Autoregressive fragment-based diffusion for pocket-aware ligand design

Anonymous Author(s)
Affiliation
Address
Email

Abstract
In this work, we introduce AutoFragDiff, a fragment-based autoregressive diffusion model for generating 3D molecular structures conditioned on target protein structures. We employ geometric vector perceptrons to predict atom types and spatial coordinates of new molecular fragments conditioned on molecular scaffolds and protein pockets. Our approach improves the local geometry of the resulting 3D molecules while maintaining high predicted binding affinity to protein targets. The model can also perform scaffold extension from user-provided starting molecular scaffold.

1 Introduction
Rational drug design against defined binding pockets relies heavily on computational modeling. Stokes et al. [2020], Anderson [2003] Traditionally, the diversity of small-molecule candidates and the high degrees of freedom inherent in ligand-protein binding systems make navigating chemical space computationally intensive. Lipinski and Hopkins [2004] Moreover, target-aware molecular design strives to balance optimizing for potency against specific target structures while maintaining desirable absorption, distribution, metabolism, and excretion (ADME) and pharmacokinetic and pharmacodynamic (PKPD) properties. Skalic et al. [2019] While many target protein structures are available, effectively harnessing this information to design novel drug-like compounds with desired therapeutic effects remains an active area of research. Ragoza et al. [2017]

Diffusion models Ho et al. [2020], Kingma et al. [2021] generate 3D molecular structures from underlying distributions of molecular data. Hoogeboom et al. [2022], thus enabling the generation of diverse molecular candidates that reflect real chemical space. However, these models struggle to capture the nuances of local molecular geometry. Specifically, maintaining the correct spatial arrangements and conformations of functional groups and atoms remains challenging. Harris et al. [2023] While the overall structure might resemble known molecules, minor deviations in local geometry can significantly impact the bioactivity and specificity of the generated compounds.

Many pocket-specific molecule generation models have leveraged autoregressive strategies. In these models, atoms are placed individually, and bonds are determined separately. Drotár et al. [2021], Liu et al. [2022]. However, this sequential approach can be cumbersome and error-prone; even generating a benzene ring is a laborious six-step procedure. Fragment-based generation strategies sidestep some of these drawbacks. Our work employs Autoregressive Diffusion Models (ARDMs) Hoogeboom et al. [2022], Drotár et al. [2021] which can generate data in a flexible order. This unique feature enables ARDMs to bridge the gap between order-agnostic autoregressive and diffusion-based generative models. Igashov et al. [2023], Schneuing et al. [2022], Guan et al. [2023] Unlike their traditional counterparts, ARDMs don’t adhere to strict architectural norms for neural networks, yet they achieve comparable results in fewer steps.

In this study, we combine fragment-based drug design with autoregressive diffusion models. Unlike traditional autoregressive methods that work atom by atom, this combined approach allows each fragment to undergo a denoising process, predicting atom coordinates and atom types. Rather than relying on a fixed fragment library, our approach dynamically generates fragments, providing flexibility in the diversity of fragments produced. This approach generates molecules with more accurate local geometries for pocket-based molecule generation, delivering greater precision and efficiency in drug design.

2 Related Work

Generative models and geometric deep learning have influenced recent pocket-based drug design. Atz et al. [2021], Bronstein et al. [2017], Li et al. [2021] introduced an autoregressive generative model designed to sample ligands, using the pocket as a conditioning constraint. Building on this work, Peng et al. [Peng et al. 2022] introduced Pocket2Mol, which uses an E(3) equivariant graph neural network [Satorras et al. 2021] that accounts for rotation and translation symmetries in 3D space for more accurate molecular representations. Similarly, Drotar et al. [2021], Liu et al. [2022] explored autoregressive models for molecular generation, generating atoms sequentially. These models incorporate angles during generation to improve molecular detail and accuracy.

Diffusion models enable pocket-free and pocket-based drug design. Kingma et al. [2021], Hoogeboom et al. [2022] introduced Equivariant Diffusion Models (EDMs), which simultaneously learn continuous coordinates and atom types for molecule generation. Multiple studies built on this approach: GeoDiff [Xu et al. 2022] predicts a molecule’s 3D conformation and DiffLinker [Ishkov et al. 2022] learns to connect seed fragments. Similarly, Schneuing et al. [Schneuing et al. 2022] developed DiffSBDD, a denoising diffusion model for pocket-based molecule design. Guan et al.’s TargetDiff uses SE(3)-equivariant networks to explicitly learn the generative process for continuous coordinates and categorical atom types [Guan et al. 2023].

