A ARCHITECTURE DETAILS

For learning the surrogate potential f_θ, we initially looked at the both the E-GNN architectures and architectures based on tensor convolutions, as in the `e3nn` library (Geiger & Smidt, 2022), and used in other docking architectures (Corso et al., 2023; Ketata et al., 2023). In practice, we found the architectures based on tensor convolutions to learn much faster and generalize better, and use it for all our experiments.

A.1 GRAPH REPRESENTATIONS

Proteins are represented as 3D graphs where nodes correspond to amino acid residues, located at the position of the Cα atom. Edges are added between residues from the same and different proteins by specifying appropriate radius cutoffs and a maximum allowed number of neighbors. We employ the following cutoff constraints to build graphs on the fly:

1. For graphs between residues from the same protein, we use a radius cutoff of 20.0 with a maximum of 30 neighbors.
2. For graphs between residues from different proteins, we consider two settings. For the model described in Section 4.2, we use a radius threshold of 80.0, with a maximum of 500 neighbors. For the model described in Section 4.3, we follow a similar argument as to Corso et al. (2023) and use a dynamic cutoff $40.0 + 3\sigma_{tr}$Å, where σ_{tr} represents the standard deviation of the diffusion noise from the translation component.

Notations We use $[N]$ to denote the set $\{1, 2, \ldots, N\}$. For each protein $i \in [N]$, let $(\mathcal{V}_i, \mathcal{E}_i)$ denote its corresponding graph representation, where $\mathcal{E}_i = (\mathcal{E}_{\text{self, i}}, \mathcal{E}_{\text{cross, i}})$, denoting the set of self (intra-protein) and cross edges (inter-protein) respectively. Let $\mathcal{V}$, $\mathcal{E}_{\text{self}}$ and $\mathcal{E}_{\text{cross}}$ denote the set of nodes, self and cross edges for proteins $[N]$ respectively.

A.2 NODE AND EDGE FEATURES

Node Features For each residue, node features include a one-hot encoding of the residue type, the hydrophobicity value (as measured by the Kyle-DooLittle scale), volume, charge, the polarity of the residue, and whether the residue is a donor or acceptor (or both). The hydrophobicity, volume and charge features were expanded into a radial basis (add values later for ranges). Our initial experiments also used language model features from ESM2, but a significant performance gain was not noticed. The residue features were concatenated with sinusoidal embeddings of the diffusion time t, with an embedding scale of 10000 and an embedding size of 32. The total number of residue features (including time embeddings) was $177 (145 + 32)$.

Edge Features Edges were encoded using a radial basis expansion upto a fixed cutoff. For edges between residues of the same protein, we used a cutoff of 20.0, while for edges between residues from different proteins, we used a cutoff of 80.0. We did not notice any significant improvement from using sinusoidal embeddings of sequence positions in the edges for the same protein. The total number of edge features used for all graphs was 32.

A.3 CONVOLUTIONAL LAYERS

Our architecture retains the same message-passing components as Corso et al. (2023), with variations in the output layer (Appendix A.4) depending on whether f is learnt directly using supervision from PYROSETTA or implicitly through the score network. For completeness, we include the details of the message-passing architecture here, but interested readers can also refer to Corso et al. (2023) for the same. We use $\mathbf{h}^{i,(l)}$ to denote the node representations of protein i at layer l, and $\mathbf{h}_k^{i,(l)}$ to denote the representations of node $k \in [n_i]$ of protein i.

At every layer, the node features form a concatenated tensor of scalars, vectors and other parity-driven combinations, each of which transform differently under the action of rotations and translations. The message passing between nodes across all graphs uses tensor products on these features with the spherical harmonic representation of the (normalized) edge vector. The learnable set of

weights associated with the tensor products are informed by the edge features (from Appendix A.2), and the scalar feature components in the current node features. These messages are then aggregated (addition) at every node, and used to update the current node features.

