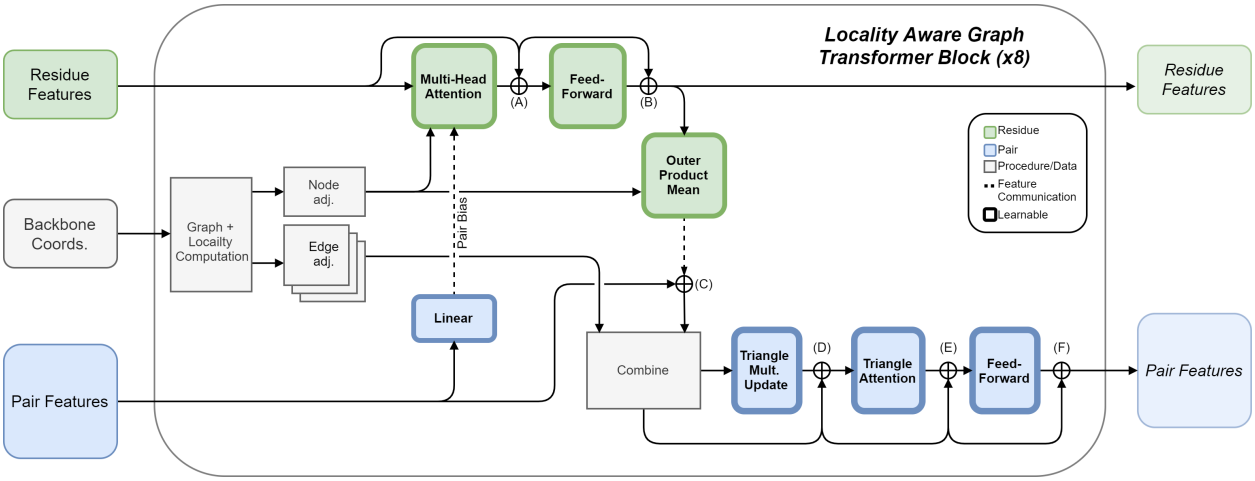

Figure A.1: **Architecture of Locality Aware Graph Transformer Block**. Standard multi-headed graph attention is used to update node features while edges between adjacent nodes are updated with triangle multiplication and attention. For each triangle update, edge adjacency information is used to select only those triangles having maximum side-length less than a fixed threshold.

A.3 Supplementary Figures

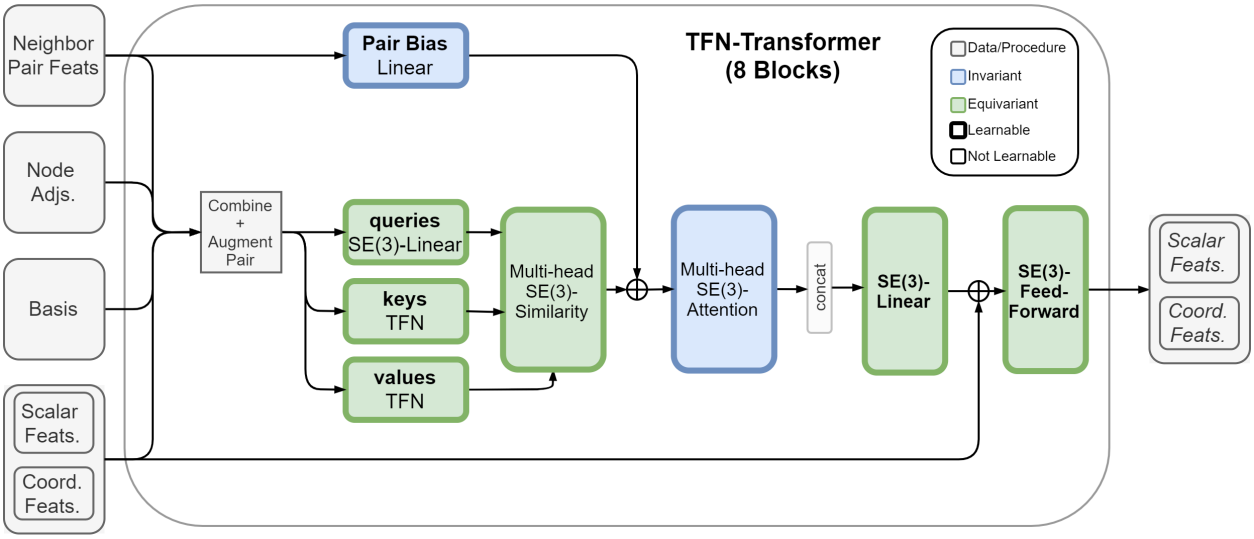

Figure A.2: **Architecture of TFN-based transformer block**. Keys and values are computed using TFNs. The input to each TFN radial kernel consists of pair features and distance information between points for the corresponding pair. Pair features are also linearly projected to bias the similarity logits for adjacent pairs of residues. The multi-headed SE(3)-similarity and SE(3)-Attention modules are detailed in Table 2.14. Learnable modules are displayed in bold font.

A.4 Supplementary Tables

	RMSD (Å)↓			$\chi-$MAE°↓				χ-Acc(%)↑		
Method	All	Sfc.	Core	χ_1	χ_2	χ_3	χ_4	All	Sfc.	Core
SCWRL	0.947	1.174	0.554	27.51	29.70	49.20	59.74	56.4%	47.2%	73.1%
FASPR	0.930	1.154	0.557	27.17	29.38	50.14	59.75	56.4%	47.9%	71.6%
RosettaPacker	0.886	1.126	0.480	25.97	28.87	47.36	54.82	58.7%	47.9%	77.3%
DL-Packer	0.783	0.989	0.446	22.14	27.00	50.38	70.18	58.9%	49.4%	75.5%
AttnPack.	0.672	0.856	**0.375**	18.32	24.24	46.28	57.97	**62.7%**	**53.9%**	**77.7%**
+Design	**0.669**	**0.843**	0.382	**17.86**	**24.13**	**45.75**	**57.57**	61.7%	52.7%	77.1%

Table A.2: **Average RMSD (Å) and MAE results for 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 A.5 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.

	CASP13					
	#Res	SCWRL	FASPR	Ros-P	DL-P	Ours
ARG	1107	2.216	2.192	2.063	1.916	**1.679**
ASN	1089	0.923	0.894	0.883	0.768	**0.663**
ASP	1212	0.888	0.906	0.872	0.725	**0.610**
CYS	238	0.533	0.549	0.447	0.446	**0.341**
GLN	844	1.489	1.478	1.331	1.241	**1.048**
GLU	1264	1.469	1.466	1.419	1.316	**1.137**
HIS	468	1.073	0.913	0.936	0.754	**0.681**
ILE	1315	0.577	0.577	0.545	0.519	**0.452**
LEU	2114	0.604	0.592	0.566	0.518	**0.432**
LYS	1161	1.751	1.732	1.647	1.524	**1.364**
MET	511	1.324	1.228	1.167	1.105	**0.956**
PHE	949	0.825	0.731	0.742	0.564	**0.450**
PRO	1017	0.248	0.236	0.228	0.188	**0.178**
SER	1558	0.720	0.721	0.698	0.580	**0.510**
THR	1401	0.514	0.506	0.490	0.448	**0.391**
TRP	385	1.332	1.128	1.001	0.837	**0.698**
TYR	897	1.034	0.965	0.980	0.678	**0.603**
VAL	1588	0.328	0.328	0.316	0.291	**0.251**
AVG.		0.934	0.910	0.872	0.772	**0.669**

Table A.3: Average Per-Residue RMSD for CASP13 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)

	CASP14					
	#Res	SCWRL	FASPR	Ros-P	DL-P	Ours
ARG	841	2.292	2.313	2.167	2.002	**1.825**
ASN	1338	1.062	1.031	0.976	0.923	**0.806**
ASP	1223	1.020	1.058	1.005	0.936	**0.841**
CYS	186	0.526	0.553	**0.304**	0.473	0.371
GLN	741	1.563	1.554	1.440	1.325	**1.199**
GLU	1277	1.611	1.596	1.554	1.497	**1.395**
HIS	378	1.118	1.057	0.947	0.893	**0.815**
ILE	1377	0.729	0.770	0.758	0.701	**0.616**
LEU	1823	0.753	0.712	0.719	0.650	**0.573**
LYS	1353	1.848	1.836	1.784	1.660	**1.462**
MET	401	1.527	1.341	1.357	1.308	**1.139**
PHE	933	0.905	0.885	0.811	0.731	**0.559**
PRO	792	0.263	0.252	0.257	**0.218**	0.220
SER	1420	0.866	0.844	0.815	0.736	**0.652**
THR	1214	0.707	0.685	0.687	0.643	**0.578**
TRP	229	1.466	1.319	1.416	1.083	**0.996**
TYR	882	1.063	1.046	0.972	0.800	**0.618**
VAL	1285	0.431	0.444	0.428	0.394	**0.352**
AVG.		1.062	1.048	1.006	0.929	**0.823**

Table A.4: Average Per-Residue RMSD for 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)

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 A.5: List of targets in each test data set.