3 Methods

3.1 Problem Definition

We represent the protein pocket and the ligand as point clouds with atomic coordinates $r$ and corresponding feature vectors $h$. The feature vector is the one-hot encoded atom type for ligand atoms and element type, plus amino acid type for the pocket atoms. For the pocket $P = (r^P_i, h^P_i)^{N_P}_{i=1}$, and for the molecule $M = (r^M_i, h^M_i)^{N_M}_{i=1}$, where $N_P$ and $N_M$ are the number of atoms in the pocket and molecule respectively. We further separate each molecule into multiple fragments and molecular scaffolds $M = [(r^F_i, r^P_i), (h^M_i, h^F_i)]^{N_F}_{i=1}$, $M_F$ and $F$ superscripts represent molecule scaffold and the fragment respectively. Note that for each molecule, there exist multiple fragments and scaffolds. The autoregressive diffusion process aims to generate a new fragment conditioned on a molecular scaffold and protein pocket at each step.

3.2 Diffusion Process

The diffusion process iteratively adds noise to data point $x$ and trains a neural network to remove noise progressively. Generative denoising inverts the trajectory when $x$ is unknown. This process for
fragment $F$ is conditioned on the molecular scaffold $M_F$ and the protein pocket $P$:

$$p \left( z^F_{t-1} | z^F_t, M_F, P \right) = q \left( z^F_{t-1} | \hat{x}^F_t, z^F_t \right)$$  \hspace{1cm} (1)$$

where $\hat{x}_t = (1/\alpha_t) z_t - (\sigma_t/\alpha_t) \epsilon_t$ is the approximation of $x^F_t$ computed by neural network $\phi$ using $\epsilon_t = \phi(z_t, t, M_F, P)$. We use Geometric Vector Perceptrons (GVP) \cite{tabor2021gvp} to parameterize $\phi$ because they outperform equivariant neural networks. \cite{satorras2021difflinker} Following DiffLinker \cite{igashov2022difflinker}, Jing et al. \cite{jin2020diffdock}, Torge et al. \cite{torge2023diffdock}, we define the "anchor point" as the scaffold atom bonded to the fragment $F$. We ensure the GNN is translationally invariant by first centering the data around the anchor point $a$ and then sampling from $\mathcal{N}(0, I)$ instead of sampling the initial noise from $\mathcal{N}(f(a), I)$ where $f(a)$ is the anchor point center of mass.

During training, we only add noise to coordinates $x$ and feature vector $h$ of the fragment $F$. We keep the scaffold molecule $M_F$ and the protein pocket intact. The input to the neural network is the noised version of fragment $z_t^F$ at time $t$ and the context $u$, which contains the molecular scaffold $M_F$, the protein pocket $P$, and the anchor point $a$. The predicted noise $\hat{\epsilon}_t^F$ for the fragment $F_t$ includes coordinates and feature vector $\hat{\epsilon}_t^F = [\hat{x}_t^F, \hat{h}_t^F]$. We only use the predicted coordinates and feature vectors for the fragment atoms and discard the rest.

Additionally, we train a separate model for anchor point prediction (see SI). During sampling, this AnchorGNN (see SI) model predicts the anchor point from among the scaffold atoms. We sample the fragment size from the data distribution conditioned on the pocket size near the anchor point. We repeat the fragment generation process until we reach a maximum number of fragments or molecule size. Algorithms 1 and 2 in SI define AutoFragDiff training and sampling procedures, respectively.

