Supporting Information: Structure-based *de novo* drug design using 3D deep generative models

Yibo Li,[†] Jianfeng Pei,[*,‡] and Luhua Lai[*,†,‡,¶]

† *Center for Life Sciences, Academy for Advanced Interdisciplinary Studies, Peking University, Beijing 100871, China*

‡ *Center for Quantitative Biology, Academy for Advanced Interdisciplinary Studies, Peking University, Beijing 100871, China*

¶ *BNLMS, College of Chemistry and Molecular Engineering, Peking University, Beijing 100871, China*

E-mail: jfpei@pku.edu.cn; lhlai@pku.edu.cn

1 Supplementary Methods

1.1 The molecule generation process

The task of our generative model is to produce molecular graphs $G = (V, E, A, B, X)$, where V is the set of nodes (atoms), E is the set of edges (bonds), $A = \{a_v\}_{v \in V}$ are the atom type labels, $B = \{b_{uv}\}_{(u,v) \in E}$ are bond type labels, and $X = \{\mathbf{x}_v\}_{v \in V}$ are the 3D positions of each atoms. Note that in theory, the bond order can be inferred directly from distances between atoms, as done in several previous works,[S1,S2] but existing bond type assignment algorithms are generally sensitive to errors, even small ones that can be later corrected. To make our model more robust, we explicitly output the bond types B.

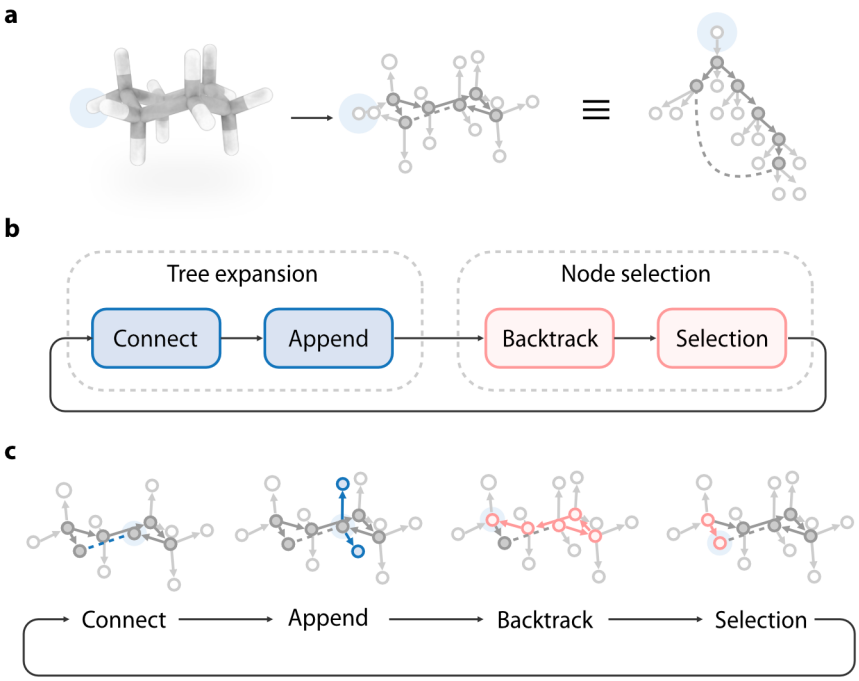

Figure S1: The molecule generation process, using cyclohexane as an example: **a.** The model generates a molecule by generating its spanning tree in a depth-first manner. The root of the tree is highlighted in light blue; **b.** The model builds the spanning tree iteratively, and two sub-steps are performed at each iteration: tree expansion and node selection. At the tree expansion step, the model connects the currently focused atom with another atom (the "connect" operations), or add new atoms to the focused atom (the "append" operations). At the node selection step, the model backtracks the spanning tree to find the next focus atom (the "backtrack" and "selection" operations); **d.** An example showing a few steps during the generation of cyclohexane. Blue atoms or bonds indicates newly added structures during the tree expansion step. Pink arrows indicate the process of finding the next focused atom. The focused atom at each step is highlighted using a light blue circle.

Our proposed model generates the graph in a step-by-step manner. More specifically, the model generates a molecular graph by iteratively building its spanning tree. A spanning tree of G is a tree structure that contains all nodes in G (see Fig. S1**a**). Tree-based structures are much simpler than general unconditional graphs, and the generation process is more straightforward using a depth-first approach. At each iteration, the following two steps are performed to build the spanning tree (Fig. S1**b**):

- Node selection: The model selects a "focus atom" from the set of suitable atoms that have already been generated. An atom is suitable for becoming a focused atom if it is not being "capped". An atom is capped either because its valency constraint has been reached, or it is actively labeled as capped by the model.

- Tree expansion: The model performs edits around the focused atom, by either adding new atoms to it (the "append" operation) or by connecting it with another existing atom (the "connect" operation).

During the "node selection" step, the model searches through the spanning tree to find the next focused atom:

- If the currently focused atom has a child atom whose valence has not been filled, the model will select that atom to be the next focus. If multiple such children exist, a ranking is performed and the highest-ranking child is selected;

- If no such child exists for the current focus, a "backtracking" operation is performed to find an ancestor who has such children. And that child is then selected as the next focus.

This process terminates when there are no atoms suitable for becoming the "focused atom", that is, the valences of all atoms have been filled. To better illustrate this process, the full generation path of the pyridine molecule is given in Fig. S2. During the generation process, there are a variety of decisions that need to be made by the model:

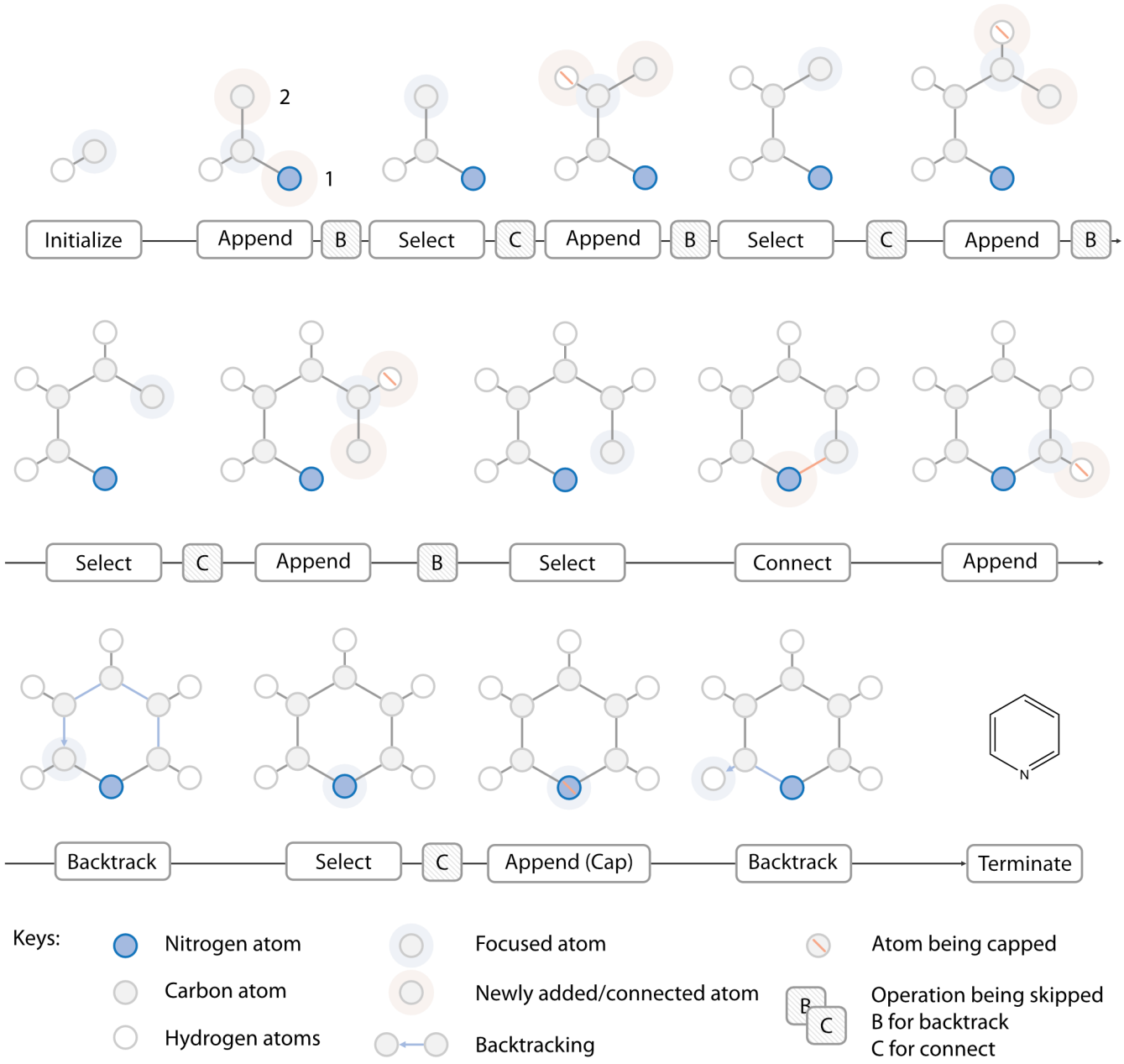

Figure S2: The full generation process of a pyridine molecule.

- During the "connect" operation, the model needs to decide which atom to connect with, using what type of bond;
- During the "append" operation, the model needs to decide how many atoms should be added to the graph, their atom type, 3D location, and the type of bonds connecting them to the focused atom.
- The model also needs to output the ranking of each atom, which will be used in the node selection step.

Those decisions are all made using a neural network with a novel architecture we called L-Net. L-Net is composed of two parts: The first part is a state encoder, which maps the intermediate molecular structure G_i at step i into a continuous representation $\mathbf{h}_i = f_\theta(G_i)$. The second part is a policy network, which assign a probability value to each available action based on the current state $p_\theta(a|\mathbf{h}_i)$. The architecture of L-Net is explained in detail from Section S1.2 though 1.5. To make the network capable of generating drug-like molecules, we construct a drug-like subset of the ChEMBL dataset[S3] and created an "expert trajectory" for generating each molecule in the dataset. L-Net is then trained by imitating those trajectories. Data collection and preprocessing workflows are given in Section S1.7, and the training details are given in Section S1.8. Finally, to validate the model's performance, we designed a set of evaluation metrics which are discussed in Section S1.11.

1.2 The architecture of the state encoder

At iteration i, the state encoder of L-Net is responsible for mapping the current molecular graph G_i to continuous representations $h_i = (\mathbf{h}_{i,\mathrm{g}}, \{\mathbf{h}_{i,v}\}_{v\in V_i}) = f_\theta(G_i)$, where $\mathbf{h}_{i,\mathrm{g}}$ is the graph level representation, and $\{\mathbf{h}_{i,v}\}_{v\in V_i}$ are atom level representations. The architecture of f_θ is shown in Fig. S3. The network adopts a U-net structure.[S4] The input is first fed into an embedding layer to create the input representation for atoms and bonds. It is then passed into the U-net, which is built from convolutional layers, pooling layers, and unpooling

layers. The convolutional layers adopt the architecture of MPNN,[S5] and are organized into DenseNet blocks[S6] to improve the performance. Pooling layers and unpooling layers use a node clustering method that is specifically designed for this use-case. The results are collected and fed to the policy network.

The following sections are devoted to giving detailed explanations of the individual components of the state encoder. We first describe the embedding layer in Section S1.3, and then graph convolution layer in Section S1.4. Pooling and unpooling layers, as well as the node clustering algorithm, are discussed in Section S1.5.

1.3 The embedding layers

1.3.1 Embeddings of atom and bond types

The embeddings of atom and bond types are created by indexing through a trainable lookup table. We also add "temporal encodings" to each atom to record the time that atom is added to the graph, similar to what is done by Vaswani et al.[S7]

1.3.2 Local coordinate system and rotational covariance

Besides type information, the position of each atom should also be included in the input. Ideally, this should be done in a rotationally and translationally covariant way. More specifically, our model parametrize a probability distribution (called the "policy") in 3D space $p(\mathbf{x}|G)$ to indicate where the new atom should be placed (Fig. S4**a**). If we rotate the existing structure G by a certain amount, the probability distribution $p(\mathbf{x}|G)$ should also be rotated by the same amount (Fig. S4**b**). Mathematically, this is expressed by the equation $p(\mathbf{x}|T(G)) = p(T^{-1}(\mathbf{x})|G)$, where T is an spatial operation from the special Euclidean group $SE(3)$.