APPENDIX B

SUPPLEMENT TO CHAPTER 3

B.1 Data Collection

For all methods, the receptor and ligand chains were randomly rotated and translated before inference. For general proteins, the smaller of the two targets was treated as the ligand (ties broken based on chain order in PDB file). For antibody-antigen chains, the antigen was always treated as the ligand.

Code for EquiDock was downloaded from the author's GitHub page. Standalone packages for HDock, PatchDock, and ZDock were downloaded from the respective servers. All binding interface and contact information were given as input for HDock and PatchDock. Still, the results required an additional post-processing step when run locally. For this, we enumerate all predictions of each program and choose the lowest energy prediction satisfying the interface and contact criteria. We reiterate that interface and contacts are defined using $C\alpha$ atoms with a 10Å cutoff. In some cases, HDock or PatchDock did not produce any decoys meeting all criteria. In these cases, we choose the lowest-scoring model with the most recovered interface residues and contacts.

AlphaFold and AlphaFold-Multimer were run with ColabFold [Mir+22] using the provided template and MSA servers. Default settings were used for all other options. ColabFold's monomer setting was used to predict all chains in the DB5 benchmark and all antigen chains in the RAbD and Ab-Bench benchmarks. The multimer setting was used to generate all predicted antibody structures with bound heavy and light chains.

As mentioned in Section 3.3.5, we filter unbound and predicted targets based on RMSD to the bound conformation. Full lists of targets used for comparisons are included with the code at https://github.com/MattMcPartlon/protein-docking

B.2 Extended Results and Examples

B.2.1 Docking Benchmark Version 5

	Docking Benchmark Version 5.5, AF2-Predicted (Top-1)						
	DockQ↑	I-RMSD↓			L-RMSD↓		
	SR (%)	25^{th}	50^{th}	75^{th}	25^{th}	50^{th}	75^{th}
Blind							
EquiDock	0.0%	9.8	**12.2**	**15.4**	26.4	42.1	50.1
ZDock	**9.1%**	**8.5**	12.9	17.6	**19.5**	**29.5**	**34.8**
PatchDock	4.5%	11.5	14.4	19.4	29.8	38.2	55.5
HDock	**9.1%**	12.0	15.2	20.6	27.5	36.9	63.6
Ours	**9.1%**	9.6	13.8	18.9	22.8	34.3	58.3
4 Interface							
ZDock	9.1%	7.2	11.9	15.3	17.6	27.8	33.1
PatchDock	4.5%	9.7	11.7	14.4	21.1	29.5	38.6
HDock	9.1%	8.7	11.3	14.4	21.1	30.1	41.4
Ours	**59.1%**	**2.2**	**3.2**	**7.4**	**6.1**	**7.1**	**22.2**
1 Contact							
ZDock	13.6%	6.5	12.0	17.6	17.0	28.4	37.8
PatchDock	4.5%	10.0	14.4	19.1	21.1	34.6	54.9
HDock	9.1%	10.9	15.0	20.6	27.5	39.0	63.6
Ours	**27.3%**	**3.6**	**7.1**	**10.3**	**10.0**	**18.3**	**29.1**
2 Contacts							
ZDock	9.1%	7.2	11.5	15.6	17.0	26.9	34.8
PatchDock	4.5%	9.1	12.8	15.4	22.1	31.1	50.1
HDock	13.6%	7.9	14.6	20.6	23.1	39.0	63.6
Ours	**66.7%**	**2.3**	**3.3**	**6.4**	**6.4**	**7.3**	**17.7**
3 Contacts							
ZDock	13.6%	6.5	11.0	15.6	15.2	28.4	34.8
PatchDock	4.5%	9.8	13.7	15.6	27.1	31.4	52.9
HDock	18.2%	7.8	15.0	20.6	18.7	38.4	63.6
Ours	**75.0%**	**1.5**	**2.4**	**4.1**	**4.6**	**7.8**	**11.7**

Table B.1: **Top-1 docking statistics for docking DB5 targets, given chain conformations predicted by AlphaFold2.**

	Docking Benchmark Version 5.5, AF2-Predicted (Top-5)						
	DockQ↑	I-RMSD↓			L-RMSD↓		
	SR (%)	25^{th}	50^{th}	75^{th}	25^{th}	50^{th}	75^{th}
Blind							
ZDock	**18.2%**	**5.5**	**10.0**	13.0	**12.0**	**22.4**	**31.1**
PatchDock	9.1%	8.4	10.2	11.6	18.6	27.4	32.0
HDock	**18.2%**	8.6	11.0	13.3	17.0	28.6	34.1
4 Interface							
ZDock	31.8%	2.9	6.8	10.2	8.1	15.2	26.9
PatchDock	18.2%	5.1	8.3	10.0	15.1	18.1	27.8
HDock	27.3%	3.6	8.3	10.1	14.6	19.6	25.9
Ours	**68.2%**	**2.2**	**2.6**	**4.1**	**5.4**	**6.8**	**10.6**
1 Contact							
ZDock	36.4%	2.7	6.5	13.0	8.0	18.4	31.1
PatchDock	27.3%	4.0	8.4	11.2	13.1	19.0	32.0
HDock	22.7%	6.0	10.2	12.8	16.8	23.8	33.5
Ours	**54.5%**	**2.4**	**2.8**	**7.2**	**6.7**	**10.5**	**15.7**
2 Contacts							
ZDock	36.4%	2.4	6.5	13.0	5.8	17.3	31.1
PatchDock	27.3%	4.0	7.9	10.6	13.1	18.6	32.0
HDock	22.7%	6.0	8.8	11.1	16.8	24.0	29.5
Ours	**90.5%**	**1.9**	**2.4**	**3.0**	**5.1**	**6.5**	**7.3**
3 Contacts							
ZDock	40.9%	2.4	6.9	13.2	5.8	15.2	31.4
PatchDock	27.3%	3.7	7.3	9.9	11.1	16.7	32.0
HDock	27.3%	3.8	8.4	10.1	14.6	22.7	28.9
Ours	**95.0%**	**1.4**	**2.0**	**2.4**	**4.0**	**5.3**	**8.0**

Table B.2: **Top-5 docking statistics for docking DB5 targets, given chain conformations predicted by AlphaFold2.**

	Docking Benchmark Version 5.5, Unbound (Top-1)						
	DockQ↑	I-RMSD↓			L-RMSD↓		
	SR (%)	25^{th}	50^{th}	75^{th}	25^{th}	50^{th}	75^{th}
Blind							
EquiDock	0.0%	11.4	14.1	17.1	35.0	40.8	50.6
ZDock	11.9%	11.8	14.1	17.3	25.0	34.4	42.8
PatchDock	0.0%	11.8	15.6	19.6	38.8	48.0	56.7
HDock	9.5%	11.3	15.9	18.0	29.7	41.8	53.5
AFM	**50.0%**	**0.9**	7.9	16.4	**2.8**	19.6	35.2
Ours	7.1%	8.9	13.3	17.4	24.2	35.4	49.5
Ours+GA	9.5%	9.7	14.0	17.5	23.1	33.4	47.5
Ours+AFM	42.8%	2.7	**5.7**	**14.3**	6.6	**17.4**	**28.7**
4 Interface							
ZDock	14.3%	8.7	11.7	13.8	18.4	26.9	34.9
PatchDock	2.4%	9.2	11.6	15.8	26.2	37.4	52.5
HDock	11.9%	8.0	10.7	14.3	19.3	29.6	39.7
Ours	**47.6%**	**2.7**	**4.7**	**8.9**	**7.1**	**14.5**	**23.6**
1 Contact							
ZDock	16.7%	7.7	11.2	14.4	19.3	31.4	38.8
PatchDock	2.4%	10.9	14.2	18.8	34.9	45.5	54.5
HDock	14.3%	10.3	14.8	17.7	26.9	38.0	52.2
Ours	**45.2%**	**2.5**	**5.8**	**9.9**	**8.6**	**13.6**	**26.8**
2 Contacts							
ZDock	19.0%	6.7	11.9	14.4	17.9	30.5	41.9
PatchDock	4.8%	9.4	14.1	17.5	27.0	40.2	53.2
HDock	14.3%	7.9	11.3	17.5	20.4	28.2	48.1
Ours	**66.7%**	**1.7**	**2.7**	**4.6**	**4.1**	**7.3**	**13.9**
3 Contacts							
ZDock	23.8%	4.5	10.3	14.2	14.4	24.6	37.6
PatchDock	7.1%	9.1	13.3	17.5	27.3	39.2	53.2
HDock	19.0%	6.5	10.5	17.0	15.5	27.3	45.3
Ours	**88.0%**	**1.6**	**2.3**	**3.5**	**5.0**	**5.8**	**9.1**