## 4 Datasets

### CrossDock

We use CrossDock\cite{francoeur2020crossdock} to evaluate AutoFragDiff for pocket-based molecule generation. Similar to other studies, we refined the original 22.5 million docked protein binding complexes by filtering for low (<1Å) RMSD and sequence identity of less than 30%. This procedure yielded 100,000 training complexes and 100 previously unseen testing pockets. We used RDKit \cite{landrum2013rdkit} and BRICS \cite{degen2008brics} to fragment molecules by breaking bonds between rings without breaking fused ring systems. We used a maximum of 8 fragments per molecule. We used breadth-first and depth-first search traversals of each molecule’s fragment connectivity graph to avoid computing an intractable enumeration of all potential fragment-wise molecule reconstructions. At each reconstruction step, we saved the scaffold atoms and coordinates, the added fragment, and the anchor point where the scaffold connects to the next fragment.

## 5 Results

As in TargetDiff \cite{guan2023targetdiff}, we use openbabel \cite{boyle2011openbabel} to reconstruct the molecules from the generated atomic point clouds. In terms of the Jensen Shannon Divergence (JSD) of angles and dihedrals for common ring structures in CrossDock, AutoFragDiff significantly surpasses other models (Table 1). Although it was not a focus of this study, we also assess the generated molecules for various chemical properties (Table 2), including drug-likeliness (QED) and average synthetic accessibility (SA) \cite{erl2009}. "Diversity" evaluates the average molecular fingerprint similarity across all generated molecule pairs. AutoFragDiff generates realistic molecules with higher calculated binding affinity than the molecules in the test set and exhibits results on par with state-of-the-art models.

### Scaffold Extension

Since our model adds fragments to an existing molecular scaffold at each step, it can further optimize a user-provided starting scaffold. We extracted the Murcko scaffold from every molecule in the CrossDock test set to test the concept. We augmented each scaffold with up to 4 fragments, generating 20 distinct molecules per CrossDock molecule. 70% of the newly generated
Table 1: JSD of angles and dihedrals for most common rings in CrossDock dataset. Best score highlighted in dark gray; second best in light gray.

<table>
<thead>
<tr>
<th>Model</th>
<th>angles</th>
<th>dihedrals</th>
<th>angles</th>
<th>dihedrals</th>
<th>angles</th>
<th>dihedrals</th>
<th>angles</th>
<th>dihedrals</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D-SBDD</td>
<td>0.458</td>
<td>0.666</td>
<td>0.293</td>
<td>0.300</td>
<td>0.457</td>
<td>0.625</td>
<td>0.342</td>
<td>0.439</td>
</tr>
<tr>
<td>Pocket2Mol</td>
<td>0.438</td>
<td>0.574</td>
<td>0.321</td>
<td>0.272</td>
<td>0.347</td>
<td>0.551</td>
<td>0.408</td>
<td>0.478</td>
</tr>
<tr>
<td>DiffSBDD</td>
<td>0.342</td>
<td>0.549</td>
<td>0.310</td>
<td>0.235</td>
<td>0.254</td>
<td>0.546</td>
<td>0.363</td>
<td>0.435*</td>
</tr>
<tr>
<td>TargetDiff</td>
<td>0.203</td>
<td>0.459</td>
<td>0.154</td>
<td>0.176</td>
<td>0.140</td>
<td>0.460</td>
<td>0.335</td>
<td>0.437</td>
</tr>
<tr>
<td>AutoFragDiff(ours)</td>
<td>0.103</td>
<td>0.151</td>
<td>0.175</td>
<td>0.166</td>
<td>0.078</td>
<td>0.241</td>
<td>0.214</td>
<td>0.277</td>
</tr>
</tbody>
</table>

Table 2: Pocket-based generative models comparison. Best score highlighted in dark gray; second best in light gray.

<table>
<thead>
<tr>
<th>Method</th>
<th>Vina (↓)</th>
<th>Diversity (↑)</th>
<th>QED (↑)</th>
<th>SA (↑)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D-SBDD</td>
<td>-6.71</td>
<td>0.70</td>
<td>0.49</td>
<td>0.62</td>
</tr>
<tr>
<td>Pocket2Mol</td>
<td>-7.15</td>
<td>0.69</td>
<td>0.56</td>
<td>0.74</td>
</tr>
<tr>
<td>DiffSBDD</td>
<td>-6.90</td>
<td>0.73</td>
<td>0.48</td>
<td>0.63</td>
</tr>
<tr>
<td>TargetDiff</td>
<td>-7.55</td>
<td>0.72</td>
<td>0.49</td>
<td>0.61</td>
</tr>
<tr>
<td>AutoFragDiff(ours)</td>
<td>-7.45</td>
<td>0.69</td>
<td>0.44</td>
<td>0.61</td>
</tr>
<tr>
<td>CrossDock Test Set Molecules</td>
<td>-7.10</td>
<td>-</td>
<td>0.47</td>
<td>0.73</td>
</tr>
</tbody>
</table>

