Supplementary Materials

Analysis on invariance in diffusion models

In the forward diffusion process for positions, which reads

$$x_{i,t} = \alpha_t x_{i,0} + \sigma_t \epsilon_{i,t} \tag{27}$$

If a translation t is performed on the input positions and the added noise, then

$$\alpha_t(x_{i,0} + t) + \sigma_t(\epsilon_{i,t} + t) = (x_{i,t} + t) + (1 - \alpha_t)t + \sigma_t t. \tag{28}$$

It indicates that the intermediate diffusion states are not translational-equivariant. Then, the translation equivariance in the reserve process is not meaningful. For this reason, we fix the CoM of the input positions and each intermediate state, so t will always equal 0.

The zero-CoM trick circumvents the problem.

By contrast, for a rotation transformation,

$$\alpha_t(Rx_{i,0}) + \sigma_t(R\epsilon_{i,t}) = Rx_{i,t}. \tag{29}$$

As $\epsilon_{i,t} \sim N(0,1)$, by this way

$$p(R\epsilon_{i,t}) = p(\epsilon_{i,t}), \tag{30}$$

and thus

$$q(Rx_{i,t}|Rx_{i,0}) = N(\alpha_t(Rx_{i,0}), \sigma_t I) \tag{31}$$

For the true position distribution, $q(Rx_{i,0}) = q(x_{i,0})$

$$\begin{aligned} q(Rx_{i,t}) &= q(Rx_{i,t}|Rx_{i,0})q(Rx_{i,0}) \\ &= q(x_{i,t}) \end{aligned} \tag{32}$$

In this way, the intermediate states diffusion process satisfies the rotation invariance. And to learn the reverse process, we only require $p(Rx_{i,0}|Rx_{i,t})$ to be rotational-equivariant. By Eq. (25), the probability of each generated intermediate state will also be rotation invariance.

Details in diffusion models

Diffusion Schedule. As α_t and σ_t are two parameters changing gradually along t-axis. Lots of noise schedules proposed, such as Linear[27] and Cosine[69], here we choose a simple polynomial schedule

$$\alpha_t^2 = (1 - (t/3)^3)^3, \tag{33}$$

and variance-preserving process as

$$\sigma_t^2 = 1 - \alpha_t^2. \tag{34}$$

Sampling Process. We give the detailed pseudo-codes of sampling algorithm of DiffBP in Algorithm S1.

Algorithm S1 Sampling from DiffBP

Input: Zero-centered protein $\{a_j, x_j\}_{j=1}^{M}$, pre-trained network φ_ω and graph denoiser ϕ_θ

Compute $[x_c^{mol}, N^{mol}] = \varphi_\omega(\{a_j, x_j\}_{j \in I_{pro}})$

Subtract x_c^{mol} from $\{a_j, x_j\}_{j=1}^{M}$, and set $N := N^{mol}$

Sample $x_{i,T} \sim N(0, I)$

For t in $T, T-1, \ldots, 1$ where $s = t - 1$ **do**

Sample $\epsilon \sim N(0, \boldsymbol{I})$

Subtract center of mass from ϵ

Compute $[\hat{\epsilon}_{i,t}, \hat{p}_{i,0}] = \phi_\theta(\{(a_{j,t}, x_{j,t})\}_{j=1}^{N+M}, t)$

Compute $x_{i,s} = \frac{1}{\alpha_{t|s}} x_{i,t} - \frac{\sigma_{t|s}^2}{\alpha_{t|s}\sigma_t} \hat{\epsilon}_{i,t} + \sigma_{t|s}' \epsilon$

Sample uniformly I_{rcv}, $|I_{rcv}|/M = (T-t)/T$

If $i \in I_{rcv} \cap I_{mol}$ **and** $a_{i,t} = K + 1$ **then**

Sample $a_{i,0} \sim Cat(\hat{p}_{i,0})$

end if

end for

output: $M = \{(a_{i,0}, x_{i,0})\}_{i=1}^{N}$

Binding Affinity

As in Pocket2Mol, the UFF refinement is not mentioned, so here we give the Binding Affinity of Pocket2Mol-without-UFF in **Table S1.**

Table S1. The Binding Affinity of Pocket2Mol-without-UFF.