Table B.3: **Top-1 docking statistics for DB5 targets, given unbound chain conformations as input.**

	Docking Benchmark Version 5.5, Unbound (Top-5)						
	DockQ↑	**I-RMSD↓**			**L-RMSD↓**		
	SR (%)	**25th**	**50th**	**75th**	**25th**	**50th**	**75th**
Blind							
ZDock	14.3%	7.3	10.1	12.7	17.2	22.7	32.6
PatchDock	0.0%	9.0	11.5	15.0	26.8	37.9	48.9
HDock	19.0%	6.8	10.2	11.6	15.7	26.2	32.1
AFM	50%	**0.9**	**4.7**	13.0	**2.6**	**12.2**	30.2
Ours+GA	16.7%	5.4	8.8	13.6	12.9	20.7	34.3
Ours+AFM	**52.4%**	2.0	5.1	**12.7**	5.2	12.4	**24.5**
4 Interface							
ZDock	33.3%	3.4	6.2	9.0	10.8	17.1	20.6
PatchDock	4.8%	8.3	9.5	12.1	17.3	26.4	40.6
HDock	31.0%	3.5	7.4	9.8	12.0	19.2	24.5
Ours	**69.0%**	**2.1**	**3.0**	**5.2**	**5.9**	**8.6**	**12.5**
1 Contact							
ZDock	31.0%	3.3	7.3	11.2	10.7	18.5	29.1
PatchDock	7.1%	7.8	10.7	14.5	18.5	36.1	46.0
HDock	33.3%	3.4	7.1	10.5	12.0	19.2	29.1
Ours	**59.5%**	**2.0**	**3.5**	**6.2**	**5.5**	**10.2**	**16.3**
2 Contacts							
ZDock	40.5%	2.9	5.9	10.1	9.8	17.1	25.9
PatchDock	14.3%	6.6	9.1	12.8	15.0	27.7	43.3
HDock	38.1%	3.4	6.5	10.0	11.3	17.1	29.3
Ours	**92.9%**	**1.6**	**2.1**	**2.6**	**3.9**	**5.3**	**8.1**
3 Contacts							
ZDock	45.2%	2.8	4.9	8.5	9.0	16.5	22.2
PatchDock	16.7%	5.5	9.1	11.8	15.0	24.9	42.9
HDock	38.1%	3.3	6.6	9.6	11.3	16.9	24.5
Ours	**100%**	**1.4**	**1.8**	**2.6**	**4.3**	**5.1**	**6.4**

Table B.4: **Top-5 docking statistics for DB5 targets, given unbound chain conformations as input.**

Antibody Benchmark

	Antibody Benchmark Predicted Targets (Top-1)						
	DockQ↑	**I-RMSD↓**			**L-RMSD↓**		
	SR (%)	**25th**	**50th**	**75th**	**25th**	**50th**	**75th**
Blind							
EquiDock	0.0%	13.2	14.5	16.4	38.3	41.6	50.0
ZDock	3.8%	10.6	12.9	15.4	21.9	26.8	37.3
PatchDock	0.0%	12.3	14.0	18.8	26.3	33.3	49.5
HDock	0.0%	11.9	13.5	18.9	25.5	31.2	52.4
Ours	26.9%	2.8	10.4	13.9	9.4	22.9	26.4
Ours+GA	**42.3%**	**1.9**	**7.3**	**9.0**	**6.5**	**15.5**	**19.4**
4 Epitope							
ZDock	7.7%	7.4	9.4	13.2	18.6	24.7	31.8
PatchDock	3.8%	10.3	12.2	14.5	24.8	27.2	40.0
HDock	3.8%	9.0	13.2	14.1	23.0	27.6	35.2
Ours	**53.8%**	**1.9**	**2.7**	**9.3**	**4.8**	**8.5**	**28.0**
12 Epitope							
ZDock	19.2%	5.7	10.3	11.6	14.0	23.2	28.6
PatchDock	11.5%	7.4	10.2	12.4	17.3	26.1	38.9
HDock	7.7%	7.8	10.2	13.5	17.9	24.3	30.2
Ours	**69.2%**	**1.3**	**1.8**	**5.8**	**3.7**	**6.1**	**21.1**

Table B.5: **Top-1 docking statistics for AlphaFold-Multimer predicted antibody and AlphaFold2 predicted antigen.**

	Antibody Benchmark Predicted Targets (Top-5)						
	DockQ↑	**I-RMSD↓**			**L-RMSD↓**		
	SR (%)	**25th**	**50th**	**75th**	**25th**	**50th**	**75th**
Blind							
ZDock	7.7%	6.4	9.6	11.5	14.7	19.8	28.7
PatchDock	3.8%	7.8	10.9	12.2	18.4	22.7	28.0
HDock	3.8%	8.1	10.1	12.4	18.3	25.2	30.7
Ours+GA	**46.2%**	**1.8**	**7.2**	**9.1**	**6.5**	**13.5**	**17.3**
4 Epitope							
ZDock	30.8%	3.9	6.1	8.8	8.9	15.3	21.0
PatchDock	7.7%	6.0	7.7	9.5	15.2	19.4	26.4
HDock	19.2%	5.0	7.5	10.4	15.8	19.5	25.9
Ours	**69.2%**	**1.5**	**2.4**	**3.4**	**4.4**	**6.0**	**11.8**
12 Epitope							
ZDock	57.7%	2.0	3.9	6.2	4.8	11.8	15.8
PatchDock	30.8%	4.5	6.0	7.4	8.9	17.2	24.3
HDock	50.0%	2.1	5.3	8.6	5.8	13.2	22.7
Ours	**88.5%**	**1.3**	**1.7**	**3.2**	**3.7**	**4.8**	**9.8**

Table B.6: **Top-5 docking statistics for AlphaFold-Multimer predicted antibody and AlphaFold2 predicted antigen.**

	Antibody Benchmark Unbound Targets (Top-1)						
	DockQ↑	**I-RMSD↓**			**L-RMSD↓**		
	SR (%)	**25th**	**50th**	**75th**	**25th**	**50th**	**75th**
Blind							
EquiDock	0.0%	11.6	13.7	16.8	31.9	41.1	51.0
ZDock	2.2%	10.1	12.8	17.0	23.9	28.2	39.3
PatchDock	0.0%	12	13.9	15.5	26.1	32.2	46.3
HDock	2.2%	12.5	15.6	19.8	24.0	47.3	58.5
AF-Mult.	28.3%	1.9	9.3	14.7	12.2	22.6	36.0
Ours	26.1%	2.5	9.2	**12.1**	8.2	19.5	25.4
Ours+GA	**37.0%**	**1.8**	**8.3**	12.4	**5.5**	**19.2**	26.4
Ours+AFM	28.3%	1.9	10.1	13.3	5.7	20.1	27.8
4 Epitope							
ZDock	8.7%	8.2	10.4	13.3	20.7	27.2	33.0
PatchDock	0.0%	9.7	11.9	14.7	22.4	28.8	39.7
HDock	8.7%	9.9	12.1	15.7	21.3	27.6	42.6
Ours	**54.3%**	**1.6**	**3.1**	**6.8**	**4.6**	**9.5**	**20.6**
12 Epitope							
ZDock	26.1%	3.6	8.5	11.4	14.4	20.7	27.8
PatchDock	0.0%	8.4	10.2	12.9	21.3	25.1	32.7
HDock	13.0%	7.6	10.3	13.1	18.6	23.9	36.1
Ours	**65.2%**	**1.2**	**1.9**	**6.8**	**3.8**	**7.5**	**22.3**

