DiffDock: Diffusion Steps, Twists, and Turns for Molecular Docking

Gabriele Corso[*], **Hannes Stärk**[*], **Bowen Jing**[*], **Regina Barzilay** & **Tommi Jaakkola**
CSAIL, Massachusetts Institute of Technology

Abstract

Predicting the binding structure of a small molecule ligand to a protein—a task known as *molecular docking*—is critical to drug design. Recent deep learning methods that treat docking as a regression problem have decreased runtime compared to traditional search-based methods but have yet to offer substantial improvements in accuracy. We instead frame molecular docking as a *generative* modeling problem and develop DIFFDOCK, a diffusion generative model over the non-Euclidean manifold of ligand poses. To do so, we map this manifold to the product space of the degrees of freedom (translational, rotational, and torsional) involved in docking and develop an efficient diffusion process on this space. Empirically, DIFFDOCK obtains a 38% top-1 success rate (RMSD<2Å) on PDBBind, significantly outperforming the previous state-of-the-art of traditional docking (23%) and deep learning (20%) methods. Moreover, DIFFDOCK has fast inference times and provides confidence estimates with high selective accuracy.

1 Introduction

A crucial task in computational drug design is *molecular docking*—predicting the position, orientation, and conformation of a ligand when bound to a target protein—from which the effect of the ligand (if any) might be inferred. Traditional approaches for docking [1, 2] rely on scoring-functions that estimate the correctness of a proposed structure, or pose, and an optimization algorithm that searches for the global maximum of the scoring function. However, since the search space is vast and the landscape of the scoring functions rugged, these methods tend to be too slow and inaccurate.

We develop DIFFDOCK, a diffusion *generative* model (DGM) over the space of ligand poses for molecular docking. We define a diffusion process over the degrees of freedom involved in docking: the position of the ligand relative to the protein (locating the binding pocket), its orientation in the pocket, and the torsion angles describing its conformation. DIFFDOCK samples poses by running the learned (reverse) diffusion process, which can be intuitively viewed as the progressive refinement of random poses via updates of their translations, rotations, and torsion angles (Figure 1).

Empirically, on the standard blind docking benchmark PDBBind, DIFFDOCK achieves 38% of top-1 predictions with ligand root mean square distance (RMSD) below 2Å, nearly doubling the performance of the previous state-of-the-art deep learning model (20%). DIFFDOCK significantly outperforms even state-of-the-art search-based methods (23%), while still being 3 to 12 times faster on GPU. Moreover, it provides an accurate confidence score of its predictions, obtaining 80% RMSD<2Å on its most confident third of the previously unseen complexes.

[*]Equal contribution. Correspondance to `{gcorso, hstark, bjing}@mit.edu`.

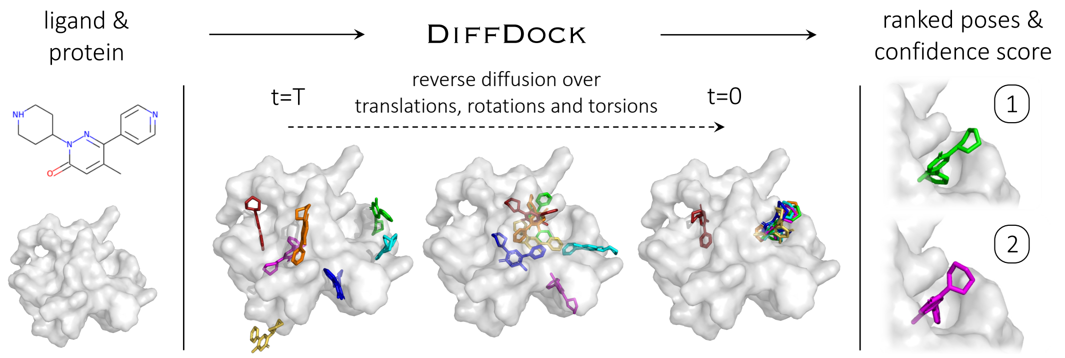
ligand & protein
DiffDock
ranked poses & confidence score
t=T
reverse diffusion over translations, rotations and torsions
t=0
1
2

Figure 1: Overview of DiffDock. *Left*: The model takes as input the separate ligand and protein structures. *Center*: Randomly sampled initial poses are denoised via a reverse diffusion over translational, rotational, and torsional degrees of freedom. *Right:*. The sampled poses are ranked by the confidence model to produce a final prediction and confidence score.

2 Method

We can regard a pose $\mathbf{x}$ as an element in $\mathbb{R}^{3n}$. However, this encompasses far more degrees of freedom than are relevant. Traditional docking methods, as well as most ML ones, take as input a seed conformation $\mathbf{c} \in \mathbb{R}^{3n}$ and change only the relative position and the torsion degrees of freedom. The space of ligand poses consistent with $\mathbf{c}$ is, therefore, an $(m+6)$-dimensional submanifold $\mathcal{M}_\mathbf{c} \subset \mathbb{R}^{3n}$. We follow this paradigm, and formulate molecular docking as learning a probability distribution $p_\mathbf{c}(\mathbf{x} \mid \mathbf{y})$ over the manifold $\mathcal{M}_\mathbf{c}$, conditioned on a protein structure $\mathbf{y}$.

DGMs on submanifolds have been formulated by [3] in terms of projecting a diffusion in ambient space onto the submanifold. However, the kernel $p(\mathbf{x}_t \mid \mathbf{x}_0)$ of such a diffusion is not available in closed form and must be sampled numerically with a geodesic random walk. We instead define a one-to-one mapping to another, "nicer" manifold where the diffusion kernel can be sampled directly and develop a DGM in that manifold. Specifically, we define a continuous family of ligand pose transformations corresponding to the $m+6$ degrees of freedom, and use it to lift points on $\mathcal{M}_\mathbf{c}$ to the product space of the corresponding groups—which is itself a manifold.

Ligand pose transformations We associate translations of ligand position with the 3D translation group $\mathbb{T}(3)$, rigid rotations of the ligand with the 3D rotation group $SO(3)$, and changes in torsion angles at each rotatable bond with a copy of the 2D rotation group $SO(2)$. More formally, we define operations of each of these groups on a ligand pose $\mathbf{c} \in \mathbb{R}^{3n}$. The translation $A_{\text{tr}} : \mathbb{T}(3) \times \mathbb{R}^{3n} \to \mathbb{R}^{3n}$ is defined straightforwardly as $A_{\text{tr}}(\mathbf{r}, \mathbf{x})_i = \mathbf{x}_i + \mathbf{r}$ using the isomorphism $\mathbb{T}(3) \cong \mathbb{R}^3$ where $\mathbf{x}_i \in \mathbb{R}^3$ is the position of the ith atom. Similarly, the rotation $A_{\text{rot}} : SO(3) \times \mathbb{R}^{3n} \to \mathbb{R}^{3n}$ is defined by $A_{\text{rot}}(R, \mathbf{x})_i = R(\mathbf{x}_i - \bar{\mathbf{x}}) + \bar{\mathbf{x}}$ where $\bar{\mathbf{x}} = \frac{1}{n}\sum \mathbf{x}_i$, corresponding to rotations around the (unweighted) center of mass of the ligand. Many valid definitions of a change in torsion angles are possible, as the torsion angle around any bond (a_i, b_i) can be updated by rotating the a_i side, the b_i side, or both. To disentangle it from rotations or translations, we define changes of torsion angles such that it causes a minimal perturbation (in an RMSD sense) to the structure:

Definition. *Let $B_{k,\theta_k}(\mathbf{x}) \in \mathbb{R}^{3n}$ be any valid torsion update by θ_k around the kth rotatable bond (a_k, b_k). We define $A_{tor} : SO(2)^m \times \mathbb{R}^{3n} \to \mathbb{R}^{3n}$such that $A_{tor}(\boldsymbol{\theta}, \mathbf{x}) = \text{RMSDAlign}(\mathbf{x}, (B_{1,\theta_1} \circ \cdots B_{m,\theta_m})(\mathbf{x}))$ where $\boldsymbol{\theta} = (\theta_1, \ldots \theta_m)$ and $\text{RMSDAlign}(\mathbf{x}, \mathbf{x}') = \arg\min_{\mathbf{x}^\dagger \in \{g\mathbf{x}' \mid g \in SE(3)\}} \text{RMSD}(\mathbf{x}, \mathbf{x}^\dagger)$*

This means that we apply all the m torsion updates in any order and then perform a global RMSD alignment with the unmodified pose. The definition is motivated by ensuring that the infinitesimal effect of a torsion is orthogonal to any rototranslation, i.e., it induces no *linear or angular momentum*:

Proposition 1. *Let $\mathbf{x}(t) := A_{tor}(t\boldsymbol{\theta}, \mathbf{x})$ for some $\boldsymbol{\theta}$ and where $t\boldsymbol{\theta} = (t\theta_1, \ldots t\theta_m)$. Then the linear and angular momentum are zero: $\frac{d}{dt}\bar{\mathbf{x}}|_{t=0} = 0$ and $\sum_i (\mathbf{x}_i - \bar{\mathbf{x}}) \times \frac{d}{dt}\mathbf{x}_i|_{t=0} = 0$ where $\bar{\mathbf{x}} = \frac{1}{n}\sum_i \mathbf{x}$.*

Now consider the product space $\mathbb{P} = \mathbb{T}^3 \times SO(3) \times SO(2)^m$ and define $A : \mathbb{P} \times \mathbb{R}^{3n} \to \mathbb{R}^{3n}$ as $A((\mathbf{r}, R, \boldsymbol{\theta}), \mathbf{x}) = A_{\text{tr}}(\mathbf{r}, A_{\text{rot}}(R, A_{\text{tor}}(\boldsymbol{\theta}, \mathbf{x})))$. These definitions collectively provide the soughtafter product space corresponding to the docking degrees of freedom. Indeed, for a seed ligand conformation $\mathbf{c}$, we can formally define the space of ligand poses $\mathcal{M}_{\mathbf{c}} = \{A(g, \mathbf{c}) \mid g \in \mathbb{P}\}$.

We now proceed to show how the product space can be used to learn a DGM over ligand poses in $\mathcal{M}_{\mathbf{c}}$. First, we need a theoretical result (proof in Appendix A):

Proposition 2. *For a given seed conformation* $\mathbf{c}$*, the map* $A(\cdot, \mathbf{c}) : \mathbb{P} \to \mathcal{M}_{\mathbf{c}}$ *is a bijection.*

which means that the inverse $A_{\mathbf{c}}^{-1} : \mathcal{M}_{\mathbf{c}} \to \mathbb{P}$ given by $A(g, \mathbf{c}) \mapsto g$ maps ligand poses $\mathbf{x} \in \mathcal{M}_{\mathbf{c}}$ to points on the product space $\mathbb{P}$. We are now ready to develop a diffusion process on $\mathbb{P}$.

To implement a diffusion model on $\mathbb{P}$, it suffices to develop a method for sampling from and computing the score of the diffusion kernel on $\mathbb{P}$. Furthermore, since $\mathbb{P}$ is a product manifold, the forward diffusion proceeds independently in each manifold [4], and the tangent space is a direct sum: $T_g\mathbb{P} = T_{\mathbf{r}}\mathbb{T}_3 \oplus T_R SO(3) \oplus T_{\boldsymbol{\theta}} SO(2)^m \cong \mathbb{R}^3 \oplus \mathbb{R}^3 \oplus \mathbb{R}^m$ where $g = (\mathbf{r}, R, \boldsymbol{\theta})$. Thus, it suffices to sample from the diffusion kernel and regress against its score in each group independently.

In all three groups, we define the forward SDE as $d\mathbf{x} = \sqrt{d\sigma^2(t)/dt}\, d\mathbf{w}$ where $\sigma^2 = \sigma^2_{\text{tr}}$, σ^2_{rot}, or σ^2_{tor} for $\mathbb{T}(3)$, $SO(3)$, and $SO(2)^m$ respectively and where $\mathbf{w}$ is the corresponding Brownian motion. Since $\mathbb{T}(3) \cong \mathbb{R}^3$, the translational case is trivial. The diffusion kernel on $SO(3)$ is given by the $IGSO(3)$ distribution [5, 6] and its score is $\nabla \ln p_t(R' \mid R) = (\frac{d}{d\omega} \log f(\omega))\hat{\omega} \in T_{R'} SO(3)$, where $R' = \mathbf{R}(\omega\hat{\omega})R$ is the result of applying Euler vector $\omega\hat{\boldsymbol{\omega}}$ to R. Finally, the $SO(2)^m$ group is diffeomorphic to the torus $\mathbb{T}^m$, on which the diffusion kernel is a *wrapped normal distribution* with variance $\sigma^2(t)$.

Architecture Although we have defined the diffusion kernel and score matching objectives on $\mathbb{P}$, we nevertheless develop the training and inference procedures to operate on ligand poses in 3D coordinates directly to leverage $SE(3)$ equivariant networks. The score model and confidence model have similar architectures except for the space of their output. The output of the score model must be in the tangent space $T_{\mathbf{r}}\mathbb{T}_3 \oplus T_R SO(3) \oplus T_{\boldsymbol{\theta}} SO(2)^m$, which corresponds to equivariant translation and rotation (Euler) vectors, and scores on invariant torsion angles. The confidence model outputs a single invariant scalar. The architectural components are detailed in Appendix C.