Enforcing rotational covariance into the network is a non-trivial task. Previous works have developed specialized network architectures, such as tensor field network[S8] or Cormorant,[S9] to ensure rotational covariance. Those works are theoretically elegant but are difficult to

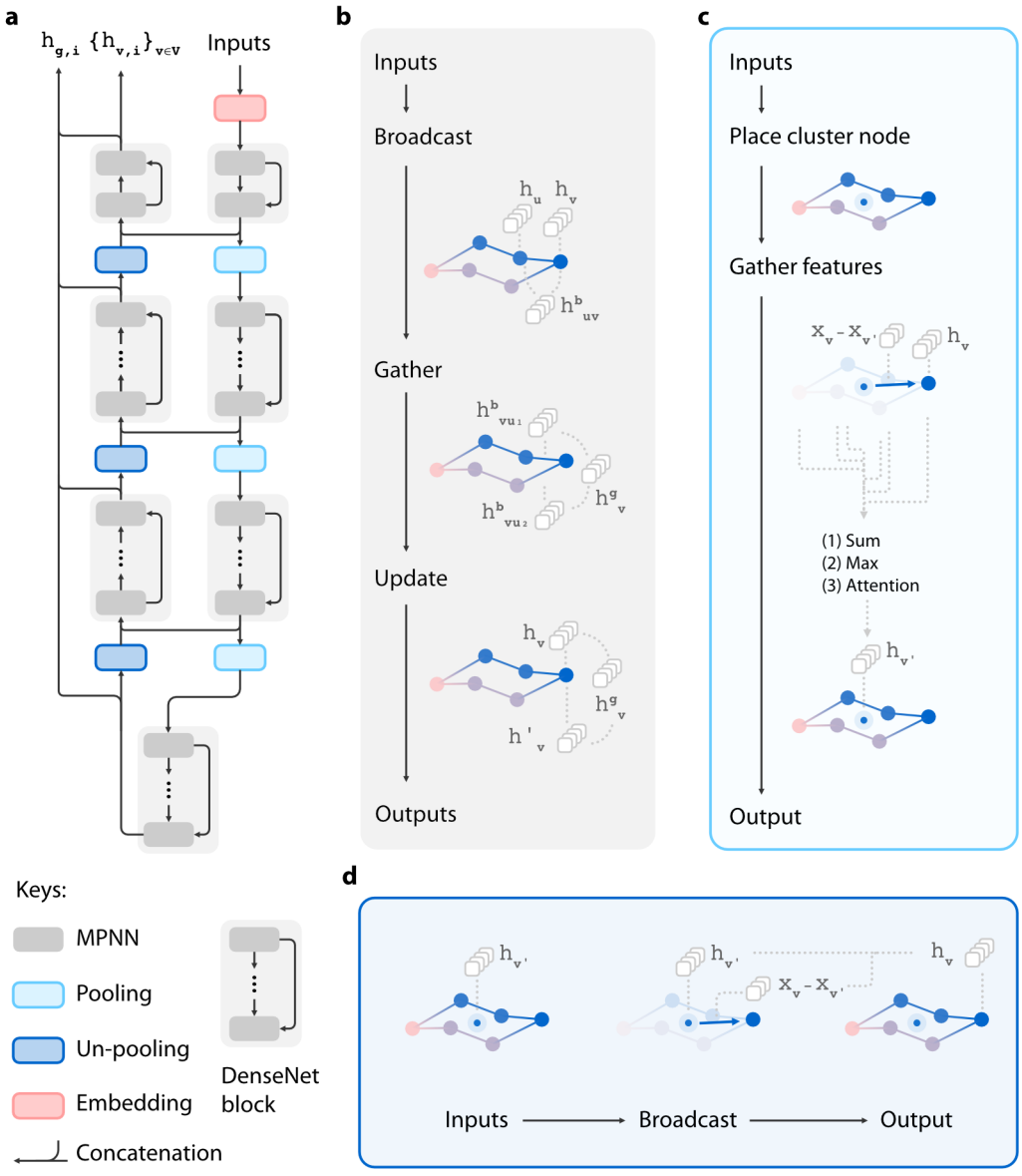

Figure S3: An overview of the architecture for the state encoder. **a.** The overall architecture of the state encoder; **b.** The architecture of each graph convolutional layer; **c.** The architecture of each pooling layer; **c.** The architecture of each unpooling layer.

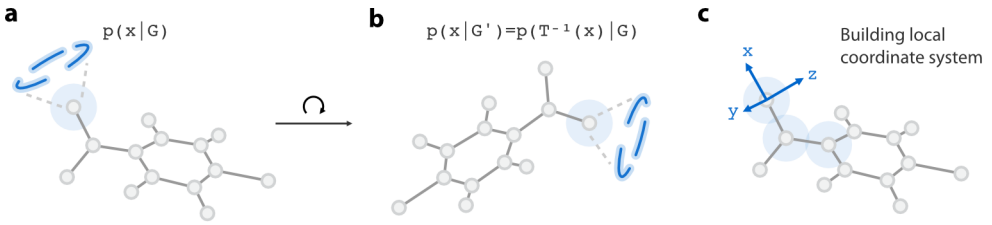

Figure S4: Enforcing rotational covariance using a local coordinate system. An ideal 3D molecule generative model should be rotationally equivariant. If the existing structure is rotated by a certain amount (**a**), the atom placement policy $p(x|G)$ should also be rotated equivalently (**b**). We solve this problem by introducing a local coordinate system in the focused atom (**c**). Since this coordinate system is rotationally covariant, so will the probability distribution defined in the coordinate system.

implement, and their expressiveness may be restricted in some highly symmetrical cases.

Here, we adopt a much simpler yet effective approach, by creating a local coordinate system at the current focus using its neighbor atoms (Fig. S4**c**). We then express the distribution in the local coordinate system $q(\mathbf{x}|G) = p(T(\mathbf{x})|T(G))$, where T indicates the transformation from global to local coordinate. It is easy to verify that q is rotationally covariant, even if p is not, since the local coordinate system is covariant against rotation. Similar methods have been previously used to construct the 3D representation of proteins,[S10] but to our knowledge, we are the first to use this technique in the generative model of small molecules.

Under this framework, the 3D information feed into the neural network are all under the local coordinate system:

$$\tilde{\mathbf{x}}_v = M(\mathbf{x}_v - \mathbf{x}_{v'})$$

$$\tilde{\mathbf{x}}_{uv} = \tilde{\mathbf{x}}_u - \tilde{\mathbf{x}}_v$$

Where $\tilde{\mathbf{x}}_v$ is the position feature of atom v, $\tilde{\mathbf{x}}_{uv}$ is the position feature of bond uv, and M is the matrix for transforming into the local coordinate system.

1.4 Graph convolutional layers

The major components of the state encoder are graph convolutional (GC) layers. The GC architecture used in this work is similar to that used before,[S11] with broadcast, gathering and update operations parametrized using linear layers with elu activation function (as shown in Fig. S3**b**). The only difference lies in the gathering operation. Besides summation and maximization, we add attention as an additional reduction method to improve the expressiveness of the model. Also similar to the previous work, we add "virtual" bonds to the graph to increase the size of receptive fields for each GC layer.

The GC layers are organized into multiple DenseNet blocks (as shown in Fig. S3). DenseNet is a type of network architecture that aims to increase the performance scalability for deeper networks by introducing short connections between any two layers.[S6] There are three major hyper-parameter for DenseNet: the growth rate, the bottleneck size, and the network depth. We experiment with three configurations of DenseNet architectures:

The standard configuration, with a bottleneck size of 94, a growth rate of 24, and the depths of DenseNet blocks (in the order of dataflow in U-Net) [2, 4, 6, 8, 6, 4, 2];

The shallow DenseNet, with the same bottleneck size and growth rate as the basic configuration, and change the depth of DenseNet blocks to [2, 2, 4, 6, 4, 2, 2];

The narrow DenseNet, with the depth of each DenseNet block the same as the basic configuration, and the bottleneck size and growth rate reduced to half.

It is shown that reducing the depth or width of the DenseNet blocks will both hurt the model's performance. Since adding more layers or depth will increase the computational burden, we suggest the use of the "standard" configuration for future adoption of the model.

1.5 Pooling and unpooling operations in graph U-net

U-nets[S4] have enjoyed great success in image-related pixel-wise prediction tasks. It can achieve a high receptive field size with fewer layers, while significantly reduced memory

consumption during training. The major problem for applying U-net in graph generation is that, unlike images and 3D voxels, there are no canonical ways to perform pooling and unpooling on graphs.[S12] In order to perform pooling and unpooling on molecular graphs, we designed a custom clustering scheme:

- In the first level of clustering, atoms with one valence, such as hydrogens, halogen, and oxygens in carbonyl groups, are collapsed to their neighbor atoms. For most molecules, almost half of their atoms are hydrogen, consuming a significant amount of GPU memory. This level of clustering enables us to include hydrogens into the generation process in an efficient way, by compressing the information of hydrogens into its neighboring heavy atoms.

- In the second level of clustering, molecules are fragmented into ring assemblies and chains. This method is previously used to define molecule scaffold[S13] and to organize scaffold datasets.[S14] After fragmentation, atoms in the same ring assembly or chain are clustered together.

- In the final level of clustering, all nodes are collapsed into a single graph-level master node.

A visual demonstration of this scheme is given in Fig. S5. After the clustering method is defined, the pooling and unpooling operations can subsequently be defined, as shown in Fig. S3**c,d**.

1.6 The policy network

After creating a continuous representation of the current state G_i using the state encoder $h_i = (\mathbf{h}_{i,g}, \{\mathbf{h}_{i,v}\}_{v \in V_i}) = f_\theta(G_i)$, the policy network is used to decide what action should be carried out. Recall that there are three types of decision the policy network need to make:

- The type and position of new atoms during the "append" operation;

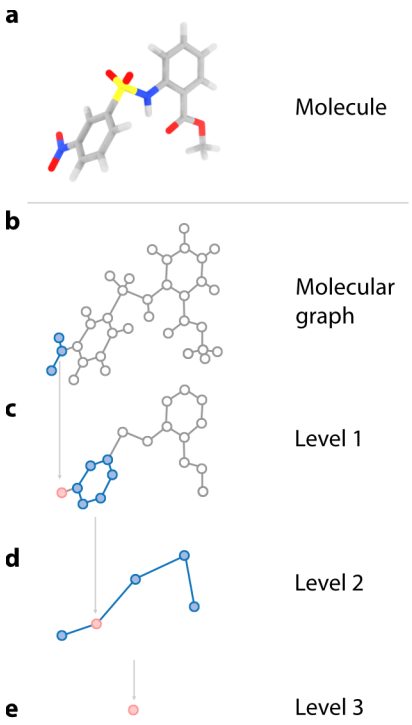

Figure S5: A custom three-level node clustering scheme for pooling and unpooling operations in molecular graphs.

- The atom to be connected and the type of connecting bond during the "connect" operation;
- The rank of the new atoms to be added.

We denote the policies for each decision as p_θ^{append}, $p_\theta^{\text{connect}}$ and p_θ^{rank}.

1.6.1 Decision making during the "append" operation

During the "append" operation, one or more atoms are created and added to the focused atom v'. We represent a newly created atom as the tuple $v^* = (a, b, \mathbf{x})$, where a is the atom type, b is the bond type used to connect the new atom with the focused atom, $\mathbf{x} = (r, \theta, \phi)$ is the spherical coordinate of this new atom in the local coordinate system (described in Section S1.3). The policy network for the append action can be written as:

$$p_\theta(v_1^*, ..., v_m^* | G_i) = p_\theta(a_1, b_1, r_1, \theta_1, \phi_1, ..., a_m, b_m, r_m, \theta_m, \phi_m | G_i)$$

Where m is the number of new atoms to add. Compared to most previous autoregressive models for 3D molecules,[S1,S15] our proposed method generates all atoms connected to v^* in a single iteration (Fig. S6).