Table B.7: **Top-1 statistics for docking Antibody Benchmark targets given unbound chain conformations as input.**

	Antibody Benchmark Unbound Targets (Top-5)						
	DockQ↑	**I-RMSD↓**			**L-RMSD↓**		
	SR (%)	**25th**	**50th**	**75th**	**25th**	**50th**	**75th**
Blind							
ZDock	17.4%	5.8	8.1	11.5	14.7	21.2	28.6
PatchDock	2.2%	6.6	10.2	12.9	19.7	23.9	37.0
HDock	8.7%	8.6	10.8	13.7	21.1	24.4	39.3
AF-Mult.	34.8%	1.8	5.8	13.1	9.2	18.3	26.4
Ours+GA	**45.7%**	**1.7**	**4.0**	**7.3**	**4.9**	**11.5**	**19.8**
Ours+AFM	37.0%	**1.7**	4.4	8.9	5.4	11.3	19.0
4 Epitope							
ZDock	28.3%	2.8	6.2	7.9	10.2	15	21.2
PatchDock	8.7%	6.0	8.2	9.2	16.8	20.1	28.5
HDock	30.4%	3.4	7.5	9.9	11.7	18.9	23.6
Ours	**71.7%**	**1.4**	**2.5**	**3.7**	**4.4**	**6.7**	**12.3**
12 Epitope							
ZDock	56.5%	1.4	3.2	7.2	5.3	9.7	20.7
PatchDock	21.7%	4.2	6.4	7.9	15.2	17.8	27.2
HDock	47.8%	2.2	4.2	7.8	7.6	14.2	22.2
Ours	**87.0%**	**1.2**	**1.5**	**2.6**	**3.5**	**5.4**	**8.8**

Table B.8: **Top-5 statistics for docking Antibody Benchmark targets given unbound chain conformations as input.**

Examples

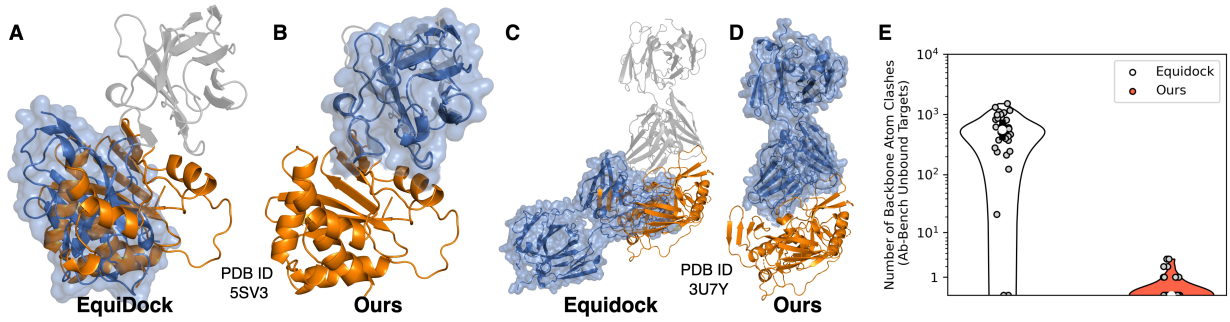

Figure B.1: **Comparison between DockGPT and Equidock** Blind docking predictions for a single domain antibody targeting the toxin Ricin (**A** and **B**), and therapeutic antibody which targets the CD4 binding site on the HIV-1 spike protein (**C** and **D**). In (A–D), we show the bound antigen in orange and the bound antibody in light gray. For clarity, we align each complex prediction to the ground truth using only the antigen chain and show only the predicted antibody in blue. We also show the solvent-accessible surface of antibody predictions (independent of the antigen) to better illustrate surface intersections. The RMSD between bound and unbound antigen chains is less than 2Å for both targets. (**E**) Distribution of the number of steric clashes for blind docking DB5 unbound targets. We consider only backbone atom clashes since EquiDock cannot modify side-chain conformations. Two atoms are said to clash if each atom belongs to a different chain and the pairwise distance is less than 90% the sum of their van der Waals radii.

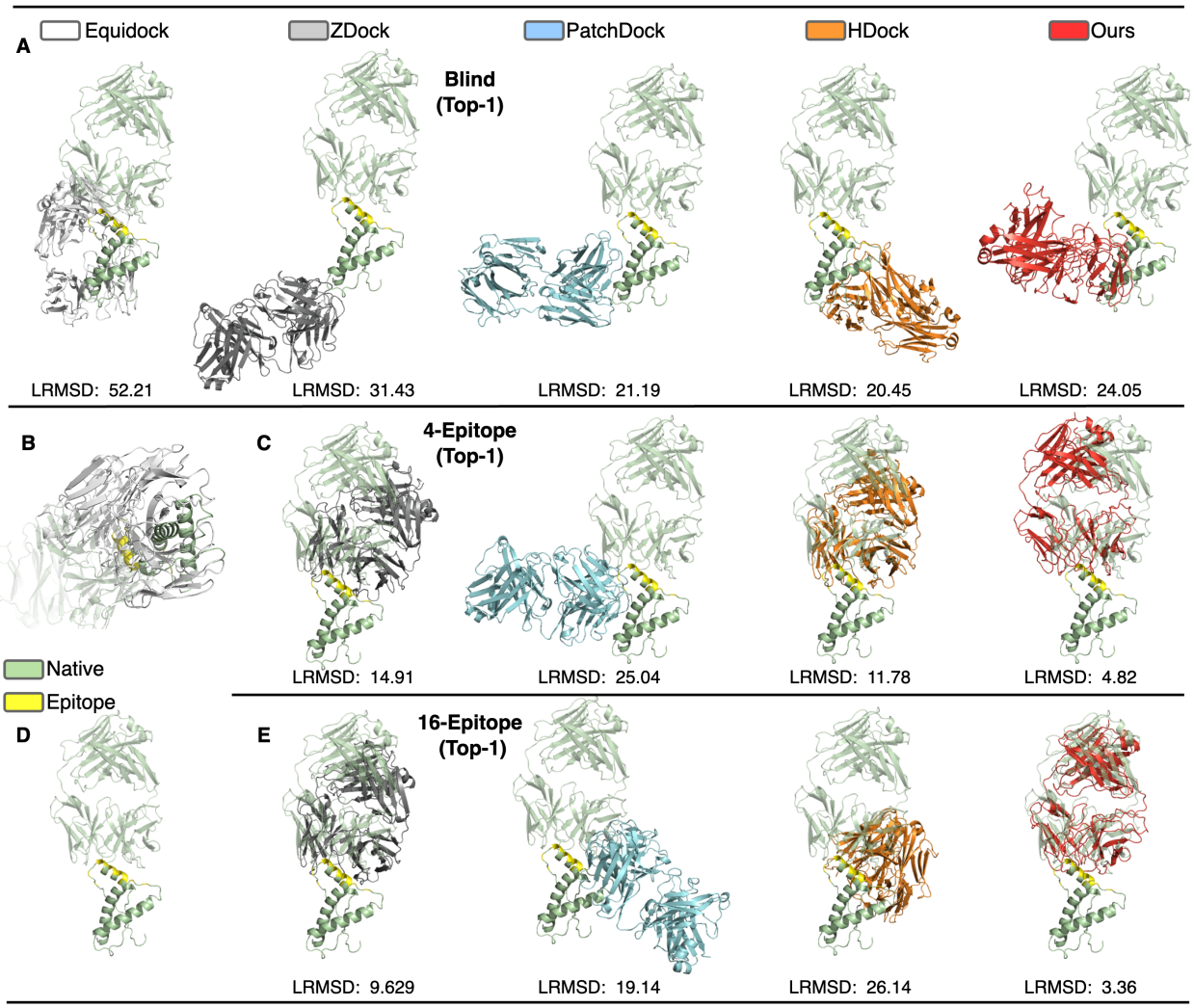

Figure B.2: **Docking Predictions for Antibody Benchmark target 2W9E** Protein backbones are shown in cartoon format with ground-truth antibody and antigen structures shown in green for each figure. The antigen epitope is highlighted in yellow. We show each method's predicted antibody orientation relative to the ground truth antigen in a different color. Ligand RMSD (LRMSD) is shown for each prediction. (**A**) Blind docking predictions for methods EquiDock, ZDock, PatchDock, HDock, and DockGPT. (**B**) Close-up of EquiDock's prediction showing the excessive surface overlap between antibody and antigen chain predictions. (**C**) Top-1 docking predictions for each method, except EquiDock given four epitope residues. (**D**) Ground truth complex. (**E**) Top-1 docking predictions for each method, except EquiDock given 12 epitope residues.