Training and inference The training and inference procedures technically depend on the choice of seed conformation $\mathbf{c}$ used to define the mapping between $\mathcal{M}_{\mathbf{c}}$ and the product space. However, providing a definite choice of $\mathbf{c}$ to the score model introduces an arbitrary inference-time parameter that may affect the final predicted distribution, which is undesirable. Thus, we develop approximate training and inference procedures that remove the dependence on the $\mathbf{c}$, which work quite well in practice. See Appendix B for further details.

Confidence model. With a trained diffusion model, one can sample an arbitrary number of ligand poses from the posterior distribution. However, researchers are often interested in seeing only one or a small number of predicted poses and an associated confidence measure for downstream analysis. Thus, we train a confidence model over the poses sampled by the diffusion model and rank them based on its confidence that they are within the error tolerance. The top-ranked ligand pose and the associated confidence are then taken as DIFFDOCK's top-1 prediction and confidence score.

In order to collect training data for the confidence model $\mathbf{d}(\mathbf{x}, \mathbf{y})$, we run the trained diffusion model to obtain a set of candidate poses for every training example and generate labels by testing whether or not each pose has RMSD below 2Å. The confidence model is then trained with cross-entropy loss to correctly predict the binary label for each pose. During inference, the diffusion model is run to generate N poses in parallel, which are passed to the confidence model that ranks them based on its confidence that they have RMSD below 2Å.

3 Experiments

Experimental setup. We evaluate our method on the complexes from PDBBind [7], a large collection of protein-ligand structures collected from PDB [8], which was used with time-based splits to benchmark many previous works [9, 10, 11]. We test models in the setting of blind docking, where, as commonly done, we provide to the methods the bound protein structure but no information on the ligand pose or pocket. Details about the experimental setup, data, baselines, and implementation are in Appendix D.3 and all code is available at `anonymous.4open.science/r/DiffDock`.

Table 1: **PDBBind blind docking.** Percentage of predictions with RMSD < 2Å and the median RMSD. For our method in parenthesis we specify the number of poses sampled from the generative model. top-1 refers to the highest ranked prediction, top-5 to the most accurate pose out of the first 5. * indicates that the method runs exclusively on CPU, "-" means not applicable; some cells are empty due to infrastructure constraints. For TANKBind, the runtimes for the top-1 and top-5 predictions are different. Further baselines and evaluation metrics are in Appendix E.

	Top-1 RMSD (Å)		Top-5 RMSD (Å)		Average
Method	%<2	Med.	%<2	Med.	Runtime (s)
QVINAW [12]	20.9	7.7			49*
GNINA [13]	22.9	7.7	32.9	4.5	127
SMINA [14]	18.7	7.1	29.3	4.6	126*
GLIDE [2]	21.8	9.3			1405*
EQUIBIND [9]	5.5	6.2	-	-	**0.04**
TANKBIND [11]	20.4	4.0	24.5	3.4	0.7/2.5
DIFFDOCK (10)	34.5±0.3	3.61±0.05	40.1±1.1	2.79±0.09	10
DIFFDOCK (40)	**37.5±0.5**	**3.46**±**0.04**	**43.3±0.5**	**2.53±0.05**	40

Accuracy and runtime. DIFFDOCK significantly outperforms all previous methods (Table 1). In particular, DIFFDOCK obtains an impressive 37.5% top-1 success rate (i.e., percentage of predictions with RMSD <2Å) when sampling 40 poses and 34.5% when sampling just 10. This performance vastly surpasses that of state-of-the-art commercial software such as GLIDE (21.8%) and the previous state-of-the-art deep learning method TANKBind (20.4%). DIFFDOCK holds its superior accuracy while being (on GPU) 3 to 12 times faster than the best search-based method, GNINA (Table 1).

Selective accuracy of confidence score. We investigate the *selective accuracy* of the confidence model across *different* complexes by evaluating how DIFFDOCK's accuracy increases if it only makes predictions when the confidence is above a certain threshold, known as *selective prediction*. In Figure 2, we plot the success rate as we decrease the percentage of complexes for which we make predictions, i.e., increase the confidence threshold. When only making predictions for the top one-third of complexes in terms of model confidence, the success rate improves from 38% to 80%. Additionally, there is a high Spearman correlation of 0.74 between DIFFDOCK's confidence and the negative RMSD. Thus, the confidence score is a good indicator of the quality of DIFFDOCK's top-ranked sampled pose and provides a highly valuable confidence measure for downstream applications.

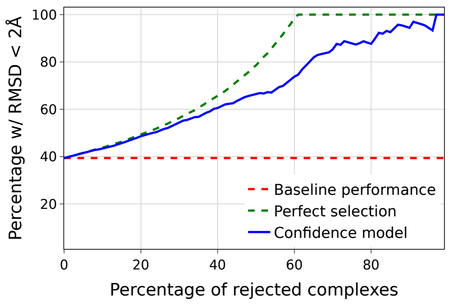

Figure 2: **Selective accuracy.** Percentage of predictions with RMSD below 2Å when only making predictions for the portion of the dataset where DIFFDOCK is most confident.

4 Conclusion

We presented DIFFDOCK, a diffusion generative model tailored to the task of molecular docking. This represents a paradigm shift from previous deep learning approaches, which use regression-based frameworks, to a generative modeling approach that is better aligned with the objective of molecular docking. To produce a fast and accurate generative model, we designed a diffusion process over the manifold describing the main degrees of freedom of the task via ligand pose transformations spanning the manifold. Empirically, DIFFDOCK outperforms the state-of-the-art by very large margins on PDBBind, has fast inference times, and provides confidence estimates with high selective accuracy. Thus, DIFFDOCK can offer great value for many existing real-world pipelines and opens up new avenues of research on how to best integrate downstream tasks, such as affinity prediction, into the framework and apply similar ideas to protein-protein and protein-nucleic acid docking.

5 Broader Impact

Molecular docking is a crucial task for computational drug discovery. Thus, a method like DiffDock that speeds up blind docking and improves the accuracy can help pave the path towards faster drug discovery. The blind docking task is also crucial for discovering mechanisms of function of a small molecule and these insights can lead to generally better treatment developments. While these are large possible positive impacts on society, there is also a big potential for negative impact: potentially practitioners might blindly trust the predictions of DiffDock and make incorrect or not careful enough decision based on them. We hope the confidence estimates of our method are able to mitigate this but still caution users against blindly trusting deep learning predictions, including ours.

Acknowledgments

We pay tribute to Octavian-Eugen Ganea (1987-2022), dear colleague, mentor, and friend without whom this work would have never been possible.

We thank Rachel Wu, Jeremy Wohlwend, Felix Faltings, Jason Yim, Victor Quach, Saro Passaro, Patrick Walters, Michael Heinzinger, Mario Geiger, Michael John Arcidiacono, Noah Getz, and John Yang for valuable feedback and insightful discussions. We thank Wei Lu for his help with running TANKBind. This work was supported by the Machine Learning for Pharmaceutical Discovery and Synthesis (MLPDS) consortium, the Abdul Latif Jameel Clinic for Machine Learning in Health, the DTRA Discovery of Medical Countermeasures Against New and Emerging (DOMANE) threats program, the DARPA Accelerated Molecular Discovery program and the Sanofi Computational Antibody Design grant. Bowen Jing acknowledges the support from the Department of Energy Computational Science Graduate Fellowship.

References

[1] Oleg Trott and Arthur J Olson. Autodock vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. *Journal of computational chemistry*, 31(2):455–461, 2010.

[2] Thomas A Halgren, Robert B Murphy, Richard A Friesner, Hege S Beard, Leah L Frye, W Thomas Pollard, and Jay L Banks. Glide: a new approach for rapid, accurate docking and scoring. 2. enrichment factors in database screening. *Journal of medicinal chemistry*, 2004.

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

[4] Emanuele Rodolà, Zorah Lähner, Alexander M Bronstein, Michael M Bronstein, and Justin Solomon. Functional maps representation on product manifolds. In *Computer Graphics Forum*, volume 38, pages 678–689. Wiley Online Library, 2019.

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

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

[7] Zhihai Liu, Minyi Su, Li Han, Jie Liu, Qifan Yang, Yan Li, and Renxiao Wang. Forging the basis for developing protein–ligand interaction scoring functions. *Accounts of Chemical Research*, 50(2):302–309, 2017.

[8] H. Berman, K. Henrick, and H. Nakamura. Announcing the worldwide Protein Data Bank. *Nat Struct Biol*, 10(12):980, Dec 2003.

[9] Hannes Stärk, Octavian Ganea, Lagnajit Pattanaik, Regina Barzilay, and Tommi Jaakkola. Equibind: Geometric deep learning for drug binding structure prediction. In *International Conference on Machine Learning*, pages 20503–20521. PMLR, 2022.

[10] Mikhail Volkov, Joseph-André Turk, Nicolas Drizard, Nicolas Martin, Brice Hoffmann, Yann Gaston-Mathé, and Didier Rognan. On the frustration to predict binding affinities from protein–ligand structures with deep neural networks. *Journal of Medicinal Chemistry*, 65(11):7946–7958, Jun 2022.

[11] Wei Lu, Qifeng Wu, Jixian Zhang, Jiahua Rao, Chengtao Li, and Shuangjia Zheng. Tankbind: Trigonometry-aware neural networks for drug-protein binding structure prediction. *Advances in neural information processing systems*, 2022.

[12] Nafisa M. Hassan, Amr A. Alhossary, Yuguang Mu, and Chee-Keong Kwoh. Protein-ligand blind docking using quickvina-w with inter-process spatio-temporal integration. *Scientific Reports*, 7(1):15451, Nov 2017.

[13] Andrew T McNutt, Paul Francoeur, Rishal Aggarwal, Tomohide Masuda, Rocco Meli, Matthew Ragoza, Jocelyn Sunseri, and David Ryan Koes. Gnina 1.0: molecular docking with deep learning. *Journal of cheminformatics*, 13(1):1–20, 2021.

[14] David Ryan Koes, Matthew P Baumgartner, and Carlos J Camacho. Lessons learned in empirical scoring with smina from the csar 2011 benchmarking exercise. *Journal of chemical information and modeling*, 53(8):1893–1904, 2013.

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

[16] Nathaniel Thomas, Tess Smidt, Steven Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley. Tensor field networks: Rotation-and translation-equivariant neural networks for 3d point clouds. *arXiv preprint*, 2018.

[17] Mario Geiger, Tess Smidt, Alby M., Benjamin Kurt Miller, Wouter Boomsma, Bradley Dice, Kostiantyn Lapchevskyi, Maurice Weiler, Michał Tyszkiewicz, Simon Batzner, Martin Uhrin, Jes Frellsen, Nuri Jung, Sophia Sanborn, Josh Rackers, and Michael Bailey. Euclidean neural networks: e3nn, 2020.

[18] Zeming Lin, Halil Akin, Roshan Rao, Brian Hie, Zhongkai Zhu, Wenting Lu, Allan dos Santos Costa, Maryam Fazel-Zarandi, Tom Sercu, Sal Candido, and Alexander Rives. Language models of protein sequences at the scale of evolution enable accurate structure prediction. *arXiv*, 2022.

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

[20] Kristof Schütt, Pieter-Jan Kindermans, Huziel Enoc Sauceda Felix, Stefan Chmiela, Alexandre Tkatchenko, and Klaus-Robert Müller. Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. *Advances in neural information processing systems*, 2017.

[21] Mario Geiger and Tess Smidt. e3nn: Euclidean neural networks. *arXiv preprint arXiv:2207.09453*, 2022.

[22] Rocco Meli and Philip C. Biggin. spyrmsd: symmetry-corrected rmsd calculations in python. *Journal of Cheminformatics*, 12(1):49, 2020.

[23] Amr Alhossary, Stephanus Daniel Handoko, Yuguang Mu, and Chee-Keong Kwoh. Fast, accurate, and reliable molecular docking with QuickVina 2. *Bioinformatics*, 31(13):2214–2216, 02 2015.

[24] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[25] Radoslav Krivák and David Hoksza. P2rank: machine learning based tool for rapid and accurate prediction of ligand binding sites from protein structure. *Journal of cheminformatics*, 10(1):1–12, 2018.

[26] Srinivas Ramachandran, Pradeep Kota, Feng Ding, and Nikolay V Dokholyan. Automated minimization of steric clashes in protein structures. *Proteins*, 79(1):261–270, January 2011.

[27] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33:13260–13271, 2020.

[28] Lifan Chen, Xiaoqin Tan, Dingyan Wang, Feisheng Zhong, Xiaohong Liu, Tianbiao Yang, Xiaomin Luo, Kaixian Chen, Hualiang Jiang, and Mingyue Zheng. Transformer cpi: improving compound–protein interaction prediction by sequence-based deep learning with self-attention mechanism and label reversal experiments. *Bioinformatics*, 36(16):4406–4414, 2020.

[29] Shuya Li, Fangping Wan, Hantao Shu, Tao Jiang, Dan Zhao, and Jianyang Zeng. Monn: a multi-objective neural network for predicting compound-protein interactions and affinities. *Cell Systems*, 10(4):308–322, 2020.

[30] Dejun Jiang, Chang-Yu Hsieh, Zhenxing Wu, Yu Kang, Jike Wang, Ercheng Wang, Ben Liao, Chao Shen, Lei Xu, Jian Wu, et al. Interactiongraphnet: A novel and efficient deep graph representation learning framework for accurate protein–ligand interaction predictions. *Journal of medicinal chemistry*, 64(24):18209–18232, 2021.

[31] Seokhyun Moon, Wonho Zhung, Soojung Yang, Jaechang Lim, and Woo Youn Kim. Pignet: a physics-informed deep learning model toward generalized drug–target interaction predictions. *Chemical Science*, 13(13):3661–3673, 2022.