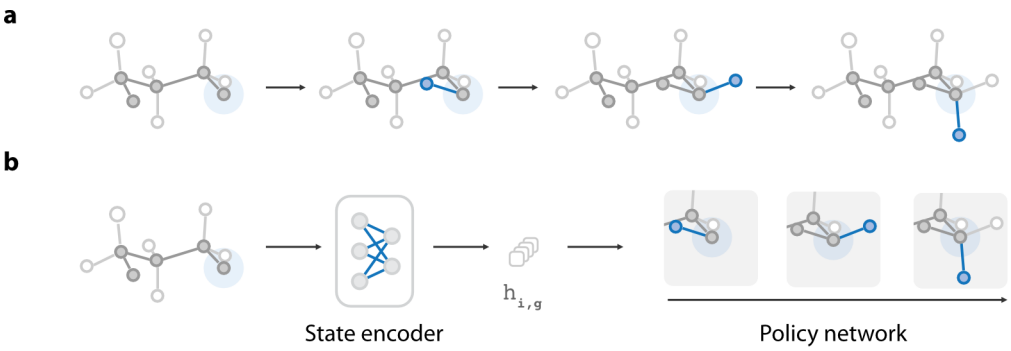

Figure S6: Different from most autoregressive models in 3D molecule generation (**a**), our proposed method generates all atoms connected to the focused atom as a group (**b**).

We incorporate several prior knowledge about drug-like molecules. First, most atoms in organic drug-like molecules have a valence less or equal to 4. That is, at most three atoms can be added to a focused atom at one time. When less than three new atoms are added, we add "null" atoms to fill the blanks. This makes the problem simpler since we are now dealing with a fixed number of random variables. Another prior knowledge we incorporate is that there are only three types of allowed local geometry for most drug-like molecules: sp (linear), sp2 (planar), and sp3 (tetrahedral). We ask the model to first generate the type of the local geometry (h), and then the position of new atoms. Empirically, we find that this can help the model to better learn the local geometry. Now, we still need to find a way to factorize $p_\theta(v_1^*, v_2^*, v_3^*|h, G_i)$, which contains $(2 + 3) * 3 = 15$ random variables. A natural choice is to factorize it into an autoregressive model:

$$p(v_1^*, v_2^*, v_3^*) = p(v_3^*|v_1^*, v_2^*)p(v_2^*|v_1^*)p(v_1^*)$$

$$p(v_i^*|\cdot) = p(\phi_i|a_i, b_i, r_i, \theta_i, \cdot)p(\theta_i|a_i, b_i, r_i, \cdot)p(r_i|a_i, b_i, \cdot)p(a_i, b_i, \cdot)$$

The conditions G_i and h are omitted for simplicity. $p_\theta(a_i, b_i|\cdot)$ is a categorical distribution of atom and bond types. We apply prior knowledge about the allowed valence for the focused atom to create a mask for $p_\theta(a_i, b_i|\cdot)$ at each step so that it will not violate the valence constraint. $p_\theta(r_i|\cdot)$, $p_\theta(\theta_i|\cdot)$, $p_\theta(\phi_i|\cdot)$ are mixtures of truncated Gaussian distributions. The number of mixture is set to be 15, 10 and 5 for v_1^*, v_2^*, v_3^* respectively. The range of r_i and θ_i are set to $[0.5, 2.5]$ and $[0, \pi]$. The parameters of those distribution are calculated using MADE (masked autoencoder for distribution estimation[S16]), which is an efficient architecture for autoregressive models based on masked linear layers. The architecture for MADE is demonstrated in Fig. S7.

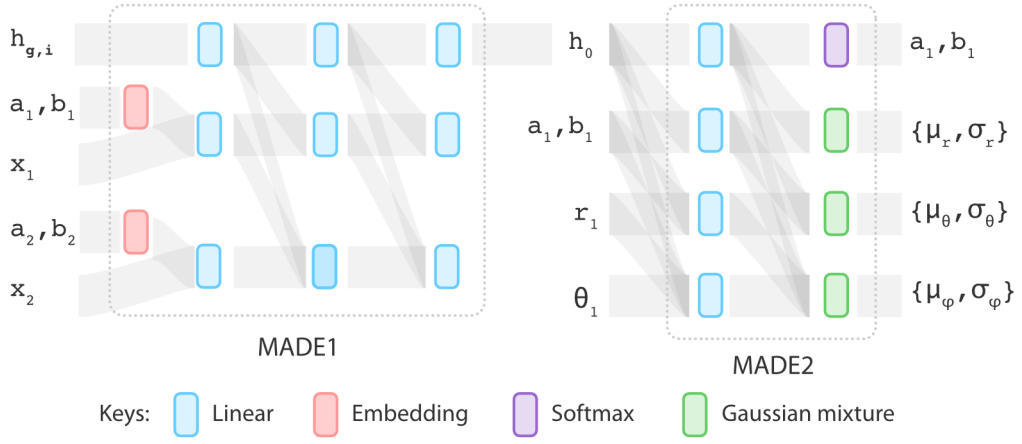

Figure S7: The architecture of MADE blocks. Output sizes of linear layers in the MADE1 block are 128, output sizes of linear layers in the MADE2 block (except that used in the Gaussian mixture layer) are 64. The embedding table used here is the same as that used in the state encoder. The activation function used for outputting standard deviation of r, θ and ϕ is softplus.

In practice, the target distribution of atom position usually resides in a low dimensional submanifold of $\mathbb{R}^3$ (Fig. S8) and using MADE to directly fit those distributions will not work well. We adopt the method from a previous work[S17] and adds a small error to the target distribution so that it can be fitted more robustly by MADE. We call this modified model SoftMADE. Different from the original paper, which samples the noise level from a uniform distribution: $c \sim u[a, b]$, we sample the noise level in two steps:

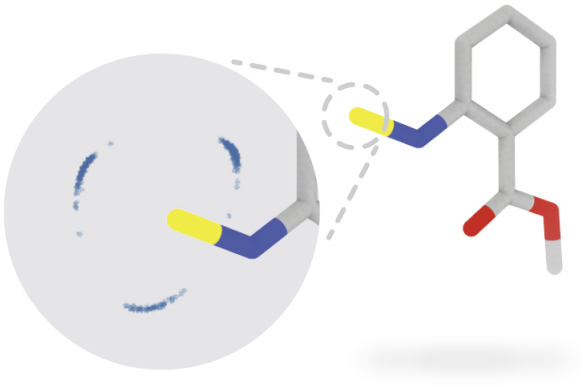

Figure S8: The problem of low dimensionality. This figure shows the new atoms sampled from the model mostly reside inside a 1D ring above the focused atom.

$$c_0 \sim u[0,1]$$

$$c = c_{\max} c_0^{\alpha}$$

Where $c_{\max}$ is the maximum level of noise, and $\alpha > 1$ is the parameter controlling the shape distribution. Three conditons are tested: (1) ordinary MADE; (2) SoftMADE with $c_{\max} = 0.2, \alpha = 3$; (3) SoftMADE with $c_{\max} = 0.4, \alpha = 4$. It is find that biger $c_{\max}$ combined with higher α (the third conditon) yields better result.

Finally, since the new atoms v_1^*, v_2^*, v_3^* can be generated in any order, we use the following corrected likelihood during training:

$$p(\{v_1^*, v_2^*, v_3^*\}) = 1/3! \sum_{\sigma} p(v_{\sigma_1}^*, v_{\sigma_2}^*, v_{\sigma_3}^*)$$

1.6.2 Decision making during the "connect" operation

For each possible action in the "connect" operation, we first compute their unnormalized scores as follows:

$$\hat{\mathbf{p}}_v^{\text{connect}} = \text{MLP}_{\text{policy}}^{\text{connect}}(\mathbf{h}_{i,v})$$

$$\hat{p}_v^{\text{skip}} = \text{MLP}_{\text{policy}}^{\text{skip}}(\mathbf{h}_{i,g})$$

Where $\text{MLP}_{\text{policy}}^{\text{connect}}$ and $\text{MLP}_{\text{policy}}^{\text{skip}}$ are fully connected layers. Those scores are then normalized using softmax:

$$[\mathbf{p}_v^{\text{connect}}; p_v^{\text{skip}}] = \text{softmax}([\hat{\mathbf{p}}_v^{\text{connect}}; \hat{p}_v^{\text{skip}}])$$

The the values $\mathbf{p}_v^{\text{connect}}[b]$ in vector $\mathbf{p}_v^{\text{connect}}$ represents the probability of connecting the focused atom v' with v using a new bond of type b. The value p_v^{skip} represents the prabability of skipping the "connect" operation and proceeds directly to the "append" operation.

1.6.3 Ranking the generated atoms

When ranking the generated atoms, we first calculate an unnormalized score for each permutation of the new atoms:

$$\hat{s}_\sigma = \text{MLP}_{\text{policy}}^{\text{rank}}([v_{\sigma_1}^*; v_{\sigma_2}^*; v_{\sigma_3}^*])$$

And then the normalized probability:

$$p(\sigma) = \frac{\exp \hat{s}_\sigma}{\sum_{\sigma'} \exp \hat{s}_{\sigma'}}$$

The ranking is then sampled from $p(\sigma)$.

1.7 Data collection and preprocessing

We construct a drug-like subset of ChEMBL[S3] for the training and evaluation of the model. The topological data of all molecules are downloaded from ChEMBL (version 27) and is then

filtered using a series of criteria:

- Molecules with atom type outside the set {C, H, O, N, P, S, F, Cl, Br, I} as well as those that do not contain carbon atoms are removed;
- Molecules with the number of heavy atoms outside the range [10, 35] are removed;
- Molecules with a QED[S18] value less than 0.5 are removed;
- Molecules with ring sizes larger than 7 are removed. Rings in the molecule are extracted using RDKit;
- Molecules containing a ring assemble with the number of SSSR(smallest set of smallest rings) greater than 4 are removed.

After filtering the topological structure, 3D structures are generated for each molecule using RDKit. The initial 3D embeddings of molecules are first created using distance geometry and then optimized using the MMFF94s forcefield. After those processings, we obtain a dataset with 1 million small molecules, with one conformer for each. The dataset is randomly split into the training set (4/6), validation set (1/6), and test set (1/6). The validation set is used during the manual hyperparameter tunning

Note that L-Net only supports generating molecules with atom types inside the set {C, H, O, N, P, S, F, Cl, Br, I}, since molecules containing elements outside this set are removed from the training set. This set is larger than the set supported by the previous 3D generative model G-SchNet, which only contains {C, H, O, N, F}. Also, this set covers most drug molecules, which makes it sufficient for most tasks in *de novo* drug discovery. L-Net cannot generalize to new atom types outside the training set, since it uses a finite lookup table for the embedding of atom types, and only supports types pre-defined inside the table. However, in cases where other atom types need to be supported, the users can always fine-tune L-Net using a new dataset.

Also note that in theory, more conformers can be generated for each molecule during data processing, but we believe that one conformer for each molecule will be sufficient for training using the drug-like subset of ChEMBL. Using fewer conformers for each molecule will not induce bias during training and testing, since statistical variables used in this work, such as the gradient estimator used for training and the MMD estimator for testing, is statistically unbiased no matter how many conformers are used. An advantage of using more conformers is that it can be used for data augmentation, which is useful for cases when the size of training data is limited and the risk of over-training is high. However, our task does not fell into this category, with 1 million unique molecules for training, validation, and testing. Additionally, metrics such as %unique, internal diversity does not show signs of over-training (as discussed in the Result section). Overall, we believe that a single conformer is sufficient for fitting the model using the ChEMBL dataset, but for future works that need to fine-tune the model on a smaller dataset, data augmentation using multiple conformers might be necessary.