BIBLIOGRAPHY

[AA22] Namrata Anand and Tudor Achim. *Protein Structure and Sequence Generation with Equivariant Denoising Diffusion Probabilistic Models*. 2022.

[Ado+18] Jared Adolf-Bryfogle et al. "RosettaAntibodyDesign (RAbD): A general framework for computational antibody design". en. In: *PLoS Comput. Biol.* 14.4 (Apr. 2018), e1006112.

[Agr+19] Piyush Agrawal et al. "Benchmarking of different molecular docking methods for protein-peptide docking". In: *BMC Bioinformatics* 19.13 (Feb. 2019), p. 426.

[Ahd+22] Gustaf Ahdritz et al. "OpenFold: Retraining AlphaFold2 yields new insights into its learning mechanisms and capacity for generalization". In: *bioRxiv* (2022). eprint: `https://www.biorxiv.org/content/early/2022/11/22/2022.11.20.517210.full.pdf`.

[Akb+22] Rahmad Akbar et al. "In silico proof of principle of machine learning-based antibody design at unconstrained scale". en. In: *MAbs* 14.1 (Jan. 2022).

[ALC97] B Al-Lazikani, A M Lesk, and C Chothia. "Standard conformations for the canonical structures of immunoglobulins". en. In: *J. Mol. Biol.* 273.4 (Nov. 1997), pp. 927–948.

[Alf+17] Rebecca F. Alford et al. "The Rosetta All-Atom Energy Function for Macromolecular Modeling and Design." In: *Journal of chemical theory and computation* 13 6 (2017), pp. 3031–3048.

[All+19] Ethan C Alley et al. "Unified rational protein engineering with sequence-based deep representation learning". In: *Nature methods* 16.12 (2019), pp. 1315–1322.

[AM15] Hossam M Ashtawy and Nihar R Mahapatra. "Machine-learning scoring functions for identifying native poses of ligands docked to known and novel proteins". en. In: *BMC Bioinformatics* 16 Suppl 6.S6 (Apr. 2015), S3.

[Ana+22] N. Anand et al. "Protein sequence design with a learned potential". In: *Nat Commun* 13.1 (Feb. 2022), p. 746.

[Ast87] Rick Astley. *Never Gonna Give You Up*. 1987.

[Bac+20] Thomas Bachlechner et al. *ReZero is All You Need: Fast Convergence at Large Depth*. 2020. arXiv: `2003.04887 [cs.LG]`.

[Bae+21] Minkyung Baek et al. "Accurate prediction of protein structures and interactions using a three-track neural network". In: *Science* 373.6557 (2021), pp. 871–876. eprint: `https://www.science.org/doi/pdf/10.1126/science.abj8754`.

[Bar+17] Damian Bartuzi et al. "Recent advances and applications of molecular docking to G protein-coupled receptors". en. In: *Molecules* 22.2 (Feb. 2017), p. 340.

[Ben+21] B.J. Bender et al. "A practical guide to large-scale docking". In: *Nature Protocols* 16 (2021), pp. 4799–4832.

[Ben+22] Nathaniel Bennett et al. "Improving de novo Protein Binder Design with Deep Learning". In: *bioRxiv* (2022).

[Ber+00] H M Berman et al. “The Protein Data Bank”. en. In: *Nucleic Acids Res.* 28.1 (Jan. 2000), pp. 235–242.

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. 2016.

[BKK20] Aleksandra Badaczewska-Dawid, Andrzej Kolinski, and Sebastian Kmiecik. “Computational reconstruction of atomistic protein structures from coarse-grained models”. In: *Computational and structural biotechnology journal* 18 (2020), pp. 162–176.

[BMP13] Laura Bettinetti, Matteo Magnani, and Alessandro Padova. “Drug Discovery by Targeting Protein–Protein Interactions”. In: *Disruption of Protein-Protein Interfaces: In Search of New Inhibitors*. Ed. by Stefano Mangani. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–29.

[Bro+20] Tom B. Brown et al. “Language Models are Few-Shot Learners”. In: *CoRR* abs/2005.14165 (2020). arXiv: 2005.14165.

[BSF94] Y Bengio, P Simard, and P Frasconi. “Learning long-term dependencies with gradient descent is difficult”. en. In: *IEEE Trans. Neural Netw.* 5.2 (1994), pp. 157–166.

[BW16] Sankar Basu and Björn Wallner. “DockQ: A quality measure for protein-protein docking models”. en. In: *PLoS One* 11.8 (Aug. 2016), e0161879.

[Byr+95] Richard H. Byrd et al. “A Limited Memory Algorithm for Bound Constrained Optimization”. In: *SIAM Journal on Scientific Computing* 16.5 (1995), pp. 1190–1208.

[Cao+10] Yang Cao et al. “Improved side-chain modeling by coupling clash-detection guided iterative search with rotamer relaxation”. In: *Bioinformatics* 27.6 (Jan. 2010), pp. 785–790. eprint: https://academic.oup.com/bioinformatics/article-pdf/27/6/785/16902094/btr009.pdf.

[Cao+21] Longxing Cao et al. “Robust de novo design of protein binding proteins from target structural information alone”. In: *bioRxiv* (2021).

[Car+20] Nicolas Carion et al. “End-to-End Object Detection with Transformers”. In: *CoRR* abs/2005.12872 (2020). arXiv: 2005.12872.

[Cha+21] Chung-ke Chang et al. “Targeting protein-protein interaction interfaces in COVID-19 drug discovery”. In: *Computational and Structural Biotechnology Journal* 19 (2021), pp. 2246–2255.

[Che+16] Tianqi Chen et al. “Training Deep Nets with Sublinear Memory Cost”. In: *CoRR* abs/1604.06174 (2016). arXiv: 1604.06174.

[Che+20] Sheng Chen et al. “To Improve Protein Sequence Profile Prediction through Image Captioning on Pairwise Residue Distance Map”. In: *Journal of Chemical Information and Modeling* 60.1 (2020). Pmid: 31800243, pp. 391–399.

[Chi+95] G. Chinea et al. “The use of position-specific rotamers in model building by homology”. In: *Proteins* 23.3 (1995), pp. 415–421.

[Cho+89] Cyrus Chothia et al. “Conformations of immunoglobulin hypervariable regions”. In: *Nature* 342.6252 (Dec. 1989), pp. 877–883.

[CL87] Cyrus Chothia and Arthur M. Lesk. “Canonical structures for the hypervariable regions of immunoglobulins”. In: *Journal of Molecular Biology* 196.4 (1987), pp. 901–917.

[CLG10] Sidhartha Chaudhury, Sergey Lyskov, and Jeffrey J. Gray. “PyRosetta: a script-based interface for implementing molecular modeling algorithms using Rosetta”. In: *Bioinformatics* 26 5 (2010), pp. 689–91.

[CW20] Chen Cai and Yusu Wang. “A Note on Over-Smoothing for Graph Neural Networks”. In: *CoRR* abs/2006.13318 (2020). arXiv: 2006.13318.

[Dau+22] J. Dauparas et al. “Robust deep learning based protein sequence design using ProteinMPNN”. In: *Science* 378.6615 (2022), pp. 49–56.

[Den+21] Congyue Deng et al. “Vector Neurons: A General Framework for SO(3)-Equivariant Networks”. In: *ArXiv* abs/2104.12229 (2021).

[Dev+18] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805.

[DH99] John R. Desjarlais and Tracy M. Handel. “Side-chain and backbone flexibility in protein core design.” In: *Journal of molecular biology* 290 1 (1999), pp. 305–18.

[Dos+20] Alexey Dosovitskiy et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: *CoRR* abs/2010.11929 (2020). arXiv: 2010.11929.

[DSK09] R Dunbrack, M Shapovalov, and G Krivov. “Improved prediction of protein side-chain conformations with SCWRL4”. In: *Proteins* 77.4 (Dec. 2009), pp. 778–795.

[Dun02] Roland L Dunbrack. “Rotamer Libraries in the 21st Century”. In: *Current Opinion in Structural Biology* 12.4 (2002), pp. 431–440.

[DW08] Pawel Durek and Dirk Walther. “The integrated analysis of metabolic and protein interaction networks reveals novel molecular organizing principles”. en. In: *BMC Syst. Biol.* 2.1 (Nov. 2008), p. 100.