[32] Vignesh Ram Somnath, Charlotte Bunne, and Andreas Krause. Multi-scale representation learning on proteins. *Advances in Neural Information Processing Systems*, 34:25244–25255, 2021.

[33] Penglei Wang, Shuangjia Zheng, Yize Jiang, Chengtao Li, Junhong Liu, Chang Wen, Atanas Patronov, Dahong Qian, Hongming Chen, and Yuedong Yang. Structure-aware multimodal deep learning for drug–protein interaction prediction. *Journal of chemical information and modeling*, 62(5):1308–1317, 2022.

A Proofs

A.1 Proof of Proposition 1

Proposition 1. *Let* $\mathbf{x}(t) := A_{tor}(t\boldsymbol{\theta}, \mathbf{x})$ *for some* $\boldsymbol{\theta}$ *and where* $t\boldsymbol{\theta} = (t\theta_1, \dots t\theta_m)$*. Then the linear and angular momentum are zero:* $\frac{d}{dt}\bar{\mathbf{x}}|_{t=0} = 0$ *and* $\sum_i (\mathbf{x}_i - \bar{\mathbf{x}}) \times \frac{d}{dt}\mathbf{x}_i|_{t=0} = 0$ *where* $\bar{\mathbf{x}} = \frac{1}{n}\sum_i \mathbf{x}$.

Proof. Let $\mathbf{x}(t) = R(t)(B(t\boldsymbol{\theta}, \mathbf{x}) - \bar{\mathbf{x}}_0) + \bar{\mathbf{x}}_0 + \mathbf{p}(t)$ where $B(t\boldsymbol{\theta}, \cdot) = B_{1,t\theta_1} \circ \cdots B_{m,t\theta_m}$ and $R(t), \mathbf{p}(t)$ are the rotation (around $\bar{\mathbf{x}}_0 := \bar{\mathbf{x}}(0)$) and translation associated with the optimal alignment between $B(t\boldsymbol{\theta}, \mathbf{x})$ and $\mathbf{x}$. By definition of RMSD, we have

$$||\mathbf{x}(t) - \mathbf{x}(0)|| = \min_{R,\mathbf{p}} ||R(t)(B(t\boldsymbol{\theta}, \mathbf{x}(0)) - \bar{\mathbf{x}}_0) + \bar{\mathbf{x}}_0 + \mathbf{p}(t) - \mathbf{x}(0)|| \tag{1}$$

which, in the limit of $t \to 0$, becomes

$$\left|\left|\frac{d}{dt}\mathbf{x}(t)\right|\right|^2_{t=0} = \min_{R,\mathbf{p}} \left|\left|\frac{d}{dt}\left(R(t)(B(t\boldsymbol{\theta}, \mathbf{x}(0)) - \bar{\mathbf{x}}_0) + \mathbf{p}(t)\right)\right|\right|^2_{t=0} \tag{2}$$

The derivative in the LHS of Equation 2 at $t = 0$ is

$$R'(t)(\mathbf{x} - \bar{\mathbf{x}}) + B'(t\boldsymbol{\theta}, \mathbf{x}(0)) + \mathbf{p}'(t) \tag{3}$$

and represents the instantaneous velocity of the points $\mathbf{x}_i$ at $t = 0$. Denoting $\mathbf{r}_i = \mathbf{x}_i - \bar{\mathbf{x}}_0$, our objective is to minimize $\sum_i ||\mathbf{r}'_i||^2$. Then, if we describe $R'(t)(\mathbf{x} - \bar{\mathbf{x}}_0) = R'(t)\mathbf{r}$ by an angular velocity $\boldsymbol{\omega}$ and abbreviate $B'(t\boldsymbol{\theta}, \mathbf{x}(0))_i := \mathbf{b}_i$ and $\mathbf{p}' = \mathbf{v}$, we have

$$\begin{aligned}
\sum_i ||\mathbf{r}_i'||^2 &= \sum_i (\mathbf{b}_i + \boldsymbol{\omega} \times \mathbf{r}_i + \mathbf{v}) \cdot (\mathbf{b}_i + \boldsymbol{\omega} \times \mathbf{r}_i + \mathbf{v}) \\
&= \sum_i \left[||\mathbf{b}_i||^2 + 2\mathbf{b}_i \cdot (\boldsymbol{\omega} \times \mathbf{r}_i) + 2\mathbf{b}_i \cdot \mathbf{v} + (\boldsymbol{\omega} \times \mathbf{r}_i) \cdot (\boldsymbol{\omega} \times \mathbf{r}_i) + 2(\boldsymbol{\omega} \times \mathbf{r}_i) \cdot \mathbf{v} + ||\mathbf{v}||^2\right] \\
&= \sum_i ||\mathbf{b}_i||^2 + 2\boldsymbol{\omega} \cdot \sum_i (\mathbf{r}_i \times \mathbf{b}_i) + 2\left(\sum_i \mathbf{b}_i\right) \cdot \mathbf{v} + n\,||\mathbf{v}||^2 + \boldsymbol{\omega}^T \mathcal{I}(\mathbf{r}) \boldsymbol{\omega}
\end{aligned} \tag{4}$$

where we have used the fact that $\sum_i \mathbf{r}_i = 0$ and where $\mathcal{I}(r) = (\sum_i \mathbf{r}_i \cdot \mathbf{r}_i)\, I - \sum_i \mathbf{r}_i \mathbf{r}_i^T$ is the 3×3 *inertia tensor*. Then setting gradients with respect to $\mathbf{v}, \boldsymbol{\omega}$ gives

$$\mathbf{v} = -\frac{1}{n} \sum_i \mathbf{b}_i \quad \text{and} \quad \boldsymbol{\omega} = -\mathcal{I}(\mathbf{r})^{-1} \left(\sum_i \mathbf{r}_i \times \mathbf{b}_i\right) \tag{5}$$

Now with $\mathbf{r}_i' = \mathbf{b}_i + \boldsymbol{\omega} \times \mathbf{r}_i + \mathbf{v}$ we evaluate the linear momentum

$$\frac{1}{n} \sum_i \mathbf{r}_i' = \frac{1}{n} \left(\sum_i \mathbf{b}_i + \boldsymbol{\omega} \times \sum_i \mathbf{r}_i + n\mathbf{v}\right) = 0 \tag{6}$$

which is zero by direct substitution of $\mathbf{v}$. Similarly, we evaluate the angular momentum

$$\begin{aligned}
\sum_i \mathbf{r}_i \times \mathbf{r}_i' &= \sum_i \mathbf{r}_i \times \mathbf{b}_i + \sum_i \mathbf{r}_i \times (\boldsymbol{\omega} \times \mathbf{r}_i) + \sum_i \mathbf{r}_i \times \mathbf{v} \\
&= \sum_i \mathbf{r}_i \times \mathbf{b}_i + \mathcal{I}(\mathbf{r}) \boldsymbol{\omega} = 0
\end{aligned} \tag{7}$$

which is zero by direct substitution of $\boldsymbol{\omega}$. Thus, the linear and angular momentum are zero at $t = 0$ for arbitrary $\mathbf{x}(0)$. $\square$

Note that since we did not use the particular form of $B(t\boldsymbol{\theta}, \mathbf{x})$ in the above proof, we have shown that RMSD alignment can be used to disentangle rotations and translations from the infinitesimal action of any arbitrary function.

A.2 Proof of Proposition 2

Proposition 2. *For a given seed conformation* $\mathbf{c}$*, the map* $A(\cdot, \mathbf{c}) : \mathbb{P} \to \mathcal{M}_\mathbf{c}$ *is a bijection.*

Proof. Since we defined $\mathcal{M}_\mathbf{c} = \{A(g, \mathbf{c}) \mid g \in \mathbb{P}\}$, $A(\cdot, \mathbf{c})$ is automatically surjective. We now show that it is injective. Assume for the sake of contradiction that $A(\cdot, \mathbf{c})$ is not injective, so that there exist elements of the product space $g_1, g_2 \in \mathbb{P}$ with $g_1 \neq g_2$ but with $A(g_1, \mathbf{c}) = A(g_2, \mathbf{c}) = \mathbf{c}'$. That is,

$$A_{\text{tr}}(\mathbf{r}_1, A_{\text{rot}}(R_1, A_{\text{tor}}(\boldsymbol{\theta}_1, \mathbf{c}))) = A_{\text{tr}}(\mathbf{r}_2, A_{\text{rot}}(R_2, A_{\text{tor}}(\boldsymbol{\theta}_2, \mathbf{c}))) \tag{8}$$

which we abbreviate as $\mathbf{c}^{(1)} = \mathbf{c}^{(2)}$. Since only A_{tr} changes the center of mass $\sum_i \mathbf{c}_i/n$, we have $\sum_i \mathbf{c}_i^{(1)}/n = \sum_i \mathbf{c}_i/n + \mathbf{r}_1$ and $\sum_i \mathbf{c}_i^{(2)}/n = \sum_i \mathbf{c}_i/n + \mathbf{r}_2$. However, since $\mathbf{c}^{(1)} = \mathbf{c}^{(2)}$, this implies $\mathbf{r}_1 = \mathbf{r}_2$. Next, consider the torsion angles $\boldsymbol{\tau}_1 = (\tau_1^{(1)}, \ldots \tau_m^{(1)})$ of $\mathbf{c}^{(1)}$ corresponding to some choice of dihedral angles at each rotatable bond. Because A_{tr} and A_{rot} are rigid-body motions, only A_{tor} changes the dihedral angles; in particular, by definition we have $\tau_i^{(1)} \cong \tau_i + \theta_i^{(1)} \mod 2\pi$ and $\tau_i^{(2)} \cong \tau_i + \theta_i^{(2)} \mod 2\pi$ for all $i = 1, \ldots m$. However, because $\tau_i^{(1)} = \tau_i^{(2)}$, this means $\theta_i^{(1)} \cong \theta_i^{(2)}$ for all i and therefore $\boldsymbol{\theta}_1 = \boldsymbol{\theta}_2$ (as elements of $SO(2)^m$). Now denote $\mathbf{c}^\star = A_{\text{tor}}(\boldsymbol{\theta}_1, \mathbf{c}) = A_{\text{tor}}(\boldsymbol{\theta}_2, \mathbf{c})$ and apply $A_{\text{tr}}(-\mathbf{r}_1, \cdot) = A_{\text{tr}}(-\mathbf{r}_2, \cdot)$ to both sides of Equation 8. We then have

$$A_{\text{rot}}(R_1, \mathbf{c}^\star) = A_{\text{rot}}(R_2, \mathbf{c}^\star) \tag{9}$$

which further leads to

$$\mathbf{c}^\star - \bar{\mathbf{c}}^\star = R_1^{-1} R_2 (\mathbf{c}^\star - \bar{\mathbf{c}}^\star) \tag{10}$$

In general, this does not imply that $R_1 = R_2$. However, $R_1 \neq R_2$ is possible only if $\mathbf{c}^\star$ is *degenerate*, in the sense that all points are collinear along the shared axis of rotation of R_1, R_2. However, in practice, conformers never consist of a collinear set of points, so we can safely assume $R_1 = R_2$. We now have $(\mathbf{r}_1, R_1, \boldsymbol{\theta}_1) = (\mathbf{r}_2, R_2, \boldsymbol{\theta}_2)$, or $g_1 = g_2$, contradicting our initial assumption. We thus conclude that $A(\cdot, \mathbf{c})$ is injective, completing the proof. $\square$

B Training and Inference

In this section we present the training and inference procedures of the diffusion generative model. First, however, there are a few subtleties of the generative approach to molecular docking that are worth mentioning. Unlike the standard generative modeling setting where the dataset consists of many samples drawn from the data distribution, each training example $(\mathbf{x}^\star, \mathbf{y})$ of protein structure $\mathbf{y}$ and ground-truth ligand pose $\mathbf{x}^\star$ is the *only sample* from the corresponding conditional distribution $p_{\mathbf{x}^\star}(\cdot \mid \mathbf{y})$ defined over $\mathcal{M}_{\mathbf{x}^\star}$. Thus, the innermost training loop iterates over distinct conditional distributions $p_{\mathbf{x}^\star}(\cdot \mid \mathbf{y})$, along with a single sample from that distribution, rather than over samples from a common data distribution $p_{\text{data}}(\mathbf{x})$.

As discussed in Section 2, during inference, $\mathbf{c}$ is the ligand structure generated with a method such as RDKit. However, during training we require $\mathcal{M}_{\mathbf{c}} = \mathcal{M}_{\mathbf{x}^\star}$ in order to define a bijection between $\mathbf{c} \in \mathcal{M}_{\mathbf{x}^\star}$ and $\mathbb{P}$. If we take $\mathbf{c} \in \mathcal{M}_{\mathbf{x}^\star}$, there will be a distribution shift between the manifolds $\mathcal{M}_{\mathbf{c}}$ considered at training time and those considered at inference time. To circumvent this issue, at training time we predict $\mathbf{c}$ with RDKit and replace $\mathbf{x}^\star$ with $\arg\min_{\mathbf{x}^\dagger \in \mathcal{M}_{\mathbf{c}}} \text{RMSD}(\mathbf{x}^\star, \mathbf{x}^\dagger)$ using the conformer matching procedure described in [15].

The above paragraph may be rephrased more intuitively as follows: during inference, the generative model docks a ligand structure generated by RDKit, keeping its non-torsional degrees of freedom (e.g., local structures) fixed. At training time, however, if we train the score model with the local structures of the ground truth pose, this will not correspond to the local structures seen at inference time. Thus, at training time, we replace the ground truth pose by generating a ligand structure with RDKit and aligning it to the ground truth pose while keeping the local structures fixed.