As mentioned in previous sections, the model generates molecules in a step-by-step manner. To train the model, we need to create an "expert trajectory" for generating each molecule G in the dataset (Fig. S9**a**):

$$(G_0, G_1, ..., G_n), \text{where } G_1 = \emptyset, G_n = G$$

The model is then trained to imitate this path by maximizing the log-likelihood:

$$\log p_\theta(G) \approx \sum_{i=1}^{n} \log p_\theta(G_i | G_0, .., G_i)$$

We use a method similar to that used previously[S11,S19] to generate those "expert trajectories". Briefly, the atoms in the molecule are first ranked using a canonical ranking algorithm in RDKit, and depth-first traversal is performed to produce a path $(G_0, G_1, ..., G_n)$. Although this method works well for 2D generative models, it does not produce good results on our 3D generative model. To improve the performance, we modify the depth-first traversal algorithm

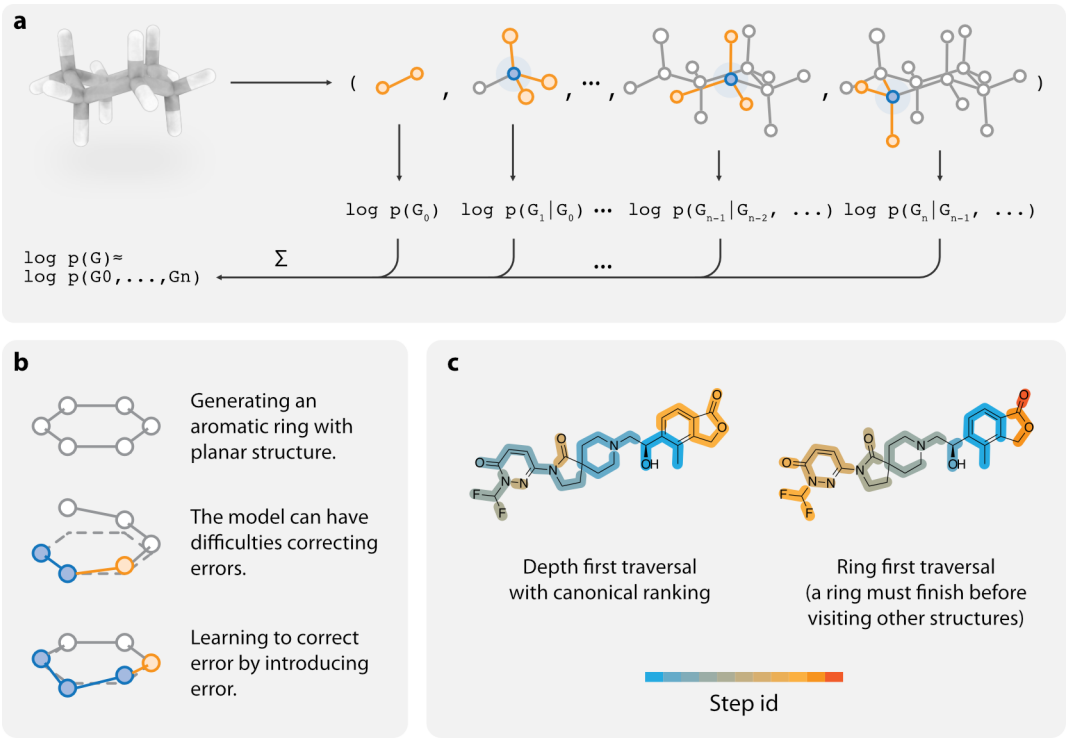

Figure S9: Data preprocessing and tricks used to improve model performance. **a.** For each data in the training set, we create an "expert trajectory" for generating this molecule, and train the model to imitate this trajectory. **b.** We find that the model may suffer from the problem of distribution mismatch, so random errors are manually added to the training data so that the model can learn to correct them. **c.** The image shows the order each atom is traversed for ordinary (left) and ring first (right) traversal scheme (blue atoms are traversed first, red atoms are traversed last). We use ring-first trajectory to train the model to close the ring first before generating other structures, which have a similar effect as treating rings as basic generation units.

to prioritize the closure of rings (see Fig. S9**c**). We find that this method could significantly improve the quality of generated samples.

It is also reported that randomized trajectories help the model to achieve better performance.[S19,S20] In this work, we randomize the trajectory by randomizing the starting position of the depth-first traversal. We also include data of the model trained using non-randomized trajectories for comparison.

One of the major issues related to imitation learning is data distribution mismatch. Specifically, the model only sees correct "expert trajectories" during training, and if a mistake happens during generate, the model may not know how to recover from that error, and eventually produce invalid results (see Fig. S9**b** for a simple example). Our solution is to "simulate" those errors by adding Gaussian noise to the input by a certain probability. We use a noise with the standard deviation of 0.1 $\mathring{A}$ and experimented with the probability of 0.1 and 0.5. It shows that a 0.5 probability significantly improves the model performance compared with the 0.1 level.

1.8 Model training

The model is implemented using PyTorch.[S21] Adam[S22] is used to optimize the model parameter. Parameters (β_1, β_2) are set to be the default value provided by PyTorch. The learning rate is initialized to be 10^{-3}, and is decreased by 0.01 for a certain amount of step. Several decay frequencies are experimented with: every 50 steps, every 100 steps, or every 200 steps. The batch size for training is set to be 128, and are trained for a total of 10 epochs. This takes around 3 days to finish. Training is performed on a single NVIDIA TITAN Xp graphics card.

1.9 Hyperparameters

As can be seen from previous sections, the model proposed here contains a large number of hyperparameters, including those for model architecture, data generation, and model

training. Considering the long training time, comprehensive optimization of hyperparameter is difficult. The hyperparameter selection is further complicated by the fact that there are a variety of metrics that can be used to evaluate the model (see Section S1.11). The best-performing model on one metric does not necessarily perform the best on the other. In this work, we target 3D MMD as the objective and perform manual hyperparameter tunning to get to the performance level that is acceptable for general usage. The result hyperparameter setting is referred to as the "standard" configuration and is what we suggest to use in future research using this model. We do note that the optimization process is not comprehensive and we expect that better performance can be obtained using more dedicated hyperparameter optimization techniques.

To understand how model performance is affected by a set of hyperparameter of interest, we perturb those parameters from the standard configuration to investigate its effect. The set of hyperparameters that are analyzed are (also summarized in Table S1):

- The depth and width of DenseNet (see Section S1.4);
- The parameters for SoftMADE (see Section S1.6.1);
- The noise added to coordinates of the input structure (see Section S1.7);
- Whether the "expert trajectories" used for training are randomized (see Section S1.7).
- The speed of learning rate decay (see Section S1.8).

Table S1: A summary of different hyperparameter configurations whose performance is reported in this work.

Methods	Randomized trajectory	c	alpha	Input noise	DenseNet architecture	Learning rate decay
Non-random initialization	No	0.4	4	0.5	Basic	100 step
SoftMADE (low noise)	Yes	0.2	3	0.5	Basic	100 step
No SoftMADE	Yes	0	0	0.5	Basic	100 step

Methods	Randomized trajectory	c	alpha	Input noise	DenseNet architecture	Learning rate decay
Low input noise	Yes	0.4	4	0.1	Basic	100 step
Shallow DenseNet	Yes	0.4	4	0.5	Shallow	100 step
Narrow DenseNet	Yes	0.4	4	0.5	Narrow	100 step
Slow lr decay	Yes	0.4	4	0.5	Basic	200 step
Fast lr decay	Yes	0.4	4	0.5	Basic	50 step
Standard configuration	Yes	0.4	4	0.5	Basic	100 step

1.10 Optimizing the speed of molecule generation

We used several techniques to accelerate the process of molecule generation. First, many CPU side operations in this model cannot be implemented efficiently using native python, and we use Numba,[S23] a just-in-time compiler for python, to speed up those codes. The performance benefit is significant, and in our case, the speedup can be more than 10 times.

Secondly, we use multiprocessing to hide CPU processing latency from GPU. During molecule generation, we need to move back and forth between GPU and CPU for action sampling and graph processing. When a large number of molecules are being generated at the same time, the graph processing time in the CPU can be significant. Since GPU operation is blocked by CPU tasks, there will be noticeable performance degradation. Our solution is to place GPU and CPU in different processes and split the molecules being generated into two batches. During CPU processing, GPU can work on the other batch of molecules, thereby increasing the GPU utilization (Fig. S10**a**).

Thirdly, we develop a method to adaptively balancing the number of molecules generated at each time. During molecule generation, as more nodes are added to the molecule, the computational cost for the neural network to process the intermediate structure will be gradually growing. This will cause significant performance degradation. At the beginning of generation, the computation burden is low, and GPU is largely underutilized. When

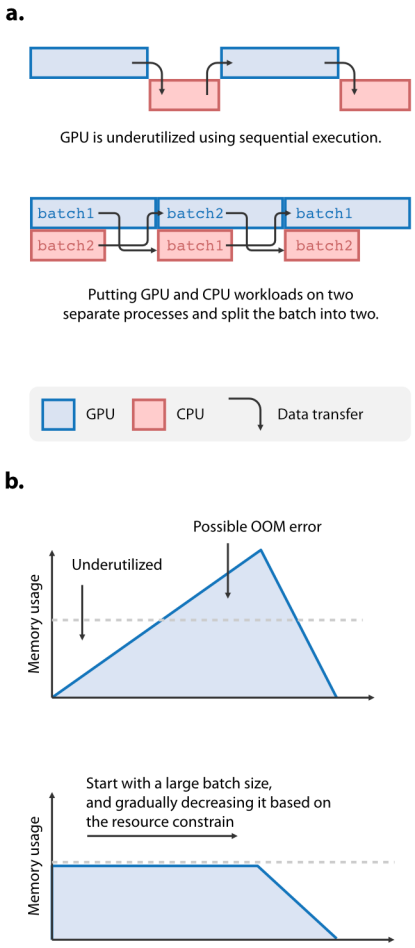

Figure S10: Tricks used to accelerate molecule generation: **a.** CPU and GPU workloads are separated into two different processes and can be executed asynchronously by each working on a different batch of data (batch1 and batch2) at the same time. **b.** Batch size is adjusted dynamically to ensure that GPU is fully utilized and does not exceed the resource constrain.

approaching the end of the generation, the computation cost is significantly higher, and can cause out of memory (OOM) error (Fig. S10**b**). We created a scheduling method to dynamically adjust the batch size of the generation task based on the maximum capacity of the GPU. Empirically this has resulted in significant speed improvement for our model.

After performing those operations, we can achieve a generation speed of 0.008 seconds per molecule in an NVIDIA TITAN Xp card. Note that this speed is slower than SMILES-based samplers, largely because of the complexity of the generative workflow and the network architecture. But we also note that there are still spaces where the performance can be further improved, like performing quantization or pruning, which will be explored in the future.

1.11 Evaluation

Several benchmarks have been developed for evaluating 2D generative models, such as MOSES[S24] and GaucaMol,[S25] but this type of work is still missing for 3D models. Here, we assemble a set of evaluation metrics for assessing the performance of 3D generative models. Emphasis is placed on measuring the quality of molecule conformations, by investigating various 3D molecular features. All the following metrics are calculated using 50,000 generated molecules.

Output validity and uniqueness

The two metrics measure the percentage of output molecules that are chemically valid and structurally unique. The validity is measured by calculating the percentage of generated molecules that pass the RDKit sanitization check (%valid). Although this metric measures the quality of topological structure, we find that in practice it is also a good proxy for the quality of 3D structures. This is because that many serious 3D errors occurring during the generation will eventually lead to invalid topological structures. The uniqueness is measured by calculating the percentage of unique structures among outputs (%uniq). This can be used to detect whether the model has been overtrained or has collapsed to a single mode.

Distribution of molecular properties

Investigating the distribution of molecular properties is a good way to intuitively assess the quality of generated samples. The mean and standard deviation of each property is reported, as well as the visualization using kernel density estimation. The properties investigated include regular topological features (molecular weight, LogP, the number of hydrogen donors and acceptors, the number of rotatable bonds, QED) as well the following 3D features:

- Normalized PMI ratios (NPRs):[S26] This is a shape descriptor composed of two components $(NPR1, NPR2) = (I_1/I_3, I_2/I_3)$, where I_1, I_2, I_3 are principal moments of inertia sorted by ascending magnitude. The point $(1, 1)$, $(0.5, 0.5)$, $(0, 1)$ corresponds to the archetypes of sphere, disk and rod, giving this descriptor high interpretability. NPRs are calculated using the implementation in RDKit.