[DYZ21] Lanying Du, Yang Yang, and Xiujuan Zhang. “Neutralizing antibodies for the prevention and treatment of COVID-19”. In: *Cellular & Molecular Immunology* 18.10 (2021), pp. 2293–2306.

[Eva+22] Richard Evans et al. “Protein complex prediction with AlphaFold-Multimer”. In: *bioRxiv* (2022).

[Eya+03] Eran Eyal et al. “Protein side-chain rearrangement in regions of point mutations”. en. In: *Proteins* 50.2 (Feb. 2003), pp. 272–282.

[FAK07] Xiaoran Fu, James R Apgar, and Amy E Keating. “Modeling backbone flexibility to achieve sequence diversity: the design of novel alpha-helical ligands for Bcl-xL”. en. In: *J. Mol. Biol.* 371.4 (Aug. 2007), pp. 1099–1117.

[Far+17] S. Farokhirad et al. “3.13 Computational Methods Related to Molecular Structure and Reaction Chemistry of Biomaterials”. In: *Comprehensive Biomaterials II*. Ed. by Paul Ducheyne. Oxford: Elsevier, 2017, pp. 245–267.

[FBB07] G Faure, A Bornot, and AG de Brevern. “Protein contacts, inter-residue interactions and side-chain modelling.” In: *Erratum in: Biochimie* 90 4 (2007), pp. 626–39.

[Fuc+20] Fabian Fuchs et al. “SE(3)-Transformers: 3D Roto-Translation Equivariant Attention Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 1970–1981.

[FW11] Amr Fahmy and Gerhard Wagner. “Optimization of van der Waals Energy for Protein Side-Chain Placement and Design”. In: *Biophysical Journal* 101.7 (2011), pp. 1690–1698.

[Gai+13] Pablo Gainza et al. “OSPREY: protein design with ensembles, flexibility, and provable algorithms”. en. In: *Methods Enzymol.* 523 (2013), pp. 87–107.

[Gan+22] Octavian-Eugen Ganea et al. “Independent SE(3)-Equivariant Models for End-to-End Rigid Protein Docking”. In: *International Conference on Learning Representations*. 2022.

[Gil+17] Justin Gilmer et al. “Neural Message Passing for Quantum Chemistry”. In: *CoRR* abs/1704.01212 (2017). arXiv: 1704.01212.

[GPS16] Thomas Gaillard, Nicolas Panel, and Thomas Simonson. “Protein side chain conformation predictions with an MMGBSA energy function”. en. In: *Proteins* 84.6 (June 2016), pp. 803–819.

[Gue+21a] Isabella Guedes et al. “New machine learning and physics-based scoring functions for drug discovery”. In: *Nature Scientific Reports* 11 (2021).

[Gue+21b] Johnathan D Guest et al. “An expanded benchmark for antibody-antigen docking and affinity prediction reveals insights into antibody recognition determinants”. en. In: *Structure* 29.6 (June 2021), pp. 606–621.

[He+21] Kaiming He et al. “Masked Autoencoders Are Scalable Vision Learners”. In: *CoRR* abs/2111.06377 (2021). arXiv: 2111.06377.

[HG16] Dan Hendrycks and Kevin Gimpel. “Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units”. In: *CoRR* abs/1606.08415 (2016).

[Hir+20] Naozumi Hiranuma et al. “Improved protein structure refinement guided by deep learning based accuracy estimation”. In: *bioRxiv* (2020). eprint: https://www.biorxiv.org/content/early/2020/11/04/2020.07.17.209643.full.pdf.

[Hog+18] Herve Hogues et al. “ProPOSE: Direct Exhaustive Protein-Protein Docking with Side Chain Flexibility”. In: *Journal of Chemical Theory and Computation* 14.9 (2018). Pmid: 30107730, pp. 4938–4947. eprint: https://doi.org/10.1021/acs.jctc.8b00225.

[HPZ20] Xiaoqiang Huang, Robin Pearce, and Yang Zhang. "FASPR: an open-source tool for fast and accurate protein side-chain packing". In: *Bioinformatics* 36.12 (Apr. 2020), pp. 3758–3765. eprint: `https://academic.oup.com/bioinformatics/article-pdf/36/12/3758/33437211/btaa234.pdf`.

[HSS16] Maria Hauser, Martin Steinegger, and Johannes Söding. "MMseqs software suite for fast and deep clustering and searching of large protein sequence sets". en. In: *Bioinformatics* 32.9 (May 2016), pp. 1323–1330.

[Hsu+22] Chloe Hsu et al. "Learning inverse folding from millions of predicted structures". In: *bioRxiv* (2022).

[Hub01] A Huber. "Scaffolding proteins organize multimolecular protein complexes for sensory signal transduction". en. In: *Eur. J. Neurosci.* 14.5 (Sept. 2001), pp. 769–776.

[Ing+19] John Ingraham et al. "Generative Models for Graph-Based Protein Design". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019.

[JBJ20] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. *Multi-Objective Molecule Generation using Interpretable Substructures*. 2020.

[JFC22] Allan Jabri, David Fleet, and Ting Chen. *Scalable Adaptive Computation for Iterative Generation*. 2022. arXiv: `2212.11972 [cs.LG]`.

[Jia+22] Yining Jiang et al. "Membrane-mediated protein interactions drive membrane protein organization". In: *Nature Communications* 13.1 (2022), p. 7373.

[Jin+20] Bowen Jing et al. *Learning from Protein Structure with Geometric Vector Perceptrons*. 2020.

[Jin+21] Bowen Jing et al. *Equivariant Graph Neural Networks for 3D Macromolecular Structure*. 2021.

[Jin+22] Wengong Jin et al. "Iterative Refinement Graph Neural Network for Antibody Sequence-Structure Co-design". In: *International Conference on Learning Representations*. 2022.

[JKS21] Michael Jendrusch, Jan O. Korbel, and S. Kashif Sadiq. "AlphaDesign: A de novo protein design framework based on AlphaFold". In: *bioRxiv* (2021).

[Joh+21] Sean R. Johnson et al. "Generating novel protein sequences using Gibbs sampling of masked language models". In: *bioRxiv* (2021).

[Jum+18] John M. Jumper et al. "Accurate calculation of side chain packing and free energy with applications to protein molecular dynamics". In: *PLOS Computational Biology* 14.12 (Dec. 2018), pp. 1–25.

[Jum+21] John Jumper et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596.7873 (Aug. 2021), pp. 583–589.

[JX21] X Jing and J Xu. "Fast and effective protein model refinement using deep graph neural networks". In: *Nat. Comput Sci* 1 (2021), pp. 462–469.

[Kab76] Wolfgang Kabsch. "A solution for the best rotation to relate two sets of vectors". In: *Acta Crystallographica Section A* 32 (1976), pp. 922–923.

[Kac+18] Agnieszka A Kaczor et al. "Protein-protein docking in drug design and discovery". en. In: *Methods Mol. Biol.* 1762 (2018), pp. 285–305.

[KB15] Diederik P. Kingma and Jimmy Ba. "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*. Ed. by Yoshua Bengio and Yann LeCun. 2015.

[Koz+17] D Kozakov et al. "The ClusPro web server for protein-protein docking". In: *Nature Protocols* 12.2 (Feb. 2017), pp. 255–278.

[Kuh19] Brian Kuhlman. "Designing protein structures and complexes with the molecular modeling program Rosetta". In: *Journal of Biological Chemistry* 294.50 (2019), pp. 19436–19443.

[KW16] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *CoRR* abs/1609.02907 (2016). arXiv: 1609.02907.

[Lem+20] J. K. Leman et al. "Macromolecular modeling and design in Rosetta: recent methods and frameworks". en. In: *Nat Methods* 17.7 (July 2020), pp. 665–680.

[LG08] Sergey Lyskov and Jeffrey J Gray. "The RosettaDock server for local protein-protein docking". en. In: *Nucleic Acids Res.* 36.Web Server issue (July 2008), W233–8.

[Lia+11] S Liang et al. "Fast and accurate prediction of protein side-chain conformations". In: *Bioinformatics* 20 (2011).

[Lin+21] Tianyang Lin et al. "A Survey of Transformers". In: *CoRR* abs/2106.04554 (2021). arXiv: 2106.04554.

[Lin+22] Zeming Lin et al. "Language models of protein sequences at the scale of evolution enable accurate structure prediction". In: *bioRxiv* (2022).