With these preliminaries, we now continue to the full procedures (Algorithms 1 and 2). The training and inference procedures of a score-based diffusion generative model on a Riemannian manifold consist of (1) sampling and regressing against the score of the diffusion kernel during training; and (2) sampling a geodesic random walk with the score as a drift term during inference [3]. Because we have developed the diffusion process on $\mathbb{P}$ but continue to provide the score model with elements in $\mathcal{M}_{\mathbf{c}} \subset \mathbb{R}^{3n}$, the full training and inference procedures involve repeatedly interconverting between the two spaces using the bijection given by the seed conformation $\mathbf{c}$.

Algorithm 1: Training procedure (single epoch)

```
Input: Training pairs $\{(\mathbf{x}^\star, \mathbf{y})\}$, RDKit predictions $\{\mathbf{c}\}$
foreach $\mathbf{c}, \mathbf{x}^\star, \mathbf{y}$ do
    Let $\mathbf{x}_0 \leftarrow \arg\min_{\mathbf{x}^\dagger \in \mathcal{M}_{\mathbf{c}}} \text{RMSD}(\mathbf{x}^\star, \mathbf{x}^\dagger)$;
    Compute $(\mathbf{r}_0, R_0, \boldsymbol{\theta}_0) \leftarrow A_{\mathbf{c}}^{-1}(\mathbf{x}_0)$;
    Sample $t \sim \text{Uni}([0,1])$;
    Sample $\Delta\mathbf{r}, \Delta R, \Delta\boldsymbol{\theta}$ from diffusion kernels $p_t^{\text{tr}}(\cdot \mid 0), p_t^{\text{rot}}(\cdot \mid 0), p_t^{\text{tor}}(\cdot \mid 0)$;
    Set $\mathbf{r}_t \leftarrow \mathbf{r}_0 + \Delta\mathbf{r}$;
    Set $R_t \leftarrow (\Delta R) R_0$;
    Set $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_0 + \Delta\boldsymbol{\theta} \mod 2\pi$;
    Compute $\mathbf{x}_t \leftarrow A((\mathbf{r}_t, R_t, \boldsymbol{\theta}_t), \mathbf{c})$;
    Predict scores $\alpha \in \mathbb{R}^3, \beta \leftarrow \mathbb{R}^3, \gamma \in \mathbb{R}^m = \mathbf{s}(\mathbf{x}_t, \mathbf{c}, \mathbf{y}, t)$ ;
    Take optimization step on loss
      $\mathcal{L} = ||\alpha - \nabla p_t^{\text{tr}}(\Delta\mathbf{r} \mid 0)||^2 + ||\beta - \nabla p_t^{\text{rot}}(\Delta R \mid 0)||^2 + ||\gamma - \nabla p_t^{\text{tor}}(\Delta\boldsymbol{\theta} \mid 0)||^2$
```

However, as noted in the main text, the dependence of these procedures on the exact choice of $\mathbf{c}$ is potentially problematic, as it suggests that at inference time, the model distribution may be different depending on the orientation and torsion angles of $\mathbf{c}$. Simply removing the dependence of the score model on $\mathbf{c}$ is not sufficient since the update steps themselves still occur on $\mathbb{P}$ and require a choice of $\mathbf{c}$ to be mapped to $\mathcal{M}_{\mathbf{c}}$. However, notice that the update steps—in both training and inference—consist of (1) sampling the diffusion kernels at the origin; (2) applying these updates to the point on $\mathbb{P}$; and (3) transferring the point on $\mathbb{P}$ to $\mathcal{M}_{\mathbf{c}}$ via $A(\cdot, \mathbf{c})$. Might it instead be possible to apply the updates to 3D ligand poses $\mathbf{x} \in \mathcal{M}_{\mathbf{c}}$ *directly*?

It turns out that the notion of applying these steps to ligand poses "directly" corresponds to the formal notion of *group action*. The operations $A_{\text{tr}}, A_{\text{rot}}, A_{\text{tor}}$ that we have already defined are formally

Algorithm 2: Inference procedure

```
Input: RDKit prediction $\mathbf{c}$, protein structure $\mathbf{y}$ (both centered at origin)
Output: Sampled ligand pose $\mathbf{x}_0$
Sample $\boldsymbol{\theta}_N \sim \text{Uni}(SO(2)^m)$, $R_N \sim \text{Uni}(SO(3))$, $\mathbf{r}_N \sim \mathcal{N}(0, \sigma^2_{\text{tor}}(T))$;
Let $\mathbf{x}_N = A((\mathbf{r}_N, R_N, \boldsymbol{\theta}_N), \mathbf{c})$;
for $n \leftarrow N$ to $1$ do
    Let $t = n/N$ and $\Delta\sigma^2_{\text{tr}} = \sigma^2_{\text{tr}}(n/N) - \sigma^2_{\text{tr}}((n-1)/N)$ and similarly for $\Delta\sigma^2_{\text{rot}}, \Delta\sigma^2_{\text{tor}}$;
    Predict scores $\alpha \in \mathbb{R}^3, \beta \in \mathbb{R}^3, \gamma \in \mathbb{R}^m \leftarrow \mathbf{s}(\mathbf{x}_n, \mathbf{c}, \mathbf{y}, t)$;
    Sample $\mathbf{z}_{\text{tr}}, \mathbf{z}_{\text{rot}}, \mathbf{z}_{\text{tor}}$ from $\mathcal{N}(0, \Delta\sigma^2_{\text{tr}}), \mathcal{N}(0, \Delta\sigma^2_{\text{rot}}), \mathcal{N}(0, \Delta\sigma^2_{\text{tor}})$ respectively;
    Set $\mathbf{r}_{n-1} \leftarrow \mathbf{r}_0 + \Delta\sigma^2_{\text{tr}}\alpha + \mathbf{z}_{\text{tr}}$;
    Set $R_{n-1} \leftarrow \mathbf{R}(\Delta\sigma^2_{\text{rot}}\beta + \mathbf{z}_{\text{rot}})R_n)$;
    Set $\boldsymbol{\theta}_{n-1} \leftarrow \boldsymbol{\theta}_n + (\Delta\sigma^2_{\text{tor}}\gamma + \mathbf{z}_{\text{tor}}) \mod 2\pi$;
    Compute $\mathbf{x}_{n-1} \leftarrow A((\mathbf{r}_{n-1}, R_{n-1}, \boldsymbol{\theta}_{n-1}), \mathbf{c})$;
Return $\mathbf{x}_0$;
```

group actions if they satisfy $A_{(\cdot)}(g_1 g_2, \mathbf{x}) = A(g_1, A(g_2, \mathbf{x}))$. While true for $A_{\text{tr}}, A_{\text{rot}}$, this is not generally true for A_{tor} if we take $SO(2)^m$ to be the direct product group; however, the approximation is increasingly good as the magnitude of the torsion angle updates decreases. If we then define $\mathbb{P}$ to be the direct product group of its constituent groups, A is a group action of $\mathbb{P}$ on $\mathcal{M}_{\mathbf{c}}$, as the operations of $A_{\text{tr}}, A_{\text{rot}}, A_{\text{tor}}$ commute and are (under the approximation) individually group actions.

The implication of A being a group action can be seen as follows. Let $\delta = g_b g_a^{-1}$ be the update which brings $g_a \in \mathbb{P}$ to $g_b \in \mathbb{P}$ via left multiplication, and let $\mathbf{x}_a, \mathbf{x}_b$ be the corresponding ligand poses $A(g_a, \mathbf{c}), A(g_b, \mathbf{c})$. Then

$$\mathbf{x}_b = A(g_b g_a^{-1} g_a, \mathbf{c}) = A(\delta, \mathbf{x}_a) \tag{11}$$

which means that the updates δ can be applied directly to $\mathbf{x}_a$ using the operation A. The training and inference procedures then become Algorithm 3 and 4 below. The initial conformer $\mathbf{c}$ is no longer used, except in the initial steps to define the manifold—to find the closest point to $\mathbf{x}^\star$ in training, and to sample $\mathbf{x}_N$ from the prior over $\mathcal{M}_{\mathbf{c}}$ in inference.

Conceptually speaking, this procedure corresponds to "forgetting" the location of the origin element on $\mathcal{M}_{\mathbf{c}}$, which is permissible because a change of the origin to some equivalent seed $\mathbf{c}' \in \mathcal{M}_{\mathbf{c}}$ merely translates—via right multiplication by $A_{\mathbf{c}}^{-1}(\mathbf{c}')$—the original and diffused data distributions on $\mathbb{P}$, but does not cause any changes on $\mathcal{M}_{\mathbf{c}}$ itself. The training and inference routines involve updates—formally left multiplications—to group elements, but as left multiplication on the group corresponds to group actions on $\mathcal{M}_{\mathbf{c}}$, the updates can act on $\mathcal{M}_{\mathbf{c}}$ directly, without referencing the origin $\mathbf{c}$.

We find that the approximation of A as a group action works quite well in practice and use Algorithms 3 and 4 for all training and experiments discussed in the paper. Of course, disentangling the torsion updates from rotations in a way that makes A_{tor} exactly a group action would justify the procedure further, and we regard this as a possible direction for future work.

Algorithm 3: Approximate training procedure (single epoch)

```
Input: Training pairs $\{(\mathbf{x}^\star, \mathbf{y})\}$, RDKit predictions $\{\mathbf{c}\}$
foreach $\mathbf{c}, \mathbf{x}^\star, \mathbf{y}$ do
    Let $\mathbf{x}_0 \leftarrow \arg\min_{\mathbf{x}^\dagger \in \mathcal{M}_{\mathbf{c}}} \text{RMSD}(\mathbf{x}^\star, \mathbf{x}^\dagger)$;
    Sample $t \sim \text{Uni}([0,1])$;
    Sample $\Delta\mathbf{r}, \Delta R, \Delta\boldsymbol{\theta}$ from diffusion kernels $p_t^{\text{tr}}(\cdot \mid 0), p_t^{\text{rot}}(\cdot \mid 0), p_t^{\text{tor}}(\cdot \mid 0)$;
    Compute $\mathbf{x}_t \leftarrow A((\Delta\mathbf{r}, \Delta R, \Delta\boldsymbol{\theta}), \mathbf{x}_0)$;
    Predict scores $\alpha \in \mathbb{R}^3, \beta \leftarrow \mathbb{R}^3, \gamma \in \mathbb{R}^m = \mathbf{s}(\mathbf{x}_t, \mathbf{y}, t)$ ;
    Take optimization step on loss
      $\mathcal{L} = ||\alpha - \nabla p_t^{\text{tr}}(\Delta\mathbf{r} \mid 0)||^2 + ||\beta - \nabla p_t^{\text{rot}}(\Delta R \mid 0)||^2 + ||\gamma - \nabla p_t^{\text{tor}}(\Delta\boldsymbol{\theta} \mid 0)||^2$
```

Algorithm 4: Approximate inference procedure

```
Input: RDKit prediction $\mathbf{c}$, protein structure $\mathbf{y}$ (both centered at origin)
Output: Sampled ligand pose $\mathbf{x}_0$
Sample $\boldsymbol{\theta}_N \sim \text{Uni}(SO(2)^m)$, $R_N \sim \text{Uni}(SO(3))$, $\mathbf{r}_N \sim \mathcal{N}(0, \sigma^2_{\text{tor}}(T))$;
Let $\mathbf{x}_N = A((\mathbf{r}_N, R_N, \boldsymbol{\theta}_N), \mathbf{c})$;
for $n \leftarrow N$ to $1$ do
    Let $t = n/N$ and $\Delta\sigma^2_{\text{tr}} = \sigma^2_{\text{tr}}(n/N) - \sigma^2_{\text{tr}}((n-1)/N)$ and similarly for $\Delta\sigma^2_{\text{rot}}, \Delta\sigma^2_{\text{tor}}$;
    Predict scores $\alpha \in \mathbb{R}^3, \beta \in \mathbb{R}^3, \gamma \in \mathbb{R}^m \leftarrow \mathbf{s}(\mathbf{x}_n, \mathbf{y}, t)$;
    Sample $\mathbf{z}_{\text{tr}}, \mathbf{z}_{\text{rot}}, \mathbf{z}_{\text{tor}}$ from $\mathcal{N}(0, \Delta\sigma^2_{\text{tr}}), \mathcal{N}(0, \Delta\sigma^2_{\text{rot}}), \mathcal{N}(0, \Delta\sigma^2_{\text{tor}})$ respectively;
    Set $\Delta\mathbf{r} \leftarrow \mathbf{r}_0 + \Delta\sigma^2_{\text{tr}}\alpha + \mathbf{z}_{\text{tr}}$;
    Set $\Delta R \leftarrow \mathbf{R}(\Delta\sigma^2_{\text{rot}}\beta + \mathbf{z}_{\text{rot}})$;
    Set $\Delta\boldsymbol{\theta} \leftarrow \Delta\sigma^2_{\text{tor}}\gamma + \mathbf{z}_{\text{tor}}$;
    Compute $\mathbf{x}_{n-1} \leftarrow A((\Delta\mathbf{r}, \Delta R, \Delta\boldsymbol{\theta}), \mathbf{x}_n)$;
Return $\mathbf{x}_0$;
```

C Architecture Details

We use convolutional networks based on tensor products of irreducible representations (irreps) of SO(3) [16] as architecture for both the score and confidence models. In particular, these are implemented using the `e3nn` library [17]. Below, $\otimes_w$ refers to the spherical tensor product of irreps with path weights w, and $\oplus$ refers to normal vector addition (with possibly padded inputs). Features have multiple channels for each irrep. Both the architectures can be decomposed into three main parts: embedding layer, interaction layers, and output layer. We outline each of them below.