$$\mathbf{h}_k^{i,(l+1)} = \mathbf{h}_k^{i,(l)} \oplus \mathbf{h}_{k,\text{self}}^{i,(l)} \oplus \mathbf{h}_{k,\text{cross}}^{i,(l)} \tag{4}$$

$$\mathbf{h}_{k,\text{self}}^{i,(l)} = \frac{1}{|\mathcal{N}_{k,\text{self}}^i|} \sum_{j \in \mathcal{N}_{k,\text{self}}^i} Y(\hat{r}_{kj}) \otimes_{\psi_{kj,i}^{\text{self}}} \mathbf{h}_j^{i,(l)} \tag{5}$$

$$\mathbf{h}_{k,\text{cross}}^{i,(l)} = \frac{1}{|\mathcal{N}_{k,\text{cross}}^i|} \sum_{j \in \mathcal{N}_{k,\text{cross}}^i} Y(\hat{r}_{kj}) \otimes_{\psi_{(k,i),(j,i_j)}^{\text{cross}}} \mathbf{h}_j^{i_j,(l)} \tag{6}$$

$$\psi_{kj,i}^{\text{self}} = \Psi(e_{kj}^i, \mathbf{h}_k^{i,(l,\text{scalar})}, \mathbf{h}_j^{i,(l,\text{scalar})}) \tag{7}$$

$$\psi_{(k,i),(j,i_j)}^{\text{cross}} = \Psi(e_{kj}^{(i,i_j)}, \mathbf{h}_k^{i,(l,\text{scalar})}, \mathbf{h}_j^{i_j,(l,\text{scalar})}) \tag{8}$$

where $\mathcal{N}_{k,\text{self}}^i$ and $\mathcal{N}_{k,\text{cross}}^i$ denote intra-protein and inter-protein neighbors for node k in protein i. Φ^{self}, Ψ^{cross} correspond to MLPs with learnable weights for the graphs between the same protein and different proteins respectively, and Y refers to the spherical harmonics upto $l = 2$. The convolutional layers use these weights to generate scalar & vector representations for each node.

A.4 Output Layers

A.4.1 Learning the Potential function f_θ

We use the node features $\{\mathbf{h}^{i,(L)}\}_{i=1}^N$, after L layers to predict a scalar output. Intuitively, we can think of this scalar capturing the energetics/and or other desirable properties of the set of proteins The scalar output is predicted as follows:

$$s = s_{\text{self}} + s_{\text{cross}} + s_{\mathcal{V}} \tag{9}$$

$$s_{\text{self}} = \frac{1}{|\mathcal{E}_{\text{self}}|} \sum_{(k,j) \in \mathcal{E}_{\text{self}}} \text{MLP}_s(\mathbf{h}_k^{i_k,(L,\text{scalar})}, \mathbf{h}_j^{i_k,(L,\text{scalar})}) \tag{10}$$

$$s_{\text{cross}} = \frac{1}{|\mathcal{E}_{\text{cross}}|} \sum_{(k,j) \in \mathcal{E}_{\text{cross}}} \text{MLP}_c(\mathbf{h}_k^{i_k,(L)}, \mathbf{h}_j^{i_j,(L)}) \tag{11}$$

$$s_{\mathcal{V}} = \frac{1}{|\mathcal{V}|} \sum_{k \in \mathcal{V}} \text{MLP}_v(\mathbf{h}_k^{i_k,(L,\text{scalar})}) \tag{12}$$

where $\text{MLP}_s, \text{MLP}_v, \text{MLP}_v$ are fully-connected networks with learnable weights for energetic contributions from intra-protein edges, inter-protein edges and residues respectively.

A.4.2 Learning the Score Network s_θ

We use the node features $\{\mathbf{h}^{i,(L)}\}_{i=1}^N$ after L layers to predict the translational and rotational scores for all non-stationary proteins relative to the stationary protein. We follow the same intuition as Corso et al. (2023) in that the translational and rotational scores represent the linear acceleration of the center of mass, and the angular acceleration of the rest of the protein around the center. Similar to them, we apply a convolution of $h^{(L)}$ about the center of mass of the system, with the natural adaptation to the multi-agent setting. For completeness, this convolution is described by the following equation, for the center of mass c,

$$\mathbf{v}^i = \frac{1}{|\mathcal{V}_i|} \sum_{k \in \mathcal{V}_i} Y(\hat{r}_{ck,i}) \otimes_{\psi_{ck,i}} \mathbf{h}_k^{i,(L)} \tag{13}$$