[Liu+17] Ke Liu et al. *Prediction of amino acid side chain conformation using a deep neural network*. 2017. arXiv: 1707.08381 [q-bio.BM].

[LMX22] Boqiao Lai, Matt McPartlon, and Jinbo Xu. "End-to-End deep structure generative model for protein design". In: *bioRxiv* (2022).

[LN98] A J Li and R Nussinov. "A set of van der Waals and coulombic radii of protein atoms for molecular and solvent-accessible surface calculation, packing evaluation, and docking". en. In: *Proteins* 32.1 (July 1998), pp. 111–127.

[Lu+20] Haiying Lu et al. "Recent advances in the development of protein–protein interactions modulators: mechanisms and clinical trials". In: *Signal Transduction and Targeted Therapy* 5.1 (2020), p. 213.

[Luz+21] Marcin Luzarowski et al. "Global mapping of protein–metabolite interactions in Saccharomyces cerevisiae reveals that Ser-Leu dipeptide regulates phosphoglycerate kinase activity". In: *Communications Biology* 4.1 (2021), p. 181.

[LY16] Michael J Lee and Michael B Yaffe. "Protein regulation in signal transduction". en. In: *Cold Spring Harb. Perspect. Biol.* 8.6 (June 2016).

[Mad+23] Ali Madani et al. "Large language models generate functional protein sequences across diverse families". In: *Nature Biotechnology* (2023).

[Man13] Stefano Mangani. "Protein–Protein Interactions in the Solid State: The Troubles of Crystallizing Protein–Protein Complexes". In: ed. by Stefano Mangani. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 113–134.

[Mar+08] Juliette Martin et al. "Structural deformation upon protein-protein interaction: A structural alphabet approach". en. In: *Bmc* 8.12 (2008).

[Mar+15] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015.

[McN+21] Andrew T McNutt et al. "GNINA 1.0: molecular docking with deep learning". en. In: *J. Cheminform.* 13.1 (June 2021), p. 43.

[MD09] David L. Mobley and Ken A. Dill. "Binding of Small-Molecule Ligands to Proteins: "What You See" Is Not Always "What You Get"". In: *Structure* 17.4 (2009), pp. 489–498.

[Mei+21] Joshua Meier et al. "Language models enable zero-shot prediction of the effects of mutations on protein function". In: *Advances in Neural Information Processing Systems* 34 (2021).

[Mic+17] Paulius Micikevicius et al. "Mixed Precision Training". In: *CoRR* abs/1710.03740 (2017). arXiv: 1710.03740.

[Mir+22] Milot Mirdita et al. "ColabFold: making protein folding accessible to all". en. In: *Nat. Methods* 19.6 (June 1, 2022), pp. 679–682.

[Mis+21] Mikita Misiura et al. "DLPacker: Deep Learning for Prediction of Amino Acid Side Chain Conformations in Proteins". In: *bioRxiv* (2021). eprint: https://www.biorxiv.org/content/early/2021/05/25/2021.05.23.445347.full.pdf.

[MK09] Daniel J Mandell and Tanja Kortemme. "Backbone flexibility in computational protein design". en. In: *Curr. Opin. Biotechnol.* 20.4 (Aug. 2009), pp. 420–428.

[MLX22] Matt McPartlon, Ben Lai, and Jinbo Xu. "A Deep SE(3)-Equivariant Model for Learning Inverse Protein Folding". In: *bioRxiv* (2022).

[MX22] Matthew McPartlon and Jinbo Xu. "AttnPacker: An end-to-end deep learning method for rotamer-free protein side-chain packing". In: *bioRxiv* (2022).

[MX23] Matt McPartlon and Jinbo Xu. "Deep Learning for Flexible and Site-Specific Protein Docking and Design". In: *bioRxiv* (2023).

[NRB12] K Nagata, A Randall, and P Baldi. "SIDEpro: a novel machine learning approach for the fast and accurate prediction of side-chain conformations". In: *Proteins* 80.1 (2012), pp. 142–153.

[OJK15] Noah Ollikainen, René M. de Jong, and Tanja Kortemme. "Coupling Protein Side-Chain and Backbone Flexibility Improves the Re-design of Protein-Ligand Specificity". In: *PLoS Computational Biology* 11 (2015).

[Orb+18] Zsuzsanna Orbán-Németh et al. “Structural prediction of protein models using distance restraints derived from cross-linking mass spectrometry data”. In: *Nature Protocols* 13.3 (Mar. 2018), pp. 478–494.

[Pas+19] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[Pie+14] Brian G Pierce et al. “ZDOCK server: interactive docking prediction of protein-protein complexes and symmetric multimers”. en. In: *Bioinformatics* 30.12 (June 2014), pp. 1771–1773.

[PMB13] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “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. Atlanta, GA, USA: JMLR.org, 2013, pp. Iii-1310-iii–1318.

[PST17] Nataraj S Pagadala, Khajamohiddin Syed, and Jack Tuszynski. “Software for molecular docking: a review”. en. In: *Biophys. Rev.* 9.2 (Apr. 2017), pp. 91–102.

[QZ20] Yifei Qi and John Z. H. Zhang. “DenseCPD: Improving the Accuracy of Neural-Network-Based Computational Protein Sequence Design with DenseNet”. In: *Journal of Chemical Information and Modeling* 60.3 (Mar. 2020), pp. 1245–1252.

[Raf+19] Colin Raffel et al. “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”. In: *CoRR* abs/1910.10683 (2019). arXiv: 1910.10683.

[Rao+19] Roshan Rao et al. “Evaluating protein transfer learning with TAPE”. In: *Advances in neural information processing systems* 32 (2019).

[Ren+14] P. Douglas Renfrew et al. “A Rotamer Library to Enable Modeling and Design of Peptoid Foldamers”. In: *Journal of the American Chemical Society* 136.24 (2014). Pmid: 24823488, pp. 8772–8782. eprint: https://doi.org/10.1021/ja503776z.

[Riv+19] Alexander Rives et al. “Biological Structure and Function Emerge from Scaling Unsupervised Learning to 250 Million Protein Sequences”. In: *Pnas* (2019).

[Roc+17] Gabriel J. Rocklin et al. “Global analysis of protein folding using massively parallel design, synthesis, and testing”. In: *Science* 357.6347 (2017), pp. 168–175.

[Rom+21] Robin Rombach et al. “High-Resolution Image Synthesis with Latent Diffusion Models”. In: *CoRR* abs/2112.10752 (2021). arXiv: 2112.10752.

[RS02] Sergei Radaev and Peter D. Sun. “Crystallization of protein–protein complexes”. In: *Journal of Applied Crystallography* 35.6 (Dec. 2002), pp. 674–676.

[Sak+21] Koichiro Saka et al. “Antibody design using LSTM based deep generative model from phage display library for affinity maturation”. en. In: *Scientific Reports* 11.5852 (Mar. 2021).

[SB21] Brandon Charles Seychell and Tobias Beck. “Molecular basis for protein-protein interactions”. en. In: *Beilstein J. Org. Chem.* 17 (Jan. 2021), pp. 1–10.

[Sch+05] Dina Schneidman-Duhovny et al. “PatchDock and SymmDock: servers for rigid and symmetric docking”. en. In: *Nucleic Acids Res.* 33.Web Server issue (July 2005), pp. 363–367.

[Sch+17] Kristof T. Schütt et al. “SchNet : A continuous-filter convolutional neural network for modeling quantum interactions”. In: (2017).

[Sch+18] K. T. Schütt et al. “SchNet - A deep learning architecture for molecules and materials”. In: *The Journal of Chemical Physics* 148.24 (June 2018), p. 241722.

[SD02] Christopher M Summa and William F DeGrado. *protCAD: Protein Computer Aided Design*. 2002. URL: `https://triad.protabit.com/` (visited on 04/06/2023).

[SD11] Maxim V Shapovalov and Roland L Dunbrack Jr. “A smoothed backbone-dependent rotamer library for proteins derived from adaptive kernel density estimates and regressions”. en. In: *Structure* 19.6 (June 2011), pp. 844–858.

[Sha20] Noam Shazeer. “GLU Variants Improve Transformer”. In: *CoRR* abs/2002.05202 (2020).

[SHD22] Cristina Sotomayor-Vivas, Enrique Hernández-Lemus, and Rodrigo Dorantes-Gilardi. “Linking protein structural and functional change to mutation using amino acid networks”. In: *Plos One* 17.1 (Jan. 2022), pp. 1–23.