- Solvent accessible surface areas (SASA): SASA is an important molecular descriptor measuring the contact area between the molecule and the solvent. We report the distribution of polar and total SASA calculated using the package FreeSASA.[S27]

Maximum mean discrepancy

We use maximum mean discrepancy (MMD) to give a quantitative measurement of the difference between the distribution of generated and real samples. Given a kernel function $\kappa(\cdot, \cdot)$, the MMD between two distributions can be estimated as :

$$MMD = \frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{j=1, j\neq i}^{N} \kappa(x_i, x_j) + \frac{1}{M(M-1)} \sum_{i=1}^{M} \sum_{j=1, j\neq i}^{M} \kappa(y_i, y_j) - \frac{2}{MN} \sum_{i=1}^{N} \sum_{j=1}^{M} \kappa(x_i, y_j)$$

Where $\{x_i\}_{i=1}^{N}$ are sampled from the real distribution and $\{y_i\}_{i=1}^{M}$ sampled from the generative model. This method has been previously used to assess the performance of 2D molecular generative models,[S11] and can be easily applied to 3D models by changing κ to measure 3D molecular similarity.

In this work, we use two types of kernel functions to calculate MMD: (1) the Tanimoto similarity of 2D Morgan fingerprint (1024 bit, radius of 2) and (2) the Manhattan distance of USRCAT fingerprint.[S28] The two MMD measures the topological and 3D discrepancy between generated and real samples. Note that for the 3D kernel function, a more accurate choice might be using the shape-based alignment method.[S29] But since the computational complexity of MMD calculation is $O(\max(M, N)^2)$, and shape alignment involves optimization for each molecule pair, this will not be feasible for our task. USRCAT is an alignment-free method with precalculated fingerprint and is more suitable for MMD calculation.

The calculation of MMD is parallelized in GPU using Cupy.[S30] For topological fingerprint, we store the 1024 bit fingerprint into 32 uint32 integers and utilize bitwise operations to speed up the calculation of the Tanimoto coefficient.

Precision and recall

Although MMD can reliably quantify the discrepancy between distributions, its value is difficult to interpret. Ideally, we want the information about:

- What percentage of generated samples are realistic;
- What percentage of real data distribution can be covered by the generative model;

We call the two metrics precision and recall for the generative model. Precision can be used to measure the sample quality, and recall can be used to assess mode coverage. Since they are both percentage values, they are more interpretable than MMD.

The definition of the two metrics follows the previous work.[S31] First of all, we define the space covered by a probability distribution from its samples $X = \{x_i\}_{i=1}^N$ as:

$$\Phi(X) = \{x | \exists\ x_i \in X \text{ s.t. } d(x - x_i) \leq d(N_k(x_i, X) - x_i)\}$$

Where $N_k(x_i, X)$ denotes the k-th nearest neighbor of x_i in the dataset X, and $d(\cdot, \cdot)$ measures the distance between two data point. Given the real data $X = \{x_i\}_{i=1}^N$ and generated

data $Y = \{y_i\}_{i=1}^{M}$, precision and recall are defined as:

$$P = \frac{|Y \cap \Phi(X)|}{|Y|}$$

$$R = \frac{|X \cap \Phi(Y)|}{|X|}$$

Similar to MMD, a 2D and 3D version of precision and recall can be calculated using Morgan fingerprint and USRCAT. The value of k is set to be 3.

Validity of local geometries

We check the correctness of local 3D structures in generated molecules by examining the distributions of bond lengths and bond angles. More specifically, we group the bond lengths and angles by its environment key, calculate the mean and standard deviation within each group, and compare them between generated and test set molecules. The environment key for the bond length contains the bond type and the type of its two adjacent atoms: (a_u, a_v, b_{uv}), while that for a bond angle contains the type and hybridization state of the central atom: (h_u, a_u). Groups containing less than 1000 data points are removed from the evaluation.

We also check the distribution of torsion angles in generated and test set molecules. Quads of atoms a-b-c-d are matched using the SMARTS pattern provided in the previous work[S32] and torsion angles are calculated from the matched coordinates. Patterns that give less than 1000 matches are discarded. MMD values of the torsion distribution in each pattern are computed.

The above evaluation aims at giving a qualitative look at the quality of local geometry, and the comparison is only performed on the model trained with standard configuration. We use RMSD (Root Mean Standard Deviation) for a more quantitative evaluation, as described below.

RMSD

For each generated molecule, we optimize the conformation using MMFF94s force field and calculate the RMSD of heavy atoms between the original and optimized structure.

To give a context on how the model performs in this metric, we perform the same calculation on the ETKDG method,[S32] which is a conformation generation method for small molecules that aims to provide a faster alternative for forcefield-based minimization. For each molecule in the test set, we generate an initial conformation using ETKDG, optimize it using MMFF94s, and calculate the RMSD for conformation before and after optimization. The average RMSD is then compared with that of the generative model.

Note however that since ETKDG and deep generative model are developed to solve two completely different problems, the comparison of RMSD can not tell which method is better or worse. Nonetheless, this comparison should give us an idea about the overall level of quality of the generated conformations.

Training on QM9 to enable comparison with G-SchNet

We train L-Net on a subset of QM9 to enable comparison with G-SchNet. Recall that L-Net contains certain assumptions about the structure it can generate. Specifically, it only generates molecules with a maximum valence of 4 and recognizes three hybridization types: sp, sp2, and sp3. To train on QM9, molecules in QM9 that do not satisfy those criteria are not considered. Many of those molecules can not be processed by RDKit under default conditions. The majority of molecules pass those filters, and a random subset of 50,000 molecules (at the same scale as G-SchNet) are constructed for training L-Net. The hyperparameter for training is similar to that used for the ChEMBL dataset (the standard configuration mentioned in Section 1.9), except that the learning rate decay is reduced to half and the number of training epoch is increased to 40.

The percentage of valid outputs and median RMSD values after optimization is compared between L-Net and G-SchNet. Note that G-SchNet uses OpenBabel for validity check, which might be slightly different from RDKit, but the overall result should be similar. For RMSD, Gebauer et al. have reported RMSD values calculated at the PBE/def2-SVP level of theory using molecules generated from G-SchNet under various sampling temperatures (T=2, 1, 0.1, 0.01, 0.001). Since L-Net does not use temperature sampling, the value corresponding

to T=1 is used for comparison. 1,000 molecules are sampled from L-Net and optimization is performed at the PBE/def2-SVP level of theory using ORCA[S33] to calculate RMSD, similar to what was done by Gebauer et al.

We note that it is more desirable to make the comparison using the drug-like ChEMBL dataset. However, a variety of technical issues need to be resolved in order to migrate G-SchNet to drug-like molecules, which is not the major focus of this work. We do acknowledge that more work is needed to create a better benchmark for 3D generative models for drug-like molecules. We believe that the evaluation module developed in this work should provide a good starting point for future advancements toward this end.

1.12 Combining L-Net with MCTS for structure-based molecule design

The model proposed in this work can be conveniently combined with other techniques such as reinforcement learning to achieve molecular design based on a given objective. As a proof of concept, we combine L-Net with Monte Carlo tree search (MCTS) and test its ability in the problem of structure-based molecule design. Previous works have combined MCTS with 2D generative models in the object-directed design of molecules,[S34] but to our knowledge, the combination of MCTS with 3D generative models has not yet been reported.

MCTS finds optimal solutions given a reward function by iteratively construct a search tree. At each iteration, it performs four operations to build the tree structure: selection, expansion, simulation, and backpropagation.

- During the selection step, a promising leaf node is chosen by traversing the tree from the root. E2W (Empirical Exponential Weight[S35]) is used in this work as the policy for tree traversal. We periodically perform an "exploit" step by pruning the search tree. Specifically, we sample a node at depth i using the Q-values of each node and delete all other nodes whose ancestors or descendants do not contain the sampled node. The

depth i starts from 0 and increments at each exploit step. The MTCS ends if the exploit operation reaches the termination node.

- At the expansion step, child nodes are added to the selected leaf by enumerating actions given the state, and one of the children is selected for the subsequent simulation step. Note that for 3D generative models, the action space is continuous during the atom placement step, but commonly used MCTS algorithms only support discrete action space. Therefore, a clustering operation is performed to discretize the actions space to enable the combination of the model with MCTS.

- Several rollouts are performed during the simulation step using L-Net, and the docking scores are calculated using Smina.[S36] Note that our model directly grows the ligand inside the binding pocket, so it only requires local optimization to obtain the docking score, making the reward evaluation process extremely fast. This serves as a major advantage compared with previous 2D-based MCTS methods, which requires a lengthy global conformation search to embed 2D molecular structures into the 3D pocket. Also note that we do not explicitly include drug-likeness and synthetical accessibility in the reward function, as those properties are implicitly captured by the L-Net trained from the drug-like ChEMBL subset. However, we do include a term in the reward function to penalize complexed ring systems (any ring assemble with the number of SSSA (smallest set of smallest rings) larger than two).

- After the rewards are collected, the model backtracks to root and updates the Q values along the path. Softmax backup[S35] is used as the update method. The initial Q value of each node is set to be the log-probability value given by L-Net.

To better utilize the computational power of GPU, root-level, leaf-level, and tree-level parallelization are introduced to MCTS, similar to that done by Chaslot et al.[S37] For root level parallelization, we are able to perform three MCTS runs simultaneously on a single

TITAN Xp GPU. An overall of 36 independent runs is performed in order to diversify the result.

2 Supplementary Tables

Table S2: The performance of LNet, measured in terms of %valid and %uniq, with different hyperparameters. The star indicates the best performing hyperparameter selection.. The standard configuration is the hyperparamter selection with best 3D MMD performance.

Method	%Valid	%Uniq
Randomized trajectory	93.4%	98.7%
SoftMADE (low noise)	92.5%	99.0%
No SoftMADE	89.8%	89.7%
Low input noise	88.2%	97.4%
Shallow DenseNet	90.5%	98.9%
Narrow DenseNet	90.7%	98.6%
Slow decay	94.3% (*)	98.2%
Fast decay	87.2%	99.0%
Standard configuration	93.5%	99.1% (*)

Table S3: Distribution of 2D molecular properties among generated molecules using different hyperparameters. The "standard configuration" is the hyperparamter selection with best 3D MMD performance.

	MW		LogP		HBA		HBD		ROTB		QED	
Methods	mean	std.	mean	std.	mean	std.	mean	std.	mean	std.	mean	std.
Deterministic trajectory	328.5	86.8	2.89	1.45	4.00	1.67	1.25	0.95	4.26	2.25	0.683	0.148
SoftMADE (low noise)	332.0	87.6	2.87	1.47	4.06	1.71	1.29	0.99	4.18	2.18	0.675	0.152
No SoftMADE	329.0	94.3	2.86	1.55	4.07	1.80	1.20	0.98	4.20	2.28	0.664	0.156
Low input noise	356.4	119.1	2.66	1.68	4.72	2.11	1.47	1.10	4.76	2.79	0.588	0.182
Shallow DenseNet	338.9	95.3	2.84	1.55	4.21	1.78	1.31	1.01	4.29	2.30	0.662	0.158
Narrow DenseNet	329.0	93.3	2.74	1.52	4.17	1.79	1.20	0.98	4.22	2.26	0.669	0.154
Slow decay	332.8	90.4	2.78	1.48	4.21	1.75	1.35	1.02	4.51	2.35	0.664	0.152
Fast decay	339.5	94.3	2.88	1.57	4.17	1.78	1.26	1.01	4.54	2.40	0.655	0.162
Standard configuration	338.0	91.7	2.91	1.55	4.12	1.75	1.26	0.99	4.40	2.28	0.665	0.156
Validation	345.0	69.7	3.05	1.30	4.13	1.55	1.19	0.91	4.30	2.04	0.700	0.117

	MW		LogP		HBA		HBD		ROTB		QED	
Test	344.8	69.7	3.05	1.29	4.13	1.56	1.20	0.90	4.30	2.04	0.701	0.116

Table S4: Distribution of 3D molecular properties among generated molecules using different hyperparameters. The "standard configuration" is the hyperparamter selection with best 3D MMD performance.