$$\psi_{ck,i} = \Psi(e_{ck}^i, \mathbf{h}_k^{i,(L)}) \tag{14}$$

The output of $\mathbf{v}^i$ is restricted to a single even and single odd vector, for each score, which are then summed up, owing to the coarse-grained representation of the protein (in terms of Cα atom graphs). The magnitudes of the two scores are adjusted with an MLP and the final outputs are normalized by scaling the translational score with $\frac{1}{\sigma_{\text{tr}}}$ and the rotational score by the expected magnitude of the $SO(3)$ score for σ_{rot}.

B EXPERIMENTAL DETAILS

B.1 DATASETS

In our experiments, we use the Dataset of Interacting Protein Structures (DIPS) (Townshend et al., 2019) and the Docking Benchmark 5.5 (DB5.5) (Vreven et al., 2015). We utilize the same protein-family splits as Ganea et al. (2022) for DIPS, and evaluate on the DB5.5 test set. DIPS is a large dataset with 42826 examples, well suited for the rigid binary docking task, with each example containing single-chain proteins. DB5.5 is a gold standard dataset used for evaluating docking methods, but only contains 253 examples. While DB5.5 has been traditionally used in the context of binary protein docking, many examples in DB5.5 consist of proteins made up of multiple chains. This allows us to utilize DB5.5 to evaluate multimeric docking. For DB5, we download the data and protein structures from `https://github.com/octavian-ganea/equidock_public`. For DIPS, we download the processed datasets from the link provided under `https://github.com/ketatam/DiffDock-PP`.

B.2 TRAINING AND EQUILIBRIUM COMPUTATION

B.2.1 TRAINING

Potential Network Training the potential network required an additional step in generating *decoys* – these are perturbed assemblies generated by applying a randomly sampled rotation and translation to the original assembly structure. For the DIPS dataset, we only had 5000 examples for which PYROSETTA was able to generate decoys in reasonable time, and we used the same examples for training both the score and potential network. For each training example, we generated 10 pairs of decoys which were scored with PYROSETTA, and utilized these to train our model with the ranking loss described in Equation 1. The potential network was trained for 10 epochs using the Adam (Kingma & Ba, 2014) optimizer with no weight decay using 4 32GB V100 GPUs. The best validation model (according to validation loss computed on decoys generated from the DB5.5 validation set) was used for equilibrium computation.

Score Network The score network was first trained on DIPS, and then finetuned on DB5.5. Across both DIPS and DB5.5, the score network was trained at the granularity of protein chains (2 chains for DIPS, and ranging between 2-8 for DB5.5). We use Adam (Kingma & Ba, 2014) as our optimizer with no weight decay. The score network was trained for 100 epochs on 4 32GB V100 GPUs. As described above, we trained our model only on 5000 examples from DIPS with, each example repeated twice. We used the model with the best validation loss as the finetuning model for DB5.5. The score network was then finetuned on DB5.5 for 100 epochs, also using 4 32GB V100 GPUs. We used a constant batch size of 4 across both training and finetuning. When finetuning on DB5.5, we ran inference on 10 complexes in the validation set every 10 epochs, using the C-RMSD percentage < 5Åto save the best inference model observed so far. The best inference model after training was used for equilbrium computation.

B.2.2 EQUILIBRIUM COMPUTATION

Equilibrium computation was carried out on a single GPU wherever applicable.

Gradient-Based Learning Approach We generate 20 equilibria for each complex, using the gradient-based updates of Equation 2 to update the roto-translational actions of all the protein chains (except one receptor chain which we keep fixed). We set $\lambda = 0.5$, learning rate $\eta^t = t^{-0.5}$, and perform 60 gradient steps.

Diffusion Generative Model Approach Here, equilibrium computation corresponds to reverse diffusion by solving the (discretized) reverse SDE with the score network finetuned on DB5.5. As our goal was to to evaluate multimeric docking, we ran inference at the granularity of protein chains, different from the traditional evaluation setting of binary dockin, which other baselines adopt. We generate 40 equilibria during inference, with the reverse diffusion running for 50 steps.