[SHW21a] Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. “E(n) Equivariant Graph Neural Networks”. In: *CoRR* abs/2102.09844 (2021).

[SHW21b] Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. “E(n) Equivariant Graph Neural Networks”. In: *CoRR* abs/2102.09844 (2021).

[Sim+19] David Simoncini et al. “A structural homology approach for computational protein design with flexible backbone”. In: *Bioinformatics* (2019).

[Sin+19] Priyanka Singh et al. “Determination of Protein–Protein Interactions in a Mixture of Two Monoclonal Antibodies”. In: *Molecular Pharmaceutics* 16.12 (2019). Pmid: 31613625, pp. 4775–4786.

[SS17] Martin Steinegger and Johannes Söding. “MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets”. In: *Nature Biotechnology* 35.11 (2017), pp. 1026–1028.

[Stä+22] Hannes Stärk et al. “EquiBind: Geometric Deep Learning for Drug Binding Structure Prediction”. In: (2022).

[Str+20] Alexey Strokach et al. “Fast and Flexible Protein Design Using Deep Graph Neural Networks”. In: *Cell Systems* 11.4 (2020), 402–411.e4.

[Süd95] Thomas C. Südhof. “The synaptic vesicle cycle: a cascade of protein–protein interactions”. In: *Nature* 375.6533 (1995), pp. 645–653.

[TB05] Dror Tobi and Ivet Bahar. “Structural changes involved in protein binding correlate with intrinsic motions of proteins in the unbound state”. In: *Proceedings of the National Academy of Sciences* 102.52 (2005), pp. 18908–18913.

[Tho+18] Nathaniel Thomas et al. "Tensor Field Networks: Rotation- and Translation-Equivariant Neural Networks for 3D Point Clouds". In: *CoRR* abs/1802.08219 (2018). arXiv: 1802.08219.

[Tor+19] Pedro H. M. Torres et al. "Key Topics in Molecular Docking for Drug Design". In: *International Journal of Molecular Sciences* 20.18 (2019).

[Tow+18] Raphael J. L. Townshend et al. *End-to-End Learning on 3D Protein Structure for Interface Prediction*. 2018.

[Tri+23] Brian L. Trippe et al. "Diffusion Probabilistic Modeling of Protein Backbones in 3D for the motif-scaffolding problem". In: *International Conference on Learning Representations*. 2023.

[van+16] G.C.P. van Zundert et al. "The HADDOCK2.2 Web Server: User-Friendly Integrative Modeling of Biomolecular Complexes". In: *Journal of Molecular Biology* 428.4 (2016). Computation Resources for Molecular Biology, pp. 720–725.

[Var+21] Mihaly Varadi et al. "AlphaFold Protein Structure Database: massively expanding the structural coverage of protein-sequence space with high-accuracy models". In: *Nucleic Acids Research* 50.D1 (Nov. 2021), pp. D439–d444. eprint: https://academic.oup.com/nar/article-pdf/50/D1/D439/43502749/gkab1061.pdf.

[Vas+17] Ashish Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762.

[Vel+18] Petar Veličković et al. *Graph Attention Networks*. 2018. arXiv: 1710.10903 [stat.ML].

[Vre+15] Thom Vreven et al. "Updates to the integrated protein-protein interaction benchmarks: Docking benchmark version 5 and affinity benchmark version 2". en. In: *J. Mol. Biol.* 427.19 (Sept. 2015), pp. 3031–3041.

[Vri+15] Sjoerd J de Vries et al. "A web interface for easy flexible protein-protein docking with ATTRACT". en. In: *Biophys. J.* 108.3 (Feb. 2015), pp. 462–465.

[Wan+04] Junmei Wang et al. "Development and testing of a general amber force field". In: *Journal of computational chemistry* 25.9 (July 2004), pp. 1157–1174.

[Wan+16] Zhe Wang et al. "Comprehensive evaluation of ten docking programs on a diverse set of protein–ligand complexes: the prediction accuracy of sampling power and scoring power". In: *Phys. Chem. Chem. Phys.* 18 (18 2016), pp. 12964–12975.

[Wan+21] Jue Wang et al. "Deep learning methods for designing proteins scaffolding functional sites". In: *bioRxiv* (2021).

[Wat+16] Andrew M. Watkins et al. "Rotamer libraries for the high-resolution design of β-amino acid foldamers". In: *bioRxiv* (2016).

[Wat+22] Joseph L. Watson et al. "Broadly applicable and accurate protein design by integrating structure prediction networks and diffusion generative models". In: *bioRxiv* (2022).

[WBA16] Andrew M. Watkins, Richard Bonneau, and Paramjit S. Arora. "Side-Chain Conformational Preferences Govern Protein-Protein Interactions." In: *Journal of the American Chemical Society* 138 33 (2016), pp. 10386–9.

[WSW10] LM Weiner, R Surana, and S Wang. "Monoclonal antibodies: versatile platforms for cancer immunotherapy." In: *Nat Rev Immunol* 10 (2010), pp. 317–327.

[Wu+22] Ruidong Wu et al. "High-resolution de novo structure prediction from primary sequence". In: *bioRxiv* (2022). eprint: `https://www.biorxiv.org/content/early/2022/07/22/2022.07.21.500999.full.pdf`.

[XB06] Jinbo Xu and Bonnie Berger. "Fast and Accurate Algorithms for Protein Side-Chain Packing". In: *J. Acm* 53.4 (July 2006), pp. 533–557.

[Xio+14] Peng Xiong et al. "Protein design with a comprehensive statistical energy function and boosted by experimental selection for foldability." In: *Nature communications* 5 (2014), p. 5330.

[Xio+20] Ruibin Xiong et al. "On Layer Normalization in the Transformer Architecture". In: *CoRR* abs/2002.04745 (2020).

[XWM20] Gang Xu, Qinghua Wang, and Jianpeng Ma. "OPUS-Rota3: Improving protein side-chain modeling by deep neural networks and ensemble methods". In: *Journal of Chemical Information and Modeling* 60 12 (2020), pp. 6691–6697.

[XWM21] Gang Xu, Qinghua Wang, and Jianpeng Ma. "OPUS-Rota4: A Gradient-Based Protein Side-Chain Modeling Framework Assisted by Deep Learning-Based Predictors". In: *bioRxiv* (2021). eprint: `https://www.biorxiv.org/content/early/2021/07/23/2021.07.22.453446.full.pdf`.

[Yan+17] Yumeng Yan et al. "HDOCK: a web server for protein-protein and protein-DNA/RNA docking based on a hybrid strategy". en. In: *Nucleic Acids Res.* 45.W1 (July 2017), W365–w373.

[Yan+20a] Yumeng Yan et al. "The HDOCK server for integrated protein–protein docking". In: *Nature Protocols* 15.5 (May 2020), pp. 1829–1852.

[Yan+20b] Jianyi Yang et al. "Improved protein structure prediction using predicted interresidue orientations". In: *Proceedings of the National Academy of Sciences* 117.3 (2020), pp. 1496–1503. eprint: `https://www.pnas.org/content/117/3/1496.full.pdf`.

[Yin+21] Chengxuan Ying et al. "Do Transformers Really Perform Bad for Graph Representation?" In: *CoRR* abs/2106.05234 (2021). arXiv: `2106.05234`.

[Yin+22] Rui Yin et al. "Benchmarking AlphaFold for protein complex modeling reveals accuracy determinants". en. In: *Protein Sci.* 31.8 (Aug. 2022), e4379.

[YSW07] Chen Yanover, Ora Schueler-Furman, and Yair Weiss. "Minimizing and Learning Energy Functions for Side-Chain Prediction". In: *Recomb.* 2007.

[YSW08] Chen Yanover, Ora Schueler-Furman, and Yair Weiss. "Minimizing and learning energy functions for side-chain prediction". en. In: *J. Comput. Biol.* 15.7 (Sept. 2008), pp. 899–911.

[YZY22] Kevin K. Yang, Niccolò Zanichelli, and Hugh Yeh. "Masked inverse folding with sequence transfer for protein representation learning". In: *bioRxiv* (2022).

[Zou+18] Xu-Dong Zou et al. "PPI network analyses of human WD40 protein family systematically reveal their tendency to assemble complexes and facilitate the complex predictions". In: *BMC Systems Biology* 12.4 (Apr. 2018), p. 41.

[ZS04] Yang Zhang and Jeffrey Skolnick. "Scoring function for automated assessment of protein structure template quality". In: *Proteins* 57.4 (Dec. 2004), pp. 702–710.