	Total SASA		Polar SASA		NPR1		NPR2	
Methods	mean	std.	mean	std.	mean	std.	mean	std.
Deterministic trajectory	514.9	100.1	126.2	57.6	0.252	0.130	0.859	0.092
SoftMADE (low noise)	518.6	100.8	129.4	58.3	0.252	0.130	0.859	0.093
No SoftMADE	514.6	107.5	128.1	60.3	0.261	0.131	0.854	0.094
Low input noise	540.3	135.9	148.8	66.9	0.278	0.134	0.848	0.095
Shallow DenseNet	524.7	107.6	133.3	59.5	0.263	0.134	0.856	0.093
Narrow DenseNet	514.3	106.5	128.8	59.7	0.265	0.132	0.854	0.094
Slow decay	522.2	104.5	132.1	58.7	0.256	0.132	0.858	0.093
Fast decay	525.3	106.9	129.2	59.6	0.271	0.136	0.856	0.093
Standard configuration	565.9	107.9	129.0	60.2	0.263	0.133	0.857	0.093
Validation	579.5	88.5	129.5	55.9	0.232	0.125	0.869	0.093
Test	537.8	84.0	128.6	53.5	0.232	0.124	0.869	0.093

Table S5: The result of topological and 3D MMD, sample diversity, precision and recall for each combination of hyperparameters. The "standard configuration" is the hyperparamter selection with best 3D MMD performance.

	3D				Topological			
Methods	MMD	Diversity	Precision	Recall	MMD	Diversity	Precision	Recall
Deterministic trajectory	0.00182	0.154	83.6% (*)	87.8%	0.00195	0.160 (*)	51.4% (*)	71.8%
SoftMADE (low noise)	0.00143	0.155 (*)	83.1%	88.2%	0.00148	0.158	46.6%	74.8%
No SoftMADE	0.00221	0.131	81.8%	88.3%	0.00699	0.158	39.2%	76.8%
Low input noise	0.00370	0.143	76.0%	88.6%	0.00391	0.148	28.8%	84.1% (*)
Shallow DenseNet	0.00156	0.151	81.5%	88.2%	0.00101 (*)	0.156	41.8%	76.8%
Narrow DenseNet	0.00253	0.148	81.7%	88.4%	0.00132	0.157	41.6%	76.6%

	3D				Topological			
Slow decay	0.00147	0.151	81.9%	88.3%	0.00159	0.155	44.9%	76.5%
Fast decay	0.00196	0.150	80.9%	88.6% (*)	0.00149	0.156	38.7%	79.0%
Standard configuration	0.00134 (*)	0.152	81.9%	88.3%	0.00115	0.157	43.1%	78.0%
Validation	-	0.157	-	-	-	0.157	-	-
Test	-	0.157	-	-	-	0.157	-	-

Table S6: The RMSD values of generated conformers before and after optimization.

	RMSD	
Methods	mean	std.
Deterministic trajectory	0.613	0.502
SoftMADE (low noise)	0.632	0.508
No SoftMADE	0.715	0.549
Low input noise	0.779	0.597
Shallow DenseNet	0.687	0.533
Narrow DenseNet	0.689	0.522
Slow decay	0.623	0.503
Fast decay	0.735	0.540
Standard configuration	0.663	0.521
Validation	0.838	0.539
Test	0.807	0.524

3 Supplementary Fig.s

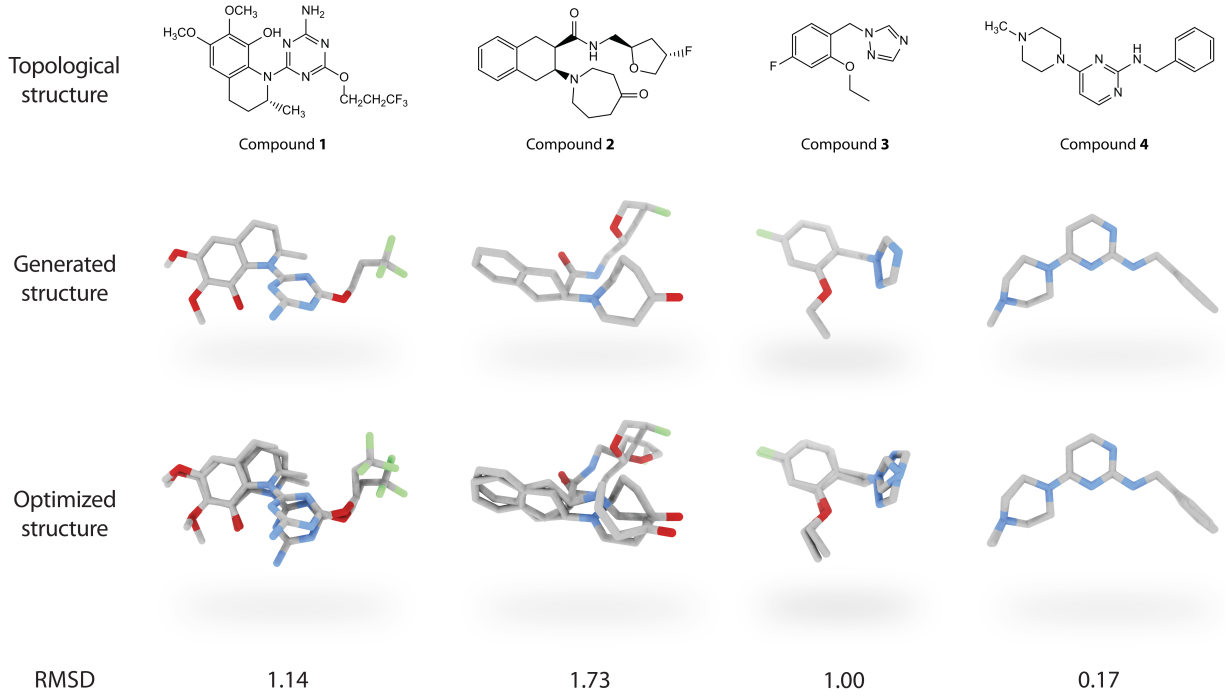

Figure S11: The topological structures, generated conformation, optimized conformation, and RMSD value (Å) for several generated molecules using L-Net.

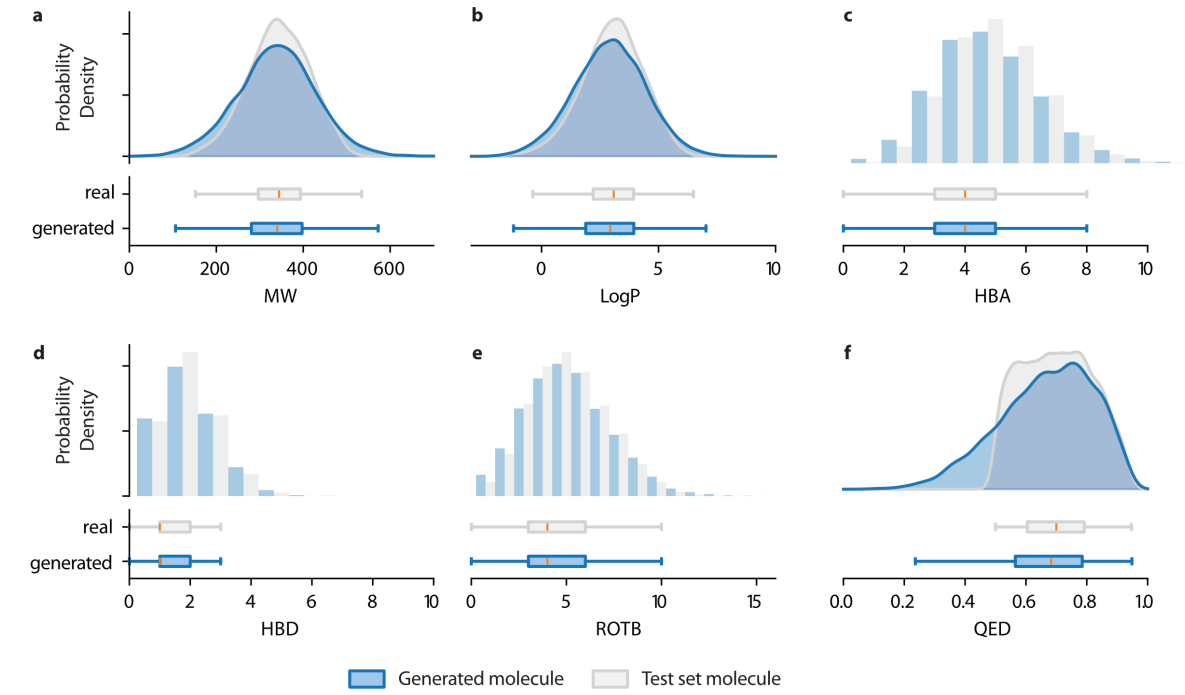

Figure S12: The distribution of 2D molecular properties of generated molecules and test set molecules. **a.** Molecular weight (MW). **b.** LogP. **c.** The number of hydrogen bond acceptors (HBA). **d.** The number of hydrogen bond donors (HBD). **e.** The number of rotatable bonds (ROTB). **f.** Druglikeness (QED). Generated molecules are shown in blue, and test set molecules are shown in grey.

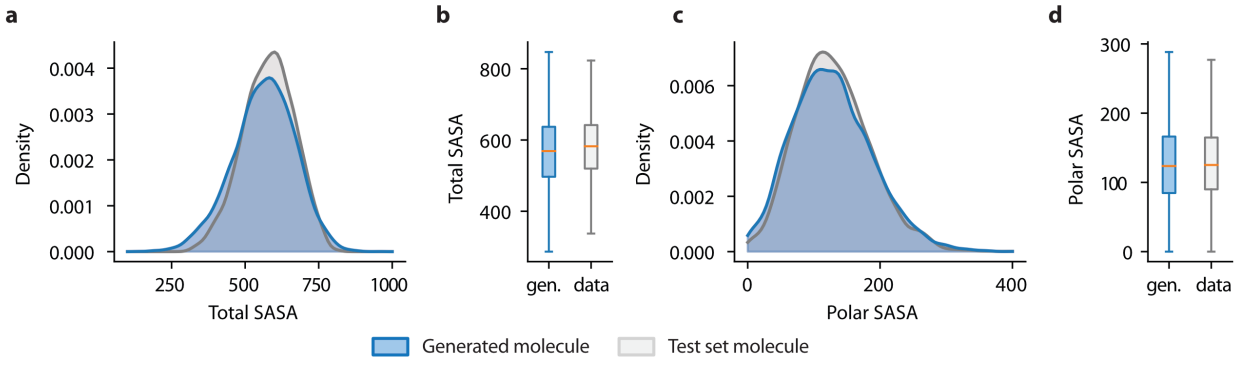

Figure S13: Comparison of Total SASA (**a,b**) and Polar SASA (**c,d**) between generated and test set molecules (**a,c**: Kernel density estimation; **b,d**: Box plot; Blue: generated molecules; Grey: test set molecules).

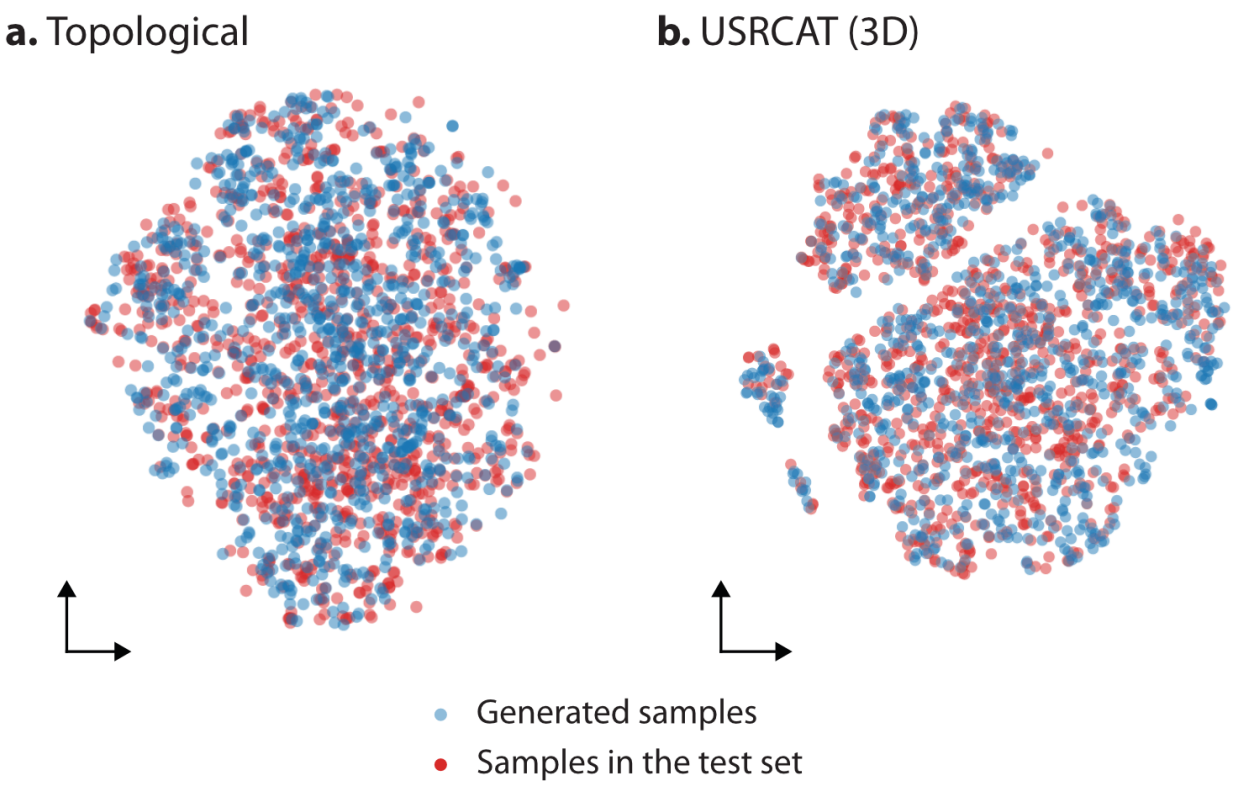

Figure S14: **a-b.** A t-SNE visualization of the distribution of Morgan (**a**) and USRCAT (**b**) fingerprint in two dimension space (Blue: generated samples; Red: samples in the test set).