B.3 HYPERPARAMETERS

Potential Network For the potential network, we used 16 scalar features and 4 vector features, with 4 tensor product convolution layers, and a dropout rate of 0.1. We use learning rate of 0.001.

Score Network For the score network, we used 16 scalar features and 4 vector features, with 4 tensor product convolution layers, and a dropout rate of 0.1, similar to the potential network. The total number of parameters in the model was 0.243M. Our hyperparameter tuning was limited to the learning rate for the score network trained on the DIPS dataset (0.001, **0.0005**, 0.0003). For the translation variance schedule, we used a $\sigma_{\text{max, tr}}$ of 25.0 and a $\sigma_{\text{min, tr}}$ of 0.01. For the rotation variance, we used a $\sigma_{\text{max, rot}}$ of 1.65 and a $\sigma_{\text{min, rot}}$ of 0.01.

B.4 BASELINES

CLUSPRO, EQUIDOCK, ATTRACT, PATCHDOCK . For these baselines, we used the provided prediction files from `https://github.com/octavian-ganea/equidock_public` to compute additional metrics such as TM-Score.

DIFFDOCK-PP We used the code associated with the paper (Ketata et al., 2023) at `https://github.com/ketatam/DiffDock-PP`. We use the `db5_esm_inference` configuration to run inference, after modifying the parameters ns $= 32$ and nv $= 6$. Inference was run using the `large_model_dips` without any additional finetuning on the DB5.5 dataset.

MULTI-LZERD We downloaded the relevant binaries from `https://kiharalab.org/proteindocking/multilzerd.php` to run MULTI-LZERD. Empirically, we found MULTI-LZERD to be extremely slow, and had to submit parallel jobs for the DB5.5 test complexes, and also modified the parameters `generations=50` and `population=50`.

ALPHAFOLD-MULTIMER, SYNDOCK Both and SYNDOCK are relevant baselines for multimer docking. We however do not compare to ALPHAFOLD-MULTIMER as the DB5.5 test set is part of their training set, leading to potentially over-optimistic numbers. For SYNDOCK, there was no open-source implementation available at the time of writing this paper. We will add a relevant comparison once their implementation is open-sourced.

Runtimes For CLUSPRO, EQUIDOCK, ATTRACT, PATCHDOCK, we used the runtimes are reported in Ganea et al. (2022). For DiffDock-PP, we use the runtime as reported by the inference script. For MULTI-LZERD, since we submitted multiple jobs in parallel (to speed up inference), we used the average of the wall-clock times of all jobs.

B.5 CODE

All our code used in this work is available at `https://anonymous.4open.science/r/iclr24-dockgame/`

C ADDITIONAL ANALYSES

C.1 PYROSETTA AS A GROUND-TRUTH POTENTIAL

Traditional physics-based energy functions are commonplace, which are tuned such that the available crystal structures have low energy. For this work, we use PYROSETTA, a commonly used protein modelling suite with several energy functions. In particular, we use the `interchain_cen`

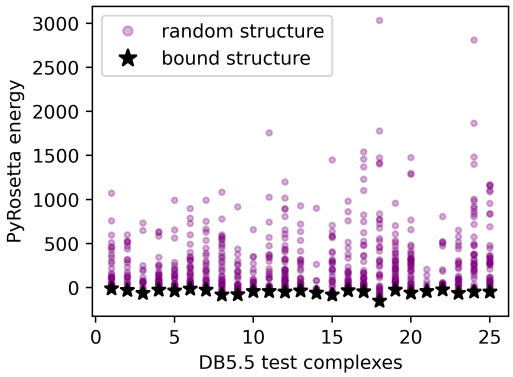

Figure 3: PyRosetta energy of DB5.5 bound complexes and of 50 random perturbations (i.e. applying random roto-traslational actions to the protein chains) for each of them. Bound complexes in the data are found to have the lowest PyRosetta energy.

function tailored for docking. In Figure 3 we plot the PyRosetta energy for each bound complex in the DB5.5 test dataset, as well as energies computed for 50 random complex perturbations. This backs up our underlying assumption that ground-truth assemblies have low PyRosetta energy.