C.1 Embedding layer

Geometric heterogeneous graph. Structures are represented as heterogeneous geometric graphs with nodes representing ligand (heavy) atoms, receptor residues (located in the position of the α-carbon atom), and receptor (heavy) atoms (only for the confidence model). Because of the high number of nodes involved, it is necessary for the graph to be sparsely connected for runtime and memory constraints. Moreover, sparsity can act as a useful inductive bias for the model, however, it is critical for the model to find the right pose that nodes that might have a strong interaction in the final pose to be connected during the diffusion process. Therefore, to build the radius graph, we connect nodes using cutoffs that are dependent on the types of nodes they are connecting:

1. Ligand atoms-ligand atoms, receptor atoms-receptor atoms, and ligand atoms-receptor atoms interactions all use a cutoff of 5Å, standard practice for atomic interactions. For the ligand atoms-ligand atoms interactions we also preserve the covalent bonds as separate edges with some initial embedding representing the bond type (single, double, triple and aromatic). For receptor atoms-receptor atoms interactions, we limit at 8 the maximum number of neighbors of each atom. Note that the ligand atoms-receptor atoms only appear in the confidence model where the final structure is already set.
2. Receptor residues-receptor residues use a cutoff of 15 Å with 24 as the maximum number of neighbors for each residue.
3. Receptor residues-ligand atoms use a cutoff of $20 + 3 * \sigma_{tr}$ Å where σ_{tr} represents the current standard deviation of the diffusion translational noise present in each dimension (zero for the confidence model). Intuitively this guarantees that with high probability, any of the ligands and receptors that will be interacting in the final pose the diffusion model converges to are connected in the message passing at every step.
4. Finally, receptor residues are connected to the receptor atoms that form the corresponding amino-acid.

Node and edge featurization. For the receptor residues, we use the residue type as a feature as well as a language model embedding obtained from ESM2 [18]. The ligand atoms have the following features: atomic number; chirality; degree; formal charge; implicit valence; the number of connected hydrogens; the number of radical electrons; hybridization type; whether or not it is in an aromatic ring; in how many rings it is; and finally, 6 features for whether or not it is in a ring of size 3, 4, 5, 6, 7, or 8. These are concatenated with sinusoidal embeddings of the diffusion time [19] and, in the case of edges, radial basis embeddings of edge length [20]. These scalar features of each node and edge are then transformed with learnable two-layer MLPs (different for each node and edge type) into a set of scalar features that are used as initial representations by the interaction layers.

Notation Let $(\mathcal{V}, \mathcal{E})$ represent the heterogeneous graph, with $\mathcal{V} = (\mathcal{V}_\ell, \mathcal{V}_r)$ respectively ligand atoms and receptor residues (receptor atoms $\mathcal{V}_a$, present in the confidence model, are for simplicity not included here), and similarly $\mathcal{E} = (\mathcal{E}_{\ell\ell}, \mathcal{E}_{\ell r}, \mathcal{E}_{r\ell}, \mathcal{E}_{rr})$. Let $\mathbf{h}_a$ be the node embeddings (initially only scalar channels) of node a, e_{ab} the edge embeddings of (a, b), and $\mu(r_{ab})$ radial basis embeddings of the edge length. Let σ^2_{tr}, σ^2_{rot}, and σ^2_{tor} represent the variance of the diffusion kernel in each of the three components: translational, rotational and torsional.

C.2 Interaction layers

At each layer, for every pair of nodes in the graph, we construct messages using tensor products of the current node features with the spherical harmonic representations of the edge vector. The weights of this tensor product are computed based on the edge embeddings and the *scalar* features—denoted $\mathbf{h}^0_a$—of the outgoing and incoming nodes. The messages are then aggregated at each node and used to update the current node features. For every node a of type t_a:

$$\mathbf{h}_a \leftarrow \mathbf{h}_a \underset{t\in\{\ell,r\}}{\oplus} \mathrm{BN}^{(t_a,t)}\left(\frac{1}{|\mathcal{N}^{(t)}_a|}\sum_{b\in\mathcal{N}^{(t)}_a} Y(\hat{r}_{ab}) \otimes_{\psi_{ab}} \mathbf{h}_b\right) \quad \text{with } \psi_{ab} = \Psi^{(t_a,t)}(e_{ab}, \mathbf{h}^0_a, \mathbf{h}^0_b) \tag{12}$$

Here, t indicates an arbitrary node type, $\mathcal{N}^{(t)}_a = \{b \mid (a, b) \in \mathcal{E}_{t_a t}\}$ the neighbors of a of type t, Y are the spherical harmonics up to $\ell = 2$, and BN the (equivariant) batch normalisation. The orders of the output are restricted to a maximum of $\ell = 1$. All learnable weights are contained in Ψ, a dictionary of MLPs, which uses different sets of weights for different edge types (as an ordered pair so four types for the score model and nine for the confidence) and different rotational orders.

C.3 Output layer

The ligand atom representations after the final interaction layer are used in the output layer to produce the required outputs. This is where the score and confidence architecture differ significantly. On one hand, the score model's output is in the tangent space $T_{\mathbf{r}}\mathbb{T}_3 \oplus T_R SO(3) \oplus T_{\boldsymbol{\theta}} SO(2)^m$. This corresponds to having two $SE(3)$-equivariant output vectors representing the translational and rotational score predictions and m $SE(3)$-invariant output scalars representing the torsional score. For each of these, we design final tensor-product convolutions inspired by classical mechanics. On the other hand, the confidence model outputs a single $SE(3)$-invariant scalar representing the confidence score. Below we detail how each of these outputs is generated.

Translational and rotational scores. The translational and rotational score intuitively represent, respectively, the linear acceleration of the center of mass of the ligand and the angular acceleration of the rest of the molecule around the center. Considering the ligand as a rigid object and given a set of forces and masses at each ligand, a tensor product convolution between the atoms and the center of mass would be capable of computing the desired quantities. Therefore, for each of the two outputs, we perform a convolution of each of the ligand atoms with the (unweighted) center of mass c.

$$\mathbf{v} \leftarrow \frac{1}{|\mathcal{V}_\ell|}\sum_{a\in\mathcal{V}_\ell} Y(\hat{r}_{ca}) \otimes_{\psi_{ca}} \mathbf{h}_a \quad \text{with } \psi_{ca} = \Psi(\mu(r_{ca}), \mathbf{h}^0_a) \tag{13}$$

We restrict the output of $\mathbf{v}$ to a single odd and a single even vectors (for each of the two scores). Since we are using coarse-grained representations of the protein, the score will neither be even nor odd; therefore, we sum the even and odd vector representations of $\mathbf{v}$. Finally, the magnitude (but not direction) of these vectors is adjusted with an MLP taking as input the current magnitude and the sinusoidal embeddings of the diffusion time. Finally, we (revert the normalization) by multiplying the outputs by $1/\sigma_{tr}$ for the translational score and by the expected magnitude of a score in $SO(3)$ with diffusion parameter σ_{rot} (precomputed numerically).

Torsional score. To predict the m $SE(3)$-invariant scalar describing the torsional score, we use a pseudotorque layer similar to that of [15]. This predicts a scalar score $\delta\tau$ for each rotatable bond from the per-node outputs of the atomic convolution layers. For rotatable bond $g = (g_0, g_1)$ and $b \in \mathcal{V}_\ell$, let r_{gb} and $\hat{r}_{gb}$ be the magnitude and direction of the vector connecting the center of bond g and b. We construct a convolutional filter T_g for each bond g from the tensor product of the spherical harmonics with a $\ell = 2$ representation of the bond axis $\hat{r}_g$:[2]

$$T_g(\hat{r}) := Y^2(\hat{r}_g) \otimes Y(\hat{r}) \tag{14}$$

$\otimes$ is the full (i.e., unweighted) tensor product as described in [21], and the second term contains the spherical harmonics up to $\ell = 2$ (as usual). This filter (which contains orders up to $\ell = 3$) is then used to convolve with the representations of every neighbor on a radius graph:

$$\begin{aligned}
\mathcal{E}_\tau &= \{(g, b) \mid g \text{ a rotatable bond}, b \in \mathcal{V}_\ell\} \\
e_{gb} &= \Upsilon^{(\tau)}(\mu(r_{gb})) \quad \forall (g, b) \in \mathcal{E}_\tau \\
\mathbf{h}_g &= \frac{1}{|\mathcal{N}_g|} \sum_{b \in \mathcal{N}_g} T_g(\hat{r}_{gb}) \otimes_{\gamma_{gb}} \mathbf{h}_b \\
\text{with } \gamma_{gb} &= \Gamma(e_{gb}, \mathbf{h}_b^0, \mathbf{h}_{g_0}^0 + \mathbf{h}_{g_1}^0)
\end{aligned} \tag{15}$$

Here, $\mathcal{N}_g = \{b \mid (g, b) \in \mathcal{E}_\tau\}$ and $\Upsilon^{(\tau)}$ and Γ are MLPs with learnable parameters. Since unlike [15], we use coarse-grained representations the parity also here is neither even nor odd, the irreps in the output are restricted to arrays both even $\mathbf{h}'_g$ and odd $\mathbf{h}''_g$ scalars. Finally, we produce a single scalar prediction for each bond:

$$\delta\tau_g = \Pi(\mathbf{h}'_g + \mathbf{h}''_g) \tag{16}$$

where Π is a two-layer MLP with $\tanh$ nonlinearity and no biases. This is also "denormalized" by multiplying by the expected magnitude of a score in $SO(2)$ with diffusion parameter σ_{tor}.

Confidence output. The single $SE(3)$-invariant scalar representing the confidence score output is instead obtained by concatenating the even and odd final scalar representation of each ligand atom, averaging these feature vectors among the different atoms, and finally applying a three layers MLP (with batch normalization).

D Experimental Details

In general, all our code is available at `https://anonymous.4open.science/r/DiffDock`. This includes running the baselines, runtime calculations, training and inference scripts for DiffDock, the PDB files of DiffDock's predictions for all 363 complexes of the test set, and visualization videos of the reverse diffusion.

D.1 Experimental Setup

Data. We use the molecular complexes in PDBBind [7] that were extracted from the Protein Data Bank (PDB) [8]. We employ the time-split of PDBBind proposed by [9] with 17k complexes from 2018 or earlier for training/validation and 363 test structures from 2019 with no ligand overlap with the training complexes. This is motivated by the further adoption of the same split [11] and the critical assessment of PDBBind splits by [10] who favor temporal splits over artificial splits based on molecular scaffolds or protein sequence/structure similarity. For completeness, we also report the results on protein sequence similarity splits in Appendix E.

Metrics. To evaluate the generated complexes, we compute the heavy-atom RMSD between the predicted and the crystal ligand atoms when the protein structures are aligned. To account for permutation symmetries in the ligand, we use the symmetry-corrected RMSD of sPyRMSD [22]. For these RMSD values, we report the percentage of predictions that have an RMSD that is less than

[2]Since the parity of the $\ell = 2$ spherical harmonic is even, this representation is indifferent to the choice of bond direction.

2Å. We choose 2Å since much prior work considers poses with an RMSD less that 2Å as "good" or successful [23, 12, 13]. This is a chemically relevant metric, unlike the mean RMSD since for further downstream analyses such as determining function changes, a prediction is only useful below a certain RMSD error threshold. Less relevant metrics such as the mean RMSD are provided in Appendix E.

D.2 Implementation details: hyperparameters, training, and runtime measurement

Training Details. We use Adam [24] as optimizer for the diffusion and the confidence model. The diffusion model with which we run inference uses the exponential moving average of the weights during training, and we update the moving average after every optimization step with a decay factor of 0.999. The batch size is 16. We run inference with 20 denoising steps on 500 validation complexes every 5 epochs and use the set of weights with the highest percentage of RMSDs less than 2Å as the final diffusion model. We trained our final score model on four 48GB RTX A6000 GPUs for 850 epochs (around 18 days). The confidence model is trained on a single 48GB GPU. For inference, only a single GPU is required. Scaling up the model size seems to improve performance and future work could explore whether this trend continues further. For the confidence model uses the validation cross-entropy loss is used for early stopping and training only takes 75 epochs. Code to reproduce all results including running the baselines or to perform docking calculations for new complexes is available at `https://anonymous.4open.science/r/DiffDock`.

Hyperparameters. For determining the hyperparameters of DIFFDOCK's score model, we trained smaller models (3.97 million parameters) that fit into 48GB of GPU RAM before scaling it up to the final model (20.24 million parameters) that was trained on four 48GB GPUs. The smaller models were only trained for 250 or 300 epochs, and we used the fraction of predictions with an RMSD below 2Å on the validation set to choose the hyperparameters. Table 2 shows the main hyperparameters we tested and the final parameters of the large model we use to obtain our results. We only did little tuning for the minimum and maximum noise levels of the three components of the diffusion. For the translation, the maximum standard deviation is 19Å. We also experimented with second-order features for the Tensorfield Network but did not find them to help. The complete set of hyperparameters next to the main ones we describe here can be found in our repository.

The confidence model has 4.77 million parameters and the parameters we tried are in Table 3. We generate 28 different training poses for the confidence model (for which it predicts whether or not they have an RMSD below 2Å) with a score model that has a maximum translation standard deviation of 34Å. The score model used to generate the training samples for the confidence model is thus different from the large score model we used during inference.