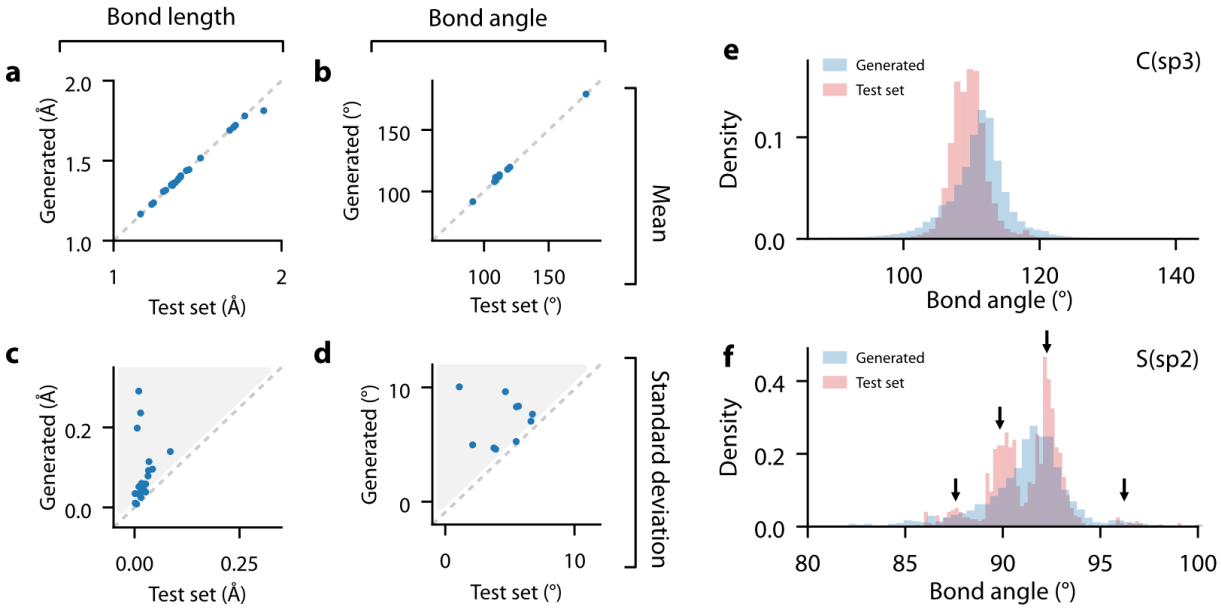

Figure S15: Comparing the distribution of bond lengths and bond angles between generated and test set molecules. **a-b.** Average bond lengths (**a**) and bond angles (**b**) for each environment key in generated (y-axis) and test set (x-axis) molecules. **c-d.** Standard deviation of bond lengths (**c**) and bond angles (**d**) each environment key in generated (y-axis) and test set (x-axis) molecules. **e-f.** The distribution of bond angle in two atomic environments: **e.** sp3 hybridized carbon atom; **f.** sp2 hybridized sulfur atom.

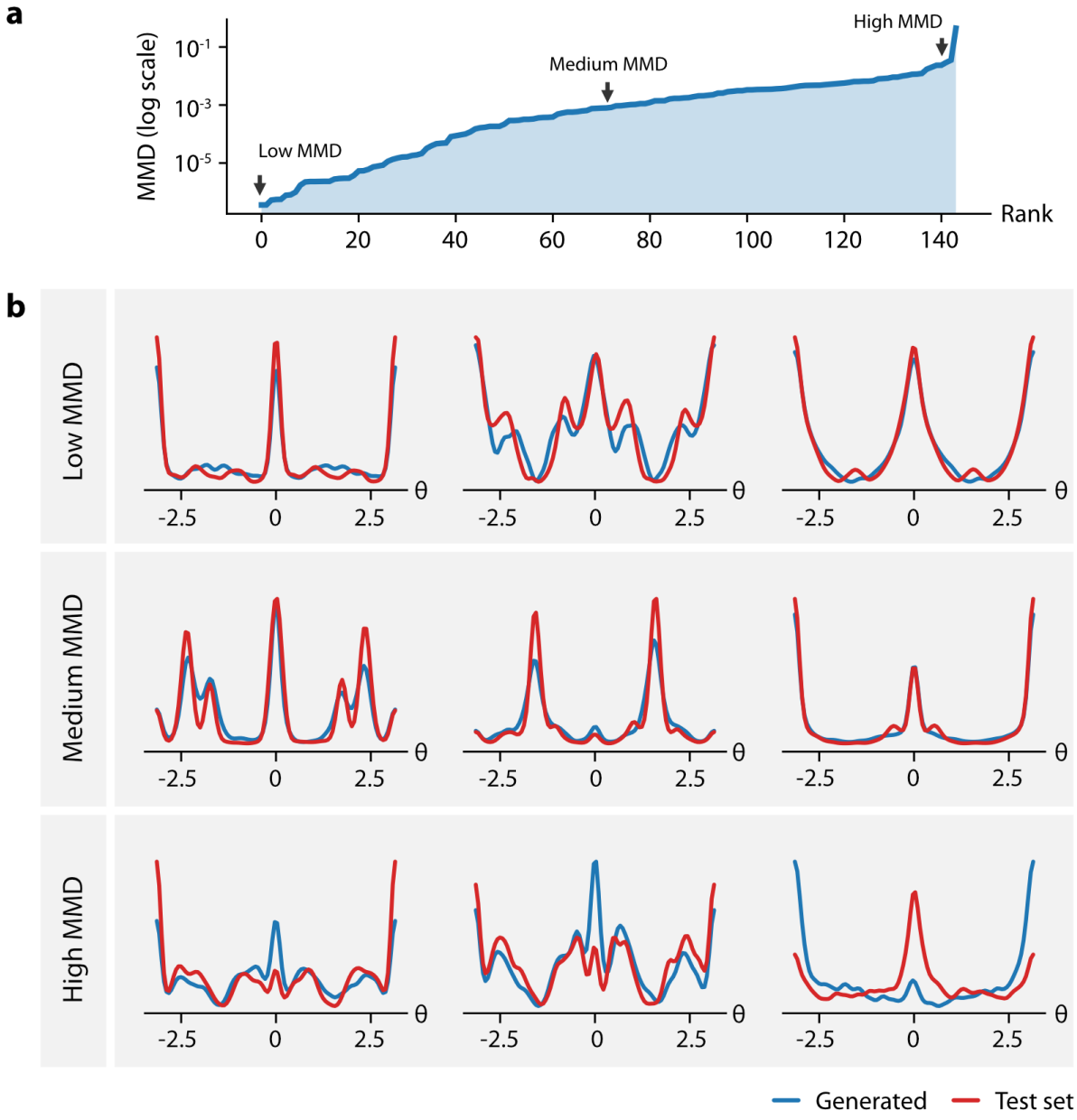

Figure S16: Comparing the distribution of torsion angles between generated and test set molecules. **a.** The MMD values of the torsion distribution for each pattern, ranked from lowest to highest. **b.** Torsion distributions with the highest, medium and lowest MMD values. Blue line indicates the generated samples, and red lines indicates samples in the test set.

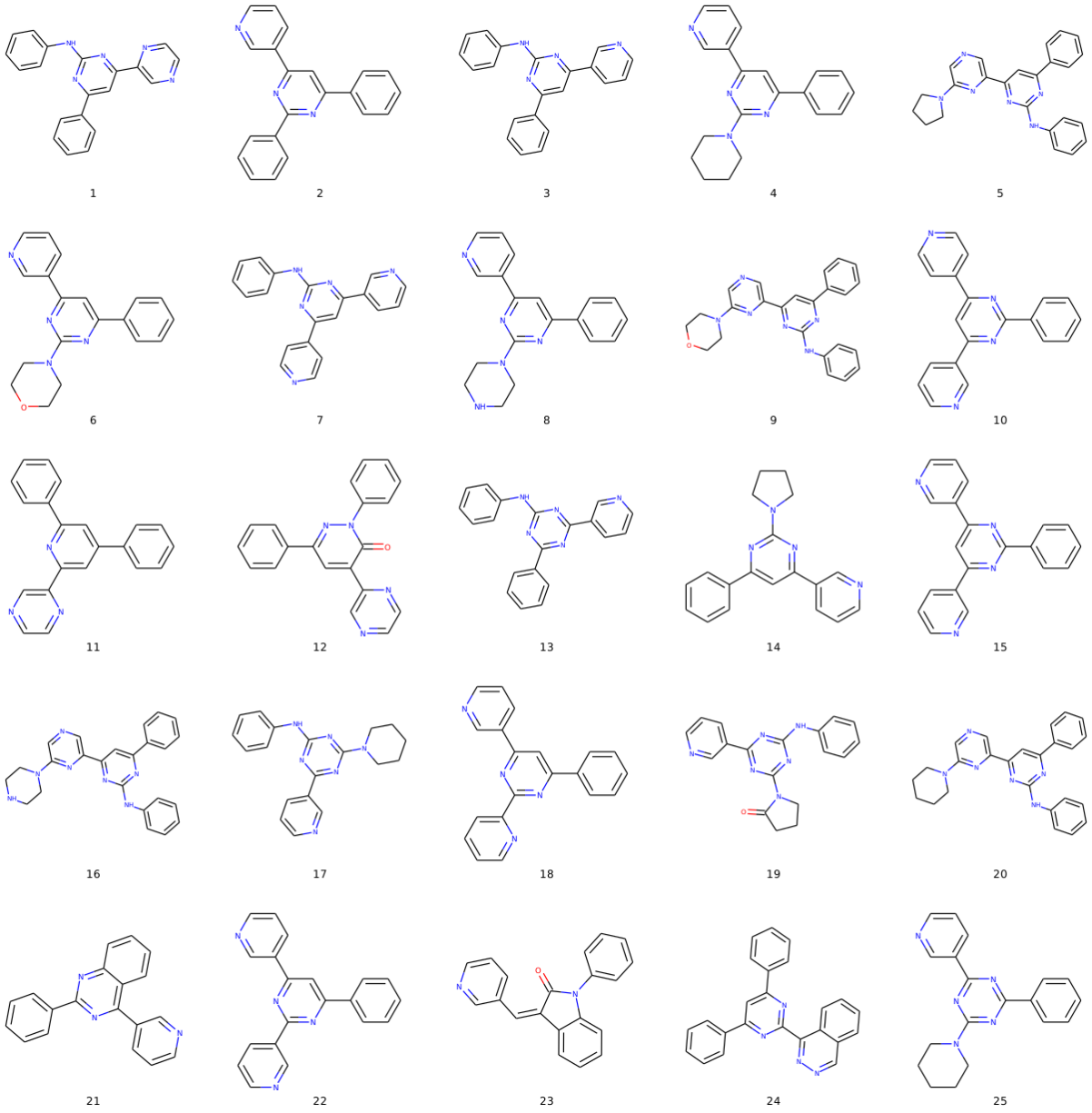

Figure S17: Top 25 scaffolds generated using DeepLigBuilder.