Molecules exhibited higher calculated binding affinity than their corresponding starting molecule (average Vina score of -7.8 generated versus -7.1 CrossDock). Figure 2 contains representative examples for an *S. cerevisiae* Cytochrome-c peroxidase pocket (pdb: 1a2g).

Figure 2: Scaffold (red) extension examples on a Cytochrome-c peroxidase (pdb: 1a2g).

6 Conclusion

We introduce AutoFragDiff (https://anonymous.4open.science/r/autofragdiff), an open-source autoregressive fragment-based diffusion model tailored for pocket-free and pocket-based molecule generation. A standout feature of AutoFragDiff is its capability for scaffold extension, which is a key aspect of many real-world drug design applications, especially in close-in optimization around lead series. The model is adept at generating molecules with high-quality local geometry and exhibits robust binding affinity to target proteins. Looking forward, we aim to enhance the ligand affinity within the pocket using guidance strategies and model architecture improvements.
References


7 Appendix

A.1 Training and Sampling

Algorithm 1 Training

1: Input: Fragment \( x^F \), Scaffold \( M_F \), anchor point \( a \), protein pocket \( P \), neural network \( \phi \)
2: Sample Permutation order \( \sigma \sim S_D \)
3: Sample fragment \( F \)
4: Sample \( t \sim \mathcal{U}(0, \ldots, T) \), \( \epsilon_t \sim \mathcal{N}(0, I) \)
5: \( z^F_t \leftarrow \alpha_t x^F + \sigma_t \epsilon_t \)
6: \( \hat{\epsilon}_t \leftarrow \phi(z_t, M_F, a, t, P) \)
7: Minimize \( ||\epsilon - \hat{\epsilon}_t||_2 \)

Algorithm 2 Sampling

1: for \( i \) in \( 1..D \); do
2: Input: Scaffold \( M_F \), anchor point \( a_i \), protein pocket \( P \), neural network \( \phi \)
3: Center everything at \( f(a_i) \)
4: Sample \( z^F_{0,i} \sim \mathcal{N}(0, I) \)
5: for \( t \) in \( T; T-1; \ldots; 1 \) do
6: Sample \( \epsilon_t \sim \mathcal{N}(0, I) \)
7: \( \hat{\epsilon}_t \leftarrow \phi(z^F_{t,i}, t, M_F, a_i, P) \)
8: \( z^F_{t-1,i} \leftarrow (1/\bar{\alpha}_t) \cdot z_t - \bar{\sigma}_t^2/(\bar{\alpha}_t \sigma_t) \cdot \hat{\epsilon}_t + \zeta_t \cdot \epsilon \)
9: end for
10: Sample \( x^F : \sim p(x^F | z^F_0, M_F, a_i, P) \)
11: end for

For sampling molecule sizes, we first bin the pocket volumes into 10 bins (using grids inside the protein pocket) and find the distribution of molecule sizes for each bin. During sampling, we sample molecule sizes from the distribution of the corresponding volume bin. For the first generation step, the anchor point is selected from the pocket atoms in contact with the original ligand. We first bin the pocket volume within 3.5 Å of the anchor point for fragment size and then sample the fragment sizes from the corresponding bin. The average size of generated molecules from our model is 27 atoms.