Table 2: The hyperparameter options we searched through for DIFFDOCK's score model. This was done with small models before scaling up to a large model. The parameters shown here that impact model size (bottom half of the table) are those of the large model. The final parameters for the large DIFFDOCK model are marked in **bold**.

PARAMETER	SEARCH SPACE
USING ALL ATOMS FOR THE PROTEIN GRAPH	YES, **NO**
USING LANGUAGE MODEL EMBEDDINGS	**YES**, NO
USING LIGAND HYDROGENS	YES, **NO**
USING EXPONENTIAL MOVING AVERAGE	**YES**, NO
MAXIMUM NUMBER OF NEIGHBORS IN PROTEIN GRAPH	10, 16, **24**, 30
MAXIMUM NEIGHBOR DISTANCE IN PROTEIN GRAPH	5, 10, **15**, 18, 20, 30
DISTANCE EMBEDDING METHOD	**SINUSOIDAL**, GAUSSIAN
DROPOUT	0, 0.05, **0.1**, 0.2
LEARNING RATES	0.01, 0.008, 0.003, **0.001**, 0.0008, 0.0001
BATCH SIZE	8, **16**, 24
NON LINEARITIES	**RELU**
CONVOLUTION LAYERS	6
NUMBER OF SCALAR FEATURES	48
NUMBER OF VECTOR FEATURES	10

Table 3: The hyperparameter options we searched through for DiffDock's confidence model. The final parameters are marked in **bold**.

Parameter	Search Space
using all atoms for the protein graph	**Yes**, No
using language model embeddings	**Yes**, No
using ligand hydrogens	**No**
using exponential moving average	**No**
maximum number of neighbors in protein graph	10, 16, **24**, 30
maximum neighbor distance in protein graph	5, 10, **15**, 18, 20, 30
distance embedding method	**sinusoidal**
dropout	0, 0.05, **0.1**, 0.2
learning rates	0.03, 0.003, **0.0003**, 0.00008
batch size	**16**
non linearities	**ReLU**
convolution layers	5
number of scalar features	24
number of vector features	6

Runtime. Similar to all the baselines, the preprocessing times are not included in the reported runtimes. For DiffDock the preprocessing time is negligible compared to the rest of the inference time where multiple reverse diffusion steps are performed. Preprocessing mainly consists of a forward pass of ESM2 to generate the protein language model embeddings, RDKit's conformer generation, and the conversion of the protein into a radius graph. We measured the inference time when running on an RTX A100 40GB GPU when generating 10 samples. The runtimes we report for generating 40 samples and ranking them are extrapolations where we multiply the runtime for 10 samples by 4. In practice, this only gives an upper bound on the runtime with 40 samples, and the actual runtime should be faster.

D.3 Baselines: implementation, used scripts, and runtime details

Our scripts to run the baselines are available at `https://anonymous.4open.science/r/DiffDock`. For obtaining the runtimes of the different methods, we always used 16 CPUs except for GLIDE as explained below. The runtimes do not include any preprocessing time for any of the methods. For instance, the time that it takes to run P2Rank is not included for TANKBind, and P2Rank + SMINA/GNINA since this receptor preparation only needs to be run once when docking many ligands to the same protein. In applications where different receptors are processed (such as reverse screening), the experienced runtimes for TANKBind and P2Rank + SMINA/GNINA will thus be higher.

We note that for all these baselines we have used the default hyperparameters unless specified differently below. Modifying some of these hyperparameters (for example the scoring method's exhaustiveness) will change the runtime and performance tradeoffs (e.g., if the searching routine is left running for longer then better poses are likely to be found), however, we leave these analyses to future work.

SMINA [14] improves Autodock Vina with a new scoring-function and user friendliness. The default parameters were used with the exception of setting `-num_modes 10`. To define the search box, we use the automatic box creation option around the receptor with the default buffer of 4Å on all 6 sides.

GNINA [13] builds on SMINA by additionally using a learned 3D CNN for scoring. The default parameters were used with the exception of setting `-num_modes 10`. To define the search box, we use the automatic box creation option around the receptor with the default buffer of 4Å on all 6 sides.

QuickVina-W [12] extends the speed-optimized QuickVina 2 [23] for blind docking. We reuse the numbers from [9] which had used the default parameters except for increasing the exhaustiveness to 64.

GLIDE [2] is a strong heavily used commercial docking tool. These methods all use biophysics based scoring-functions. We reuse the numbers from [9] since we do not have a license. As explained by

[9], the very high runtime of GLIDE with 1405 seconds per complex is partially explained by the fact that GLIDE only uses a single thread when processing a complex. This fact and the parallelization options of GLIDE are explained here `https://www.schrodinger.com/kb/1165`. With GLIDE, it is possible to start data-parallel processes that compute the docking results for a different complex in parallel. However, each process also requires a separate software license.

EquiBind [9], we reuse the numbers reported in their paper and generate the predictions that we visualize with their code at `https://github.com/HannesStark/EquiBind`.

TANKBind [11], we use the code associated with the paper at `https://github.com/luwei0917/TankBind`. The runtimes do not include the runtime of P2Rank or any preprocessing steps. In Table 1 we report two runtimes (0.72/2.5 sec). The first is the runtime when making only the top-1 prediction and the second is for producing the top-5 predictions. Producing only the top-1 predictions is faster since TANKBind produces distance predictions that need to be converted to coordinates with a gradient descent algorithm and this step only needs to be run once for the top-1 prediction, while it needs to be run 5 times for producing 5 outputs. To obtain our runtimes we run the forward pass of TANKBind on GPU (0.28 seconds) with the default batch size of 5 that is used in their GitHub repository. To compute the time the distances-to-coordinates conversion step takes, we run the file `baseline_run_tankbind_parallel.sh` in our repository, which parallelizes the computation across 16 processes which we also run on an Intel Xeon Gold 6230 CPU. This way, we obtain 0.44 seconds runtime for the conversion step of the top-1 prediction (averaged over the 363 complexes of the testset).

P2Rank [25], is a tool that predicts multiple binding pockets and ranks them. We use it for running TANKBind and P2Rank + SMINA/GNINA. We download the program from `https://github.com/rdk/p2rank` and run it with its default parameters.

EquiBind + SMINA/GNINA [9], the bounding box in which GNINA/SMINA searches for binding poses is constructed around the prediction of EquiBind with the `-autobox_ligand` option of GNINA/SMINA. EquiBind is thus used to find the binding pocket and SMINA/GNINA to find the exact final binding pose. We use `-autobox_add 10` to add an additional 10Å on all 6 sides of the bounding box following [9].

P2Rank + SMINA/GNINA. The bounding box in which GNINA/SMINA searches for binding poses is constructed around the pocket center that P2Rank predicts as the most likely binding pocket. P2Rank is thus used to find the binding pocket and SMINA/GNINA to find the exact final binding pose. The diameter of the search box is the diameter of a ligand conformer generated by RDKit with an additional 10Å on all 6 sides of the bounding box.

E Additional Results

E.1 Physically plausible predictions

Table 4: **Steric clashes.** Percentage of test complexes for which the predictions of the different methods exhibit steric clashes. Search-based methods never produced steric clashes.

Method	Top-1 % steric clashes	Top-5 % steric clashes
EQUIBIND	26	-
TANKBIND	6.6	3.6
DIFFDOCK (10)	**2.8**	**0**
DIFFDOCK (40)	**2.2**	**2.2**

Due to the averaging phenomenon of regression-based methods such as TANKBind and EquiBind, they make predictions at the mean of the distribution. If aleatoric uncertainty is present, such as in case of symmetric complexes, this leads to predicting the ligand to be at an un-physical state in the middle of the possible binding pockets as visualized in Figure 9. The Figure also illustrates how DIFFDOCK does not suffer from this issue and is able to accurately sample from the modes.

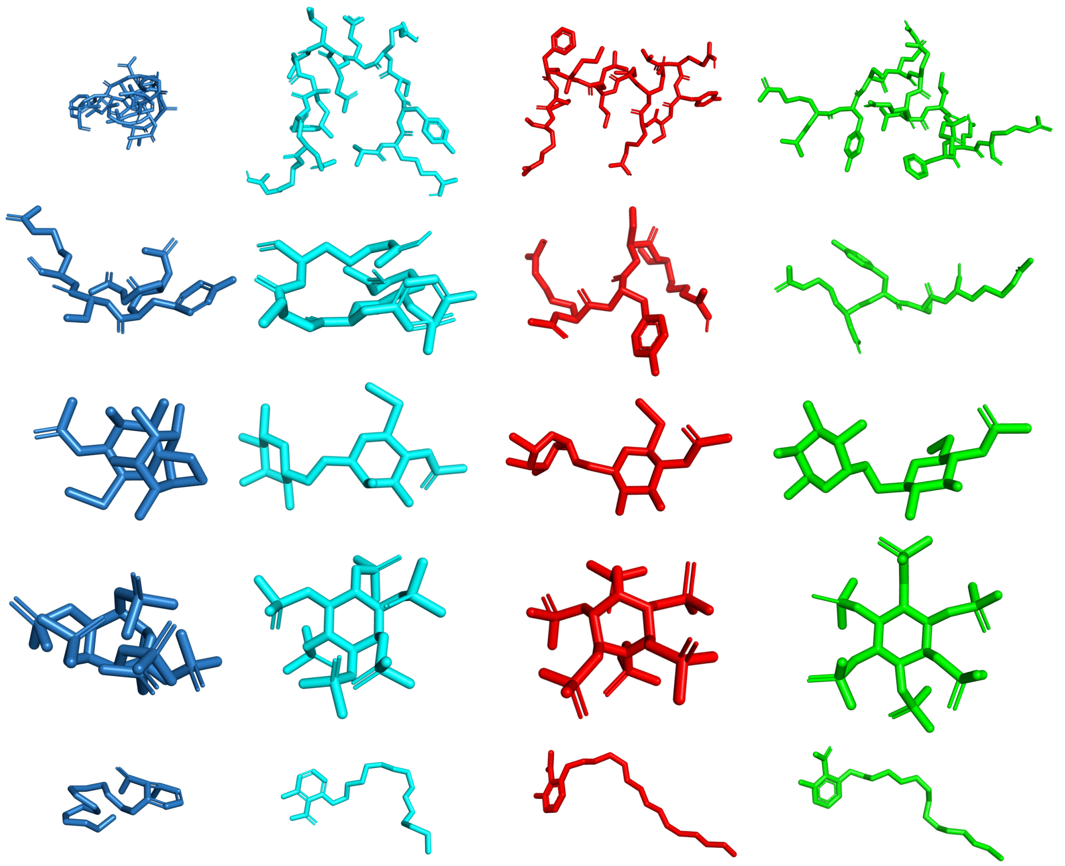

Figure 3: **Ligand self-intersections.** TANKBind (blue), EquiBind (cyan), DiffDock (red), and crystal structure (green). Due to the averaging phenomenon that occurs when epistemic uncertainty is present, the regression-based deep learning models tend to produce ligands with atoms that are close together, leading to self-intersections. DiffDock, as a generative model, does not suffer from this averaging phenomenon, and we never found a self-intersection in any of the investigated results of DiffDock.

In the scenario when epistemic uncertainty about the correct ligand conformation is present, this often results in "squashed-up" predictions of the regression-based methods as visualized in Figure 3. If there is uncertainty about the correct conformer, the square error minimizing option is to put all atoms close to the mean.

These averaging phenomena in the presence of either aleatoric or epistemic uncertainty cause the regression-based methods to often generate steric clashes and self intersections. To investigate this quantitatively, we determine the fraction of test complexes for which the methods exhibit steric clashes. We define a ligand as exhibiting a steric clash if one of its heavy atoms is within 0.4Å of a heavy receptor atom. This cutoff is used by protein quality assessment tools and in previous literature [26]. Table 4 shows that DiffDock, as a generative model, produces fewer steric clashes than the regression-based baselines. We generally observe no unphysical predictions from DiffDock unlike the self intersections that, e.g., TANKBind produces (Figure 3) or its incorrect local structures (Figure 4). This is also visible in the randomly chosen examples of Figure 8 and can be examined in our repository, where we provide all predictions of DiffDock for the test set.

E.2 Further Results and Metrics

In this section, we present further evaluation metrics on the results presented in Table 1. Figure 5-*left* shows the proportion of RMSDs below an arbitrary threshold ϵ with DiffDock exceeding previous

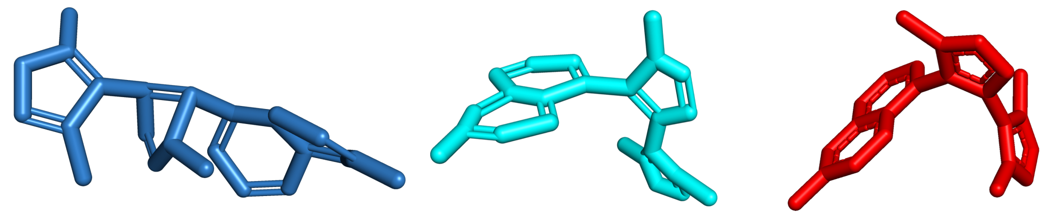

Figure 4: **Chemically plausible local structures.** TANKBind (blue), EquiBind (cyan), and DiffDock (red) structures for complex 6g2f. EquiBind (without their correction step) produces very unrealistic local structures and TANKBind, e.g., produces non-planar aromatic rings. DiffDock's local structures are the realistic local structures of RDKit.

methods for almost every possible ϵ.[3] Figure 5-*right* plots how the model's performance changes with the number of generative samples.