	Ratio	MPBG	ΔBinding
Small	36.62%	43.77%	3.96%
Medium	59.02%	28.62%	7.47%
Large	4.36%	12.82%	13.98%
Overall		33.47%	6.46%

Detailed drug properties

Since there is a high correlation between binding scores and molecule sizes, here we list the other score functions of drug properties to figure out their preferences in **Table S2**.

It shows that score functions of QED, SA and LPSK have their preference for molecule sizes. QED SA, and LPSK tend to give smaller molecules higher scores.

Table S2. Detailed drug properties split into groups by molecule size.

□	3DSBDD					Pocket2Mol				
□	Ratio	QED	SA	SIM	LPSK	Ratio	QED	SA	SIM	LPSK
Small	41.45%	0.4271	0.5418	0.3537	0.7728	36.62%	0.5479	0.5745	0.4176	0.8929
Medium	54.06%	0.3562	0.5089	0.3463	0.6126	59.02%	0.4923	0.5333	0.4463	0.7731
Large	4.48%	0.2568	0.4207	0.3277	0.3639	4.36%	0.4443	0.4105	0.4033	0.6921
Overall	□	0.3811	0.5185	0.3485	0.6678	□	0.5106	0.5430	0.4104	0.8134
□	GraphBP					DiffBP				
□	Ratio	QED	SA	SIM	LPSK	Ratio	QED	SA	SIM	LPSK
Small	27.72%	0.5092	0.5312	0.2757	0.8893	5.22%	0.5269	0.5676	0.3081	0.8335
Medium	32.03%	0.4596	0.5183	0.2628	0.6931	75.19%	0.4663	0.5598	0.3297	0.7043
Large	37.97%	0.2592	0.4467	0.2289	0.3360	19.59%	0.3317	0.4452	0.3318	0.6696
Overall	□	0.3830	0.4828	0.2707	0.5961	□	0.4431	0.5377	0.3290	0.7042

Atom type analysis

Expect the sub-structure, we give the ratio of different element types of atoms, for further comparison, as shown in **Table S3**. Note that we only count the percentage of heavy atoms, in line with the generative models. It shows that for the most common elements like ‘C’,‘O’,‘N’, DiffBP and Pocket2Mol generate the average ratio that most closely resembles the true distribution. However, for the uncommon elements, such as ‘S’ and ‘Cl’, DiffBP generates more than reference.

Table S3. Atom type ratio of different methods.

Atom type	Train	Test	3DSBDD	Pocket2Mol	GraphBP	DiffBP
C	16.02	14.03	11.45	**12.29**	20.57	**15.25**
N	2.82	2.35	1.91	**2.31**	1.32	**2.12**
O	3.79	4.98	2.04	**2.92**	1.86	**5.49**

F	0.31	0.04	0.03	0.04	**0.16**	**0.09**
P	0.24	0.39	0.08	**0.11**	0.06	0.06
S	0.25	0.22	0.03	0.02	**0.11**	1.45
Cl	0.15	0.09	0	0	0.07	0.73
Br	0.05	0	0	0	0.04	0
Se	0	0	0	0	0.02	0.01
I	0.01	0	0	0	0.03	0
Other	0	0	0	0	0	0

Implementation details

Datasets. The training set includes 10,000 protein-ligand paired samples, of which 99981 are valid molecules. 100 protein pockets are used for the test, and for each sample, 100 molecules are generated in each trial.

Hyperparameters. In practice, we found that GVP is more numerically stable than EGNN, so we chose 12-layer GVP for the pre-generation model and 9-layer GVP for Graph denoiser. The hidden dimensions of vertices are fixed as 256 and 64 for invariant node attributes (atom types) and equivariant geometry positions, respectively. In training, the batch size is set as 8, and the initial learning rate is set as 10^{-4}. AdamW optimizer is used, together with the Exponential Moving Average training trick.

Hardware. We use a single NVIDIA A100(81920MiB) GPU for a trial. The codes are implemented in Python 3.9 mainly with Pytorch 1.12, and run on Ubuntu Linux.