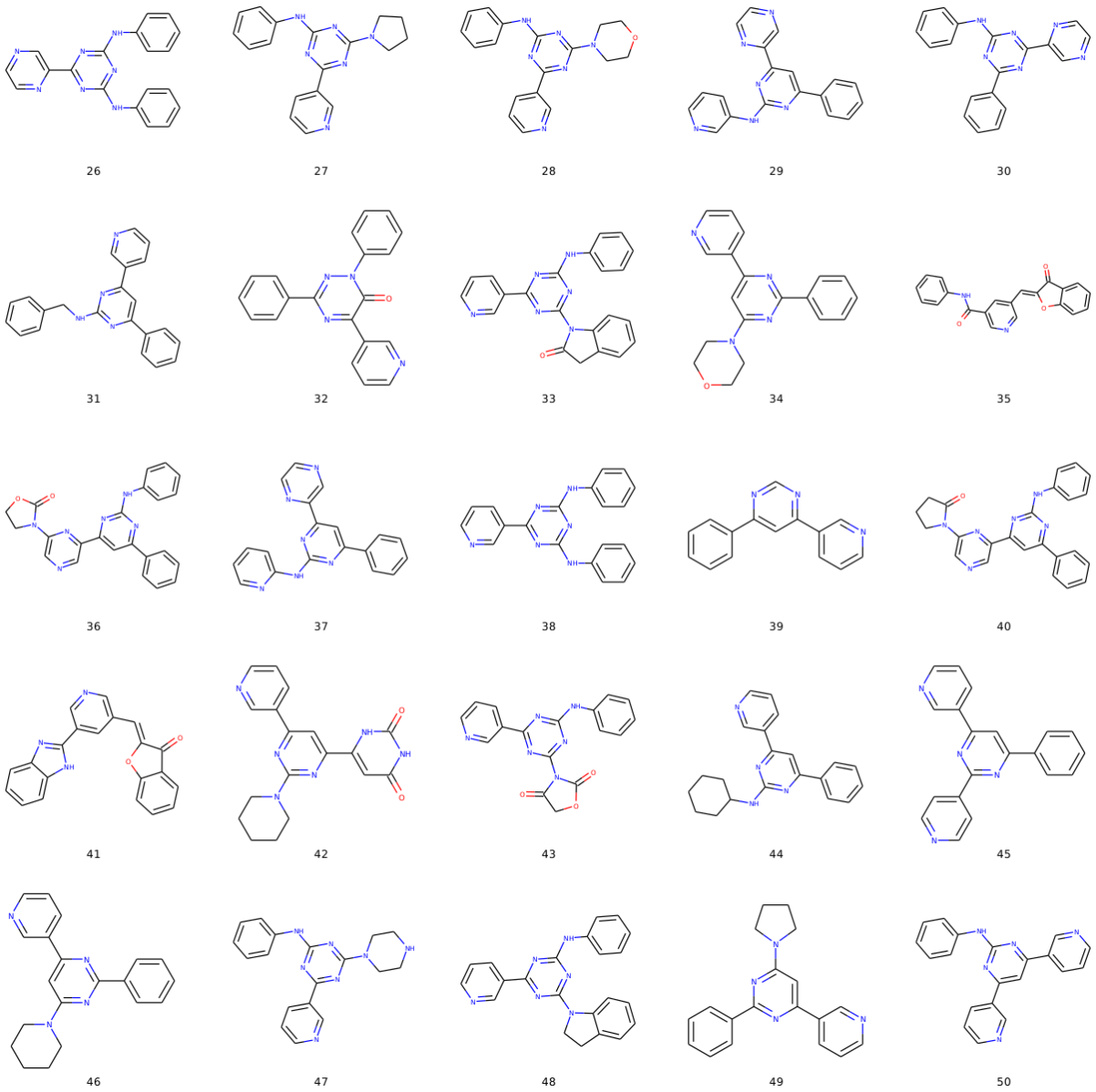

Figure S18: Top 26th to 50th scaffolds generated using DeepLigBuilder.

References

(S1) Gebauer, N. W. A.; Gastegger, M.; Schütt, K. T. Symmetry-adapted generation of 3d point sets for the targeted discovery of molecules. *arXiv:1906.00957* **2019**,

(S2) Ragoza, M.; Masuda, T.; Koes, D. R. Learning a Continuous Representation of 3D Molecular Structures with Deep Generative Models. *arXiv:2010.08687* **2020**,

(S3) Mendez, D.; Gaulton, A.; Bento, A. P.; Chambers, J.; De Veij, M.; Félix, E.; Magariños, M.; Mosquera, J.; Mutowo, P.; Nowotka, M.; Gordillo-Marañón, M.; Hunter, F.; Junco, L.; Mugumbate, G.; Rodriguez-Lopez, M.; Atkinson, F.; Bosc, N.; Radoux, C.; Segura-Cabrera, A.; Hersey, A.; Leach, A. ChEMBL: towards direct deposition of bioassay data. *Nucleic Acids Res.* **2018**, *47*, gky1075–.

(S4) Ronneberger, O.; Fischer, P.; Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. *arXiv:1505.04597* **2015**,

(S5) Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; Dahl, G. E. Neural Message Passing for Quantum Chemistry. *arXiv:1704.01212* **2017**,

(S6) Huang, G.; Liu, Z.; Maaten, L. v. d.; Weinberger, K. Q. Densely Connected Convolutional Networks. *arXiv:1608.06993* **2016**,

(S7) Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; Polosukhin, I. Attention Is All You Need. *arXiv:1706.03762* **2017**,

(S8) Thomas, N.; Smidt, T.; Kearnes, S.; Yang, L.; Li, L.; Kohlhoff, K.; Riley, P. Tensor field networks: Rotation- and translation-equivariant neural networks for 3D point clouds. *arXiv:1802.08219* **2018**,

(S9) Anderson, B.; Hy, T.-S.; Kondor, R. Cormorant: Covariant Molecular Neural Networks. *arXiv:1906.04015* **2019**,

(S10) Ingraham, J.; Riesselman, A. J.; Sander, C.; Marks, D. S. Learning Protein Structure with a Differentiable Simulator. 2019.

(S11) Li, Y.; Hu, J.; Wang, Y.; Zhou, J.; Zhang, L.; Liu, Z. DeepScaffold: A Comprehensive Tool for Scaffold-Based De Novo Drug Discovery Using Deep Learning. *J. Chem. Inf. Model.* **2019**, *60*, 77–91.

(S12) Gao, H.; Ji, S. Graph U-Nets. *arXiv:1905.05178* **2019**,

(S13) Bemis, G. W.; Murcko, M. A. The Properties of Known Drugs. 1. Molecular Frameworks. *J. Med. Chem.* **1996**, *39*, 2887–2893.

(S14) Wilkens, S. J.; Janes, J.; Su, A. I. HierS: Hierarchical Scaffold Clustering Using Topological Chemical Graphs. *J. Med. Chem.* **2005**, *48*, 3182–3193.

(S15) Simm, G. N. C.; Pinsler, R.; Csányi, G.; Hernández-Lobato, J. M. Symmetry-Aware Actor-Critic for 3D Molecular Design. *arXiv:2011.12747* **2020**,

(S16) Germain, M.; Gregor, K.; Murray, I.; Larochelle, H. MADE: Masked Autoencoder for Distribution Estimation. *arXiv:1502.03509* **2015**,

(S17) Kim, H.; Lee, H.; Kang, W. H.; Lee, J. Y.; Kim, N. S. SoftFlow: Probabilistic Framework for Normalizing Flow on Manifolds. *arXiv:2006.04604* **2020**,

(S18) Bickerton, G. R.; Paolini, G. V.; Besnard, J.; Muresan, S.; Hopkins, A. L. Quantifying the chemical beauty of drugs. *Nat. Chem.* **2012**, *4*, 90–98.

(S19) Li, Y.; Zhang, L.; Liu, Z. Multi-objective de novo drug design with conditional graph generative model. *J. Cheminformatics* **2018**, *10*, 33.

(S20) Arús-Pous, J.; Johansson, S. V.; Prykhodko, O.; Bjerrum, E. J.; Tyrchan, C.; Reymond, J.-L.; Chen, H.; Engkvist, O. Randomized SMILES strings improve the quality of molecular generative models. *J. Cheminformatics* **2019**, *11*, 71.

(S21) Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; Chintala, S. PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems. 2019.

(S22) Kingma, D. P.; Ba, J. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980* **2014**,

(S23) Lam, S. K.; Pitrou, A.; Seibert, S. Numba: A llvm-based python jit compiler. Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC. 2015; pp 1–6.

(S24) Polykovskiy, D.; Zhebrak, A.; Sanchez-Lengeling, B.; Golovanov, S.; Tatanov, O.; Belyaev, S.; Kurbanov, R.; Artamonov, A.; Aladinskiy, V.; Veselov, M.; Kadurin, A.; Johansson, S.; Chen, H.; Nikolenko, S.; Aspuru-Guzik, A.; Zhavoronkov, A. Molecular Sets (MOSES): A Benchmarking Platform for Molecular Generation Models. *Front. Pharmacol.* **2020**, *11*, 565644.

(S25) Brown, N.; Fiscato, M.; Segler, M. H. S.; Vaucher, A. C. GuacaMol: Benchmarking Models for de Novo Molecular Design. *J. Chem. Inf. Model.* **2019**, *59*, 1096–1108.

(S26) Sauer, W. H. B.; Schwarz, M. K. Molecular Shape Diversity of Combinatorial Libraries: A Prerequisite for Broad Bioactivity. *J. Chem. Inf. Comp. Sci.* **2003**, *43*, 987–1003.

(S27) Mitternacht, S. FreeSASA: An open source C library for solvent accessible surface area calculations. *F1000Research* **2016**, *5*, 189.

(S28) Schreyer, A. M.; Blundell, T. USRCAT: real-time ultrafast shape recognition with pharmacophoric constraints. *J. Cheminformatics* **2012**, *4*, 27.

(S29) Grant, J. A.; Gallardo, M.; Pickup, B. T. A fast method of molecular shape comparison: A simple application of a Gaussian description of molecular shape. *J. Comput. Chem.* **1996**, *17*, 1653–1666.

(S30) Nishino, R.; Loomis, S. H. C. CuPy: A NumPy-Compatible Library for NVIDIA GPU Calculations. *31st confernce on neural information processing systems* **2017**, 151.

(S31) Kynkäänniemi, T.; Karras, T.; Laine, S.; Lehtinen, J.; Aila, T. Improved Precision and Recall Metric for Assessing Generative Models. *arXiv:1904.06991* **2019**,

(S32) Riniker, S.; Landrum, G. A. Better Informed Distance Geometry: Using What We Know To Improve Conformation Generation. *J. Chem. Inf. Model.* **2015**, *55*, 2562–2574.

(S33) Neese, F.; Wennmohs, F.; Becker, U.; Riplinger, C. The ORCA quantum chemistry program package. *J. Chem. Phys.* **2020**, *152*, 224108.

(S34) Yang, X.; Zhang, J.; Yoshizoe, K.; Terayama, K.; Tsuda, K. ChemTS: an efficient python library for de novo molecular generation. *Sci. Technol. Adv. Mat.* **2017**, *18*, 972–976.

(S35) Xiao, C.; Huang, R.; Mei, J.; Schuurmans, D.; Muller, M. Maximum entropy monte-carlo planning. *Advances in Neural Information Processing Systems* **2019**, *32*, 9520–9528.

(S36) Koes, D. R.; Baumgartner, M. P.; Camacho, C. J. Lessons Learned in Empirical Scoring with smina from the CSAR 2011 Benchmarking Exercise. *J. Chem. Inf. Model.* **2013**, *53*, 1893–1904.

(S37) Chaslot, G. M. J. B.; Winands, M. H. M.; Herik, H. J. v. d. Parallel Monte-Carlo tree search. *Proceedings of 6th International Conference on Computers and Games (CG)* **2008**, *5131*, 60–71.