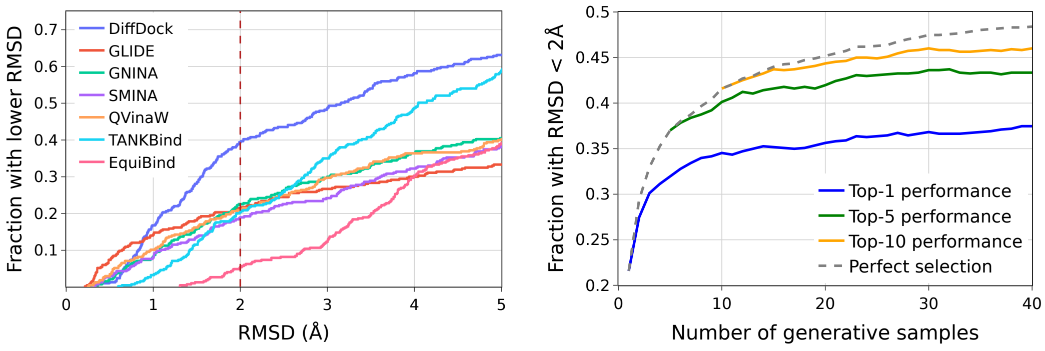

Figure 5: **Left:** cumulative density histogram of the methods' RMSD. **Right:** DiffDock's performance as a function of the number of samples from the generative model. "Perfect selection" refers to choosing the sample with the lowest RMSD.

For both top-1 (Table 5) and top-5 (Table 6) we report: 25th, 50th and 75th percentiles, the proportion below 2Å and below 5Å of both ligand RMSD and centroid distance. Moreover, while [10] advocated against artificial protein set splits and for time-based splits, for completeness, in Table 7 and Figure 6, we report the performances of the different methods when evaluated exclusively on the portion of the test set where the UniProt IDs of the proteins are not contained in the data that is seen by DiffDock in its training and validation.

E.3 Ablation studies

Below we report the performance of our method over different hyperparameter settings. In particular, we highlight the different ways in which it is possible to control the tradeoff between runtime and accuracy in our method. These mainly are: (1) model size, (2) diffusion time, and (3) diffusion samples.

Model size. The final DiffDock score model has 20.24 million parameters from its 6 convolution layers with 48 scalar and 10 vector features. In Table 8 we show the results for a smaller score model with 5 convolutions, 24 scalar, and 6 vector features resulting in 3.97 million parameters that can be trained on a single 48GB GPU. The confidence model used is the same for both score models. We find that scaling up the model size helped improve performance which we did as far as possible using four 48GB GPUs for training. Scaling the model size further is a promising avenue for future work.

Diffusion steps. Another hyperparameter determining the runtime of the method during inference is the number of steps we take during the reverse diffusion. Since these are applied sequentially DiffDock's runtime scales approximately linearly with the number of diffusion steps. In the rest of

[3]With the exception of very small ϵ <1Å where GLIDE performs better.

Table 5: **Top-1 PDBBind docking.** The top half contains methods that directly find the pose; the bottom half those that use a pocket prediction method. The last two lines show our method's performance where, in parenthesis, we specify the number of poses sampled from the generative model.

	Ligand RMSD					Centroid Distance				
	Percentiles ↓			% below threshold ↑		Percentiles ↓			% below thresh. ↑	
Methods	25th	50th	75th	5 Å	2 Å	25th	50th	75th	5 Å	2 Å
QVINA-W	2.5	7.7	23.7	40.2	20.9	0.9	3.7	22.9	54.6	41.0
GNINA	2.4	7.7	17.9	40.8	22.9	0.8	3.7	23.1	53.6	40.2
SMINA	3.1	7.1	17.9	38.0	18.7	1.0	2.6	16.1	59.8	41.6
GLIDE (c.)	2.6	9.3	28.1	33.6	21.8	0.8	5.6	26.9	48.7	36.1
EQUIBIND	3.8	6.2	10.3	39.1	5.5	1.3	2.6	7.4	67.5	40.0
TANKBIND	2.5	4.0	8.5	59.0	20.4	0.9	1.8	4.4	77.1	55.1
P2RANK+SMINA	2.9	6.9	16.0	43.0	20.4	0.8	2.6	14.8	60.1	44.1
P2RANK+GNINA	1.7	5.5	15.9	47.8	28.8	0.6	2.2	14.6	60.9	48.3
EQUIBIND+SMINA	2.4	6.5	11.2	43.6	23.2	0.7	2.1	7.3	69.3	49.2
EQUIBIND+GNINA	1.8	4.9	13	50.3	28.8	0.6	1.9	9.9	66.5	50.8
DIFFDOCK (10)	1.5	3.6	**7.3**	61.3	34.5	**0.5**	**1.2**	3.2	**80.9**	63.7
DIFFDOCK (40)	**1.4**	**3.5**	7.4	**63.3**	**37.5**	**0.5**	**1.2**	**3.0**	80.6	**64.5**

Table 6: **Top-5 PDBBind docking.** The top half contains methods that directly find the pose; the bottom half those that use a pocket prediction method. The last two lines show our method's performance where, in parenthesis, we specify the number of poses sampled from the generative model.

	Ligand RMSD					Centroid Distance				
	Percentiles ↓			% below threshold ↑		Percentiles ↓			% below thresh. ↑	
Methods	25th	50th	75th	5 Å	2 Å	25th	50th	75th	5 Å	2 Å
GNINA	1.6	4.5	11.8	52.8	29.3	0.6	2.0	8.2	66.8	49.7
SMINA	1.7	4.6	9.7	53.1	29.3	0.6	1.85	6.2	72.9	50.8
TANKBIND	2.1	3.4	6.1	67.5	24.5	0.8	1.4	2.9	86.8	62.0
P2RANK+SMINA	1.5	4.4	14.1	54.8	33.2	0.6	1.8	12.3	66.2	53.4
P2RANK+GNINA	1.4	3.4	12.5	60.3	38.3	0.5	1.4	9.2	69.3	57.3
EQUIBIND+SMINA	1.3	3.4	8.1	60.6	38.6	0.5	1.3	5.1	74.9	58.9
EQUIBIND+GNINA	1.4	3.1	9.1	61.7	39.1	0.5	1.1	5.3	73.7	60.1
DIFFDOCK (10)	1.3	2.8	5.2	74.0	40.1	0.5	1.0	2.2	86.5	72.1
DIFFDOCK (40)	**1.2**	**2.5**	**4.7**	**77.1**	**43.3**	**0.4**	**0.9**	**2.1**	**87.0**	**74.2**

the paper, we always use 20 steps, but in Figure 7 we show how the performance of the model varies with the number of steps. We note that the model reaches nearly the full performance even with just 10 steps, suggesting that the model can be sped up 2x with a small drop in accuracy.

Diffusion samples. Given a score-based model and a number of steps for the diffusion model, it remains to be determined how many independent samples N to query from the diffusion model and then feed to the confidence model. As expected the more samples the confidence model receives the more likely it is that it will find a pose that it is confident about and, therefore, the higher the performance. The runtime of DIFFDOCK on GPU scales sublinearly until the different samples fit in parallel in the model (depends on the protein size and the GPU memory) and approximately linearly for larger sample sizes (however it can be easily parallelized across different GPUs). In Figure 5 we show how the success rate for the top-1, top-5, and top-10 prediction change as a function of N. For example, for the top-1 prediction, the proportion of the prediction with RMSD below 2Å varies between 22% of a random sample of the diffusion model ($N = 1$) to 38% when the confidence model is allowed to choose between 40 samples.

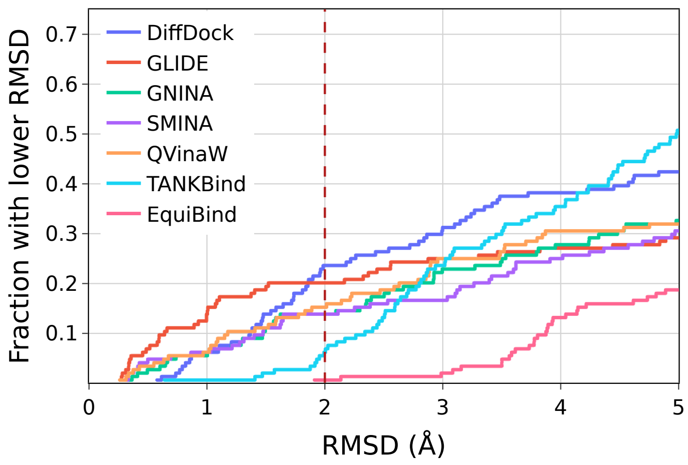

Figure 6: **PDBBind docking on unseen receptors.** Shown is the cumulative density histogram of the methods' RMSD.

Table 7: **PDBBind docking on unseen receptors.** Percentage of predictions for which the RMSD to the crystal structure is below 2Å and the median RMSD. "*" indicates the method run exclusively on CPU, "-" means not applicable; some cells are empty due to infrastructure constraints.

	Top-1 RMSD		Top-5 RMSD		Average
Method	%<2	Med.	%<2	Med.	Runtime (s)
QVINAW	15.3	10.3			49*
GNINA	14.0	13.6	23.0	7.0	127
SMINA	14.0	8.5	21.7	6.7	126*
GLIDE	**19.6**	18.0			1405*
EQUIBIND	0.7	9.1	-	-	**0.04**
TANKBIND	6.3	**5.0**	11.1	4.4	0.7/2.5
DIFFDOCK (10)	16.4±1.2	6.3±0.3	20.8±1.0	4.3±0.1	10
DIFFDOCK (40)	19.2±1.8	6.3±0.4	**25.2±0.7**	**3.8±0.1**	40

E.4 Affinity prediction

To validate the quality of the predicted poses, we also do some experiments in predicting the binding affinity labels already present in PDBBind. In this section we report some preliminary results on this task that show that a simple approach can already achieve results competitive with the state-of-the-art. We leave a more thorough and sophisticated analysis on how to best use the DIFFDOCK framework for binding affinity to future work.

Affinity prediction framework. We train the binding affinity predictor by generating a fixed number of poses with the diffusion model and then feeding them to an affinity prediction model with

Table 8: **Model size comparison.** All methods receive a small molecule and are tasked to find its binding location, orientation, and conformation. Shown is the percentage of predictions for which the RMSD to the crystal structure is below 2Å and the median RMSD.

	Top-1 RMSD (Å)		Top-5 RMSD (Å)		Average
Method	%<2	Med.	%<2	Med.	Runtime (s)
DIFFDOCK-SMALL (10)	26.2	3.9	34.2	3.0	7
DIFFDOCK-SMALL (40)	32.0	3.9	40.8	2.7	28
DIFFDOCK (10)	34.5±0.3	3.61±0.05	40.1±1.1	2.79±0.09	10
DIFFDOCK (40)	**37.5±0.5**	**3.46±0.04**	**43.3±0.5**	**2.53±0.05**	40

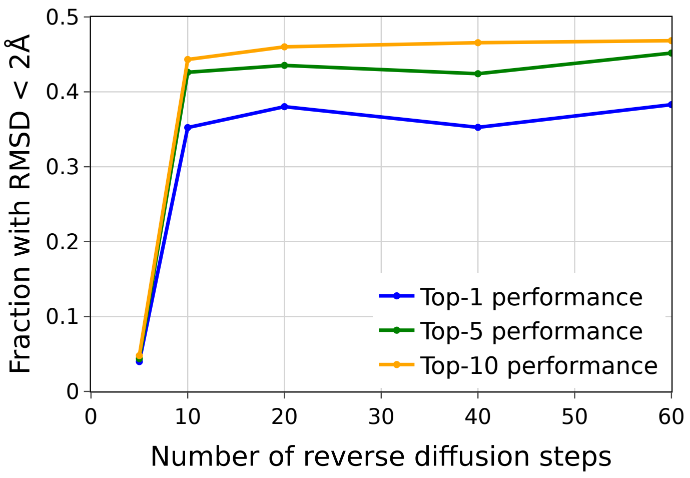

Figure 7: Ablation study on the number of reverse diffusion steps.

architecture almost analogous to the confidence model. This affinity prediction model takes in the poses as a single heterogeneous graph with a single receptor but multiple sets of ligand nodes, which have edges to the same receptor but not among themselves. After the final interaction layer, the scalar representations of nodes in each ligand are aggregated with a mean pooling and passed through a set of dense layers (as it is done for the confidence prediction). Then, the representations of the different ligands are aggregated using multiple permutation invariant aggregators (mean, maximum, minimum, and standard deviation) as in [27], and transformed with another set of dense layers producing a single output, the predicted affinity.

Dataset, baselines, and training. To train we use PDBBind with the same splits used to train the diffusion and confidence models. This provides for each of the complexes an affinity measure that consists of inhibiting concentration ($IC50$), inhibition constant (K_i), or dissociation constant (K_d) and its conversion to the $-\log K_d/K_i$ metric. As baselines, we use a series of state-of-the-art sequence-based and structure-based methods: TransformerCPI [28], MONN [29], IGN [30], PIGNet [31], HOLOPTOT [32], STAMPDPI [33] and TANKBind [11]. We take the baselines' performances from [11].

Table 9: **Binding affinity prediction.** Prediction of $-\log K_d/K_i$ on PDBBind. The baseline numbers are from [11]. No hyperparameter tuning was performed for DiffDock's performance.