A.2 Diffusion Process

At each timestep \( t = 0 \ldots T \) the conditional distribution of the intermediate state \( z^F_t \) for a single fragment \( F \) given the previous state is defined by the multivariate normal distribution:

\[
q(z^F_t | z^F_{t-1}) = \mathcal{N} \left( z^F_t | \bar{\alpha}_t z^F_{t-1}, \bar{\sigma}_t^2 I \right) \tag{2}
\]

In this equation \( \bar{\alpha}_i = \alpha_i/\alpha_{i-1} \) controls how much signal is retained and \( \bar{\sigma}_i = \sigma_i^2 - \bar{\alpha}_i^2 \sigma_{i-1}^2 \) controls how much noise is added. The full transition model for diffusion is Markovian:

\[
q(z^F_0, z^F_1, \ldots, z^F_T | x^F) = q(z^F_0 | x^F) \prod_{t=1}^T q(z^F_t | z^F_{t-1}) \tag{3}
\]

The true denoising process has a closed-form solution when conditioned on \( x^F \):

\[
q(z^F_{t-1} | z^F_t, x^F) = \mathcal{N} \left( z^F_{t-1} | \mu_t(x, z_t), \zeta_t I \right) \tag{4}
\]

where \( \mu_t(x^F, z^F_t) \) and \( \zeta_t \) have analytical solutions:

\[
\mu_t(x^F, z^F_t) = \frac{\bar{\alpha}_t \sigma^2_t}{\bar{\sigma}_t^2} z_t + \frac{\alpha_s \sigma^2_t}{\sigma_t^2}, \quad \zeta_t = \frac{\bar{\sigma}_t \sigma_{t-1}}{\sigma_t} \tag{5}
\]
We trained AutoFragDiff with $T = 500$ diffusion steps using a polynomial noise scheduler:

$$\alpha_t = (1 - 2s) \cdot (1 - (t/T)^2)$$

(6)

where $s = 10^{-5}$ is the precision value to help with numerical issues.

### A.3 Geometric Vector Perceptrons

GVP [Jing et al., 2020] uses nodes with scalar features $s$ as inputs. These scalars represent embedded features of atoms without accompanying vector features. Edges within the graph incorporate a normed direction vector alongside the distance between two nodes. More specifics about this can be found in GVP paper [Jing et al., 2020].

The attributes of nodes and edges undergo linear transformations. Torge et al. [2023] Edge embeddings are achieved in two phases: initially, their inputs are normalized using layer normalization [Ba et al., 2016], and following this, they are channeled through a GVP. Here, both $\sigma$ and $\sigma^+$ operate as the identity function, resulting in a scalar with a hidden size of $h/2$ and a singular vector. Nodes undergo a parallel embedding process, culminating in outputs of $h$ scalars and $h/2$ vectors, summing up to $h$ features. The message-passing layers can be expressed as:

$$m_{vw}' = \phi_e(h_v, h_w, e_{vw}),$$

$$m_v' = \sum_{w \in N_v} \tilde{e}_{vw} m_{vw},$$

$$h_v' = \phi_h(h_v, m_v').$$

(7)

Within this equation, $\tilde{e}_{vw} = \phi_{att}(m_{vw})$ acts as an attention mechanism, enabling the learning of soft edge estimates, mirroring the approach in EGNN. The function $\phi_e$ combines three GVPs featuring hidden sizes $(h, h/2)$. Notably, the final GVP has $\sigma$ as its identity function. Meanwhile, $\phi_{att}$ embodies a single GVP translating to a singular scalar with $\sigma$ functioning as the sigmoid activation.

A factor of $C = 100$ normalizes the resulting output.

The relationship between $\phi_h(h_v, m_v')$ is captured by the equation $\phi_h(h_v, m_v') = \text{norm}(h_v + \phi_h'(\text{norm}(h_v + m_v')))$. This employs a residual architecture where $\phi_h'$ integrates two GVPs with sizes $(h, h/2)$. This encapsulates input, hidden, and output dimensions. The terminal layer once again adopts $\sigma$ as the identity function. The term "norm" represents layer normalization, which isn’t applied to vectors.

### A.4 Autoregressive Diffusion Models

Autoregressive models can factorize a multivariate distribution into a product of $D$ univariate distributions.

$$\log p(x) = \sum_{t=1}^{D} \log p(x_t|x_1, \ldots, x_{t-1})$$

(8)

Sampling from such models can be done through $D$ iterative sampling steps. Order agnostic models can generate variables with random orderings $\sigma \in S_D$ where $S_D$ is the set of all permutations for building the molecule from its fragments. The log-likelihood of these models can be written as:

$$\log p(M|P) \geq \mathbb{E}_{\sigma \sim \mathcal{U}(S_D)} \sum_{i=1}^{D} \log p(M_{\sigma_i}|M_{\sigma(<i)}, P)$$

(9)

In this equation, $M$ is the set of molecule atoms, $P$ is the set of protein atoms, and $M_{\sigma_i}$ is the molecule generated with sampled ordering $\sigma$ at the fragment step $i$. Hoogeboom et al. [Hoogeboom et al., 2021] derived an objective for order agnostic diffusion models that only needs to be optimized for a single step at a time.
According to this objective, during training, we sample a random order $\sigma$ of molecule generation uniformly from the set of all generation orders $S_D$, and a single fragment from the uniform distribution of all fragments in the molecule. We train the diffusion model to predict this single fragment. In practice, we optimize a simplified $L(t) = ||\epsilon - \hat{\epsilon}_t||^2$ loss by mini-batch gradient descent.

### A.5 Hyperparameters

We consider the protein graph as the protein atoms within 7 Å of the original ligand. Edges within the ligand are fully connected, while protein-ligand and protein-protein edges are drawn with a radius threshold of 4.5 Å. The edge features for nodes $i$ and $j$ are the distance $d_{ij}$ and the normalized direction vector $(x_i - x_j)/d_{ij}$. As previously suggested by Hoogeboom et al. [2022], we scale node types $h$ by a factor of 0.25. The final model had 6 GVP layers, with hidden dimension of 128 and a joint embedding dimension of 32. We trained the model with a learning rate of $2 \times 10^{-4}$ for 500 epochs.

### A.6 Additional results

Table 3 compares different models by JSD for different types of bonds.

<table>
<thead>
<tr>
<th>Bond</th>
<th>3D-SBDD</th>
<th>Pocket2Mol</th>
<th>DiffSBDD</th>
<th>TargetDiff</th>
<th>AutoFragDiff</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-C</td>
<td>0.576</td>
<td>0.455</td>
<td>0.347</td>
<td>0.286</td>
<td>0.340</td>
</tr>
<tr>
<td>C=C</td>
<td>0.421</td>
<td>0.561</td>
<td>0.314</td>
<td>0.220</td>
<td>0.223</td>
</tr>
<tr>
<td>C-N</td>
<td>0.383</td>
<td>0.321</td>
<td>0.313</td>
<td>0.242</td>
<td>0.267</td>
</tr>
<tr>
<td>C=N</td>
<td>0.443</td>
<td>0.377</td>
<td>0.348</td>
<td>0.179</td>
<td>0.249</td>
</tr>
<tr>
<td>C-O</td>
<td>0.394</td>
<td>0.326</td>
<td>0.353</td>
<td>0.298</td>
<td>0.310</td>
</tr>
<tr>
<td>C=O</td>
<td>0.511</td>
<td>0.446</td>
<td>0.398</td>
<td>0.398</td>
<td>0.356</td>
</tr>
<tr>
<td>C:C</td>
<td>0.459</td>
<td>0.309</td>
<td>0.316</td>
<td>0.176</td>
<td>0.285</td>
</tr>
<tr>
<td>C:N</td>
<td>0.582</td>
<td>0.377</td>
<td>0.348</td>
<td>0.158</td>
<td>0.225</td>
</tr>
</tbody>
</table>

### A.7 Fragmentation

Figure 3: Fragment connectivity for molecule shown in the left panel. Anchor points highlighted; these are the scaffold atoms that are connected to the next fragment.
During fragmentation, we first break all the bonds between rings without breaking fused ring systems. In addition, we also use RDKit and use BRICS to fragment the molecules. A maximum of 8 fragments is used for each molecule; if the number of fragments exceeds 8, we connect the smallest fragments iteratively until the maximum of 8 fragments is reached. Given the fragment connectivity of a molecule (fragment adjacency), we compute all BFS and DFS traversals of the molecule graph based on the fragment. We only use BFS and DFS traversals to avoid computing all of the molecule’s fragmentation graph’s traversals for computational feasibility. A generation order defines how fragments are added step-wise based on their connectivity to make a complete molecule. At each step of the generation order, we save the scaffold atoms and their coordinates, the added fragment at the generation step and its coordinates, the generation step, and the scaffold anchor point for the next fragment. This strategy greatly augments the size of the