Methods	RMSE ↓	Pearson ↑	Spearman ↑	MAE ↓
TransCPI	1.741	0.576	0.540	1.404
MONN	1.438	0.624	0.589	1.143
PIGNet	2.640	0.511	0.489	2.110
IGN	1.433	0.698	0.641	1.169
HOLOPROT	1.546	0.602	0.571	1.208
STAMPDPI	1.658	0.545	0.411	1.325
TANKBind	**1.346**	**0.726**	0.703	1.070
DiffDock	1.347	0.692	**0.718**	**1.052**

Results. The results presented in Table 9 highlight how even preliminary results with a straightforward way of using DiffDock's predictions for affinity prediction achieve a performance that is on par with the state-of-the-art. We hope this can motivate future work on better integrating affinity prediction in the method and scaling to larger amounts of data.

E.5 Visualizations

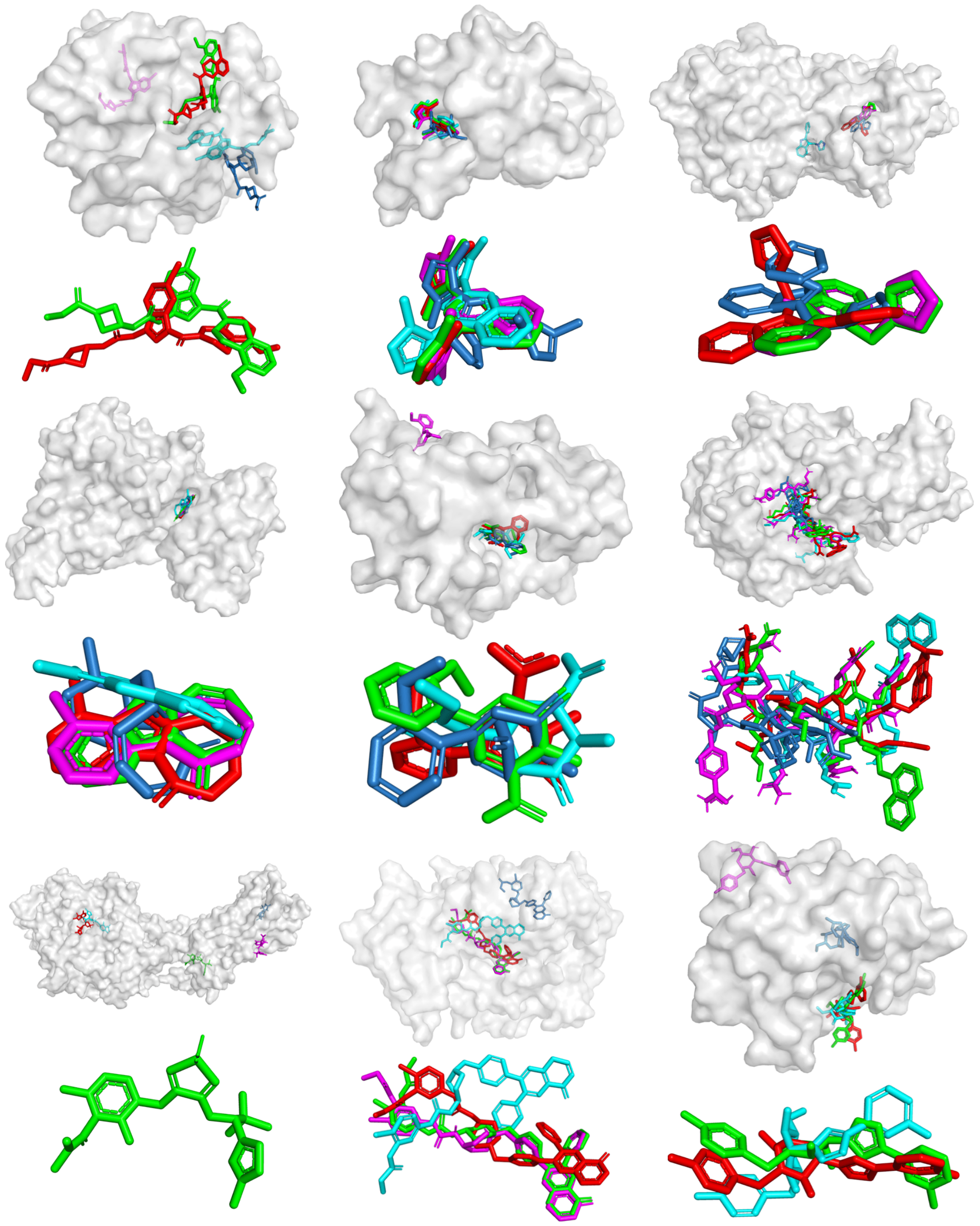

Figure 8: **Randomly picked examples.** The predictions of TANKBind (blue), EquiBind (cyan), GNINA (magenta), DIFFDOCK (red), and crystal structure (green). Shown are the predictions once with the protein and without it below. The complexes were chosen with a random number generator from the test set. TANKBind often produces self intersections (examples at the top-right; middle-middle; middle-right; bottom-right). DIFFDOCK and GNINA sometimes almost perfectly predict the bound structure (e.g., top-middle). The complexes in reading order are: 6p8y, 6mo8, 6pya, 6t6a, 6e30, 6hld, 6qzh, 6hhg, 6qln.

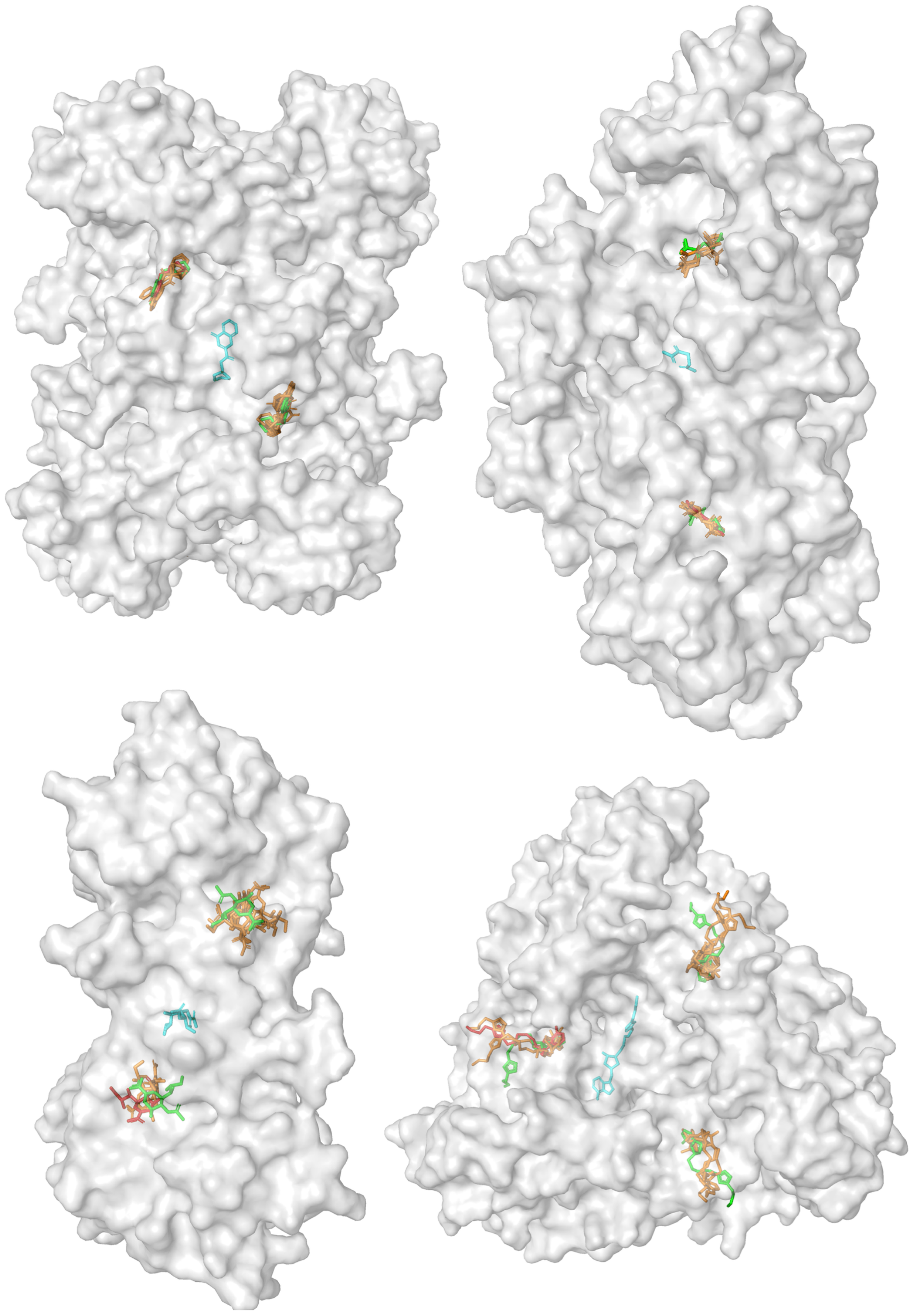

Figure 9: **Symmetric complexes and multiple modes.** EquiBind (cyan), DiffDock highest confidence sample (red), all other DiffDock samples (orange), and the crystal structure (green). We see that, since it is a generative model, DiffDock is able to produce multiple correct modes and to sample around them. Meanwhile, as a regression-based model, EquiBind is only able to predict a structure at the mean of the modes. The complexes are unseen during training. The PDB IDs in reading order: 6agt, 6gdy, 6ckl, 6dz3.

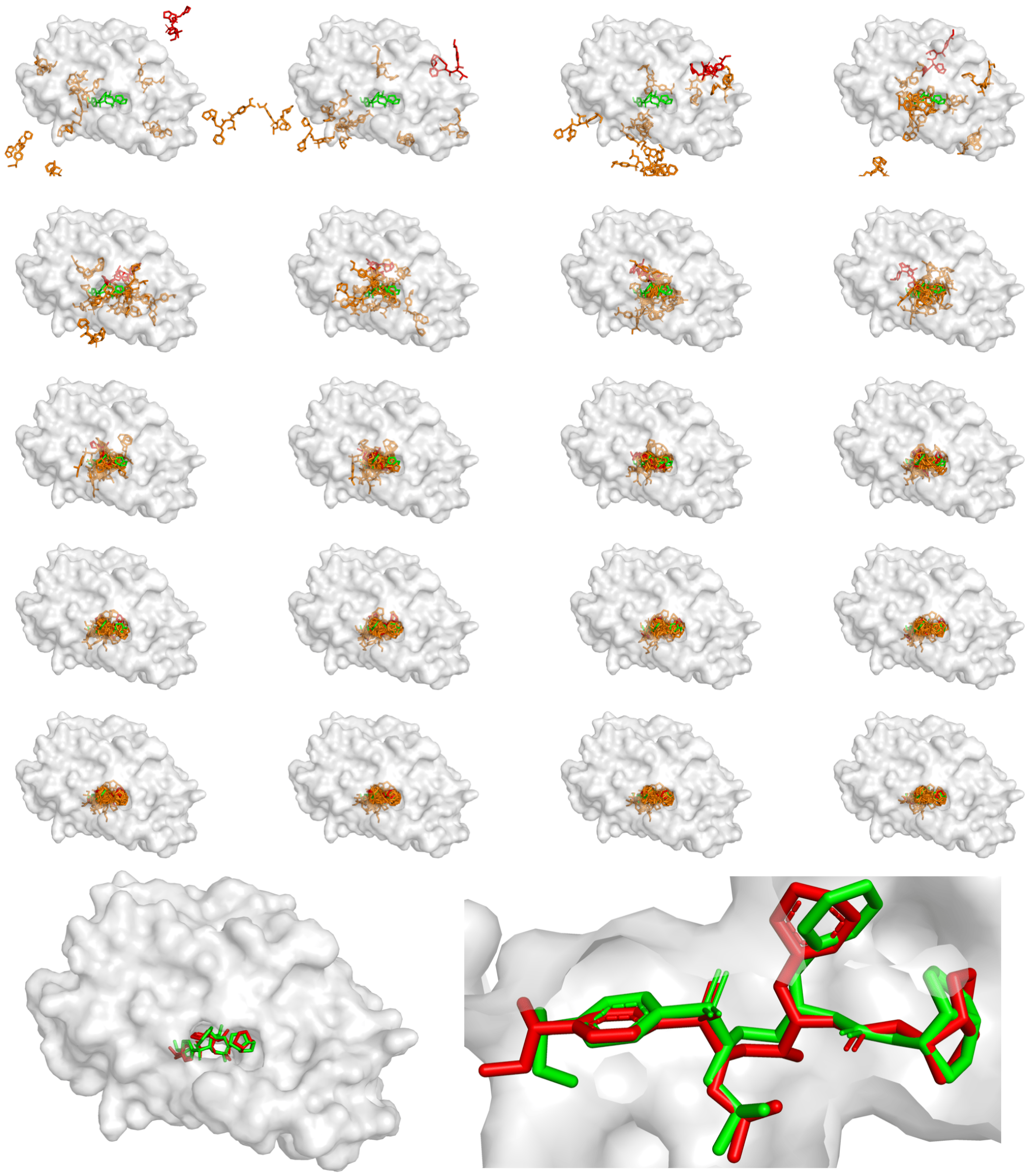

Figure 10: **Reverse Diffusion.** Reverse diffusion of a randomly picked complex from the test set. Shown are DIFFDOCK highest confidence sample (red), all other DIFFDOCK samples (orange), and the crystal structure (green). Shown are the 20 steps of the reverse diffusion process (in reading order) of DIFFDOCK for the complex 6oxx. Videos of the reverse diffusion are available at `https://anonymous.4open.science/r/DiffDock/visualizations/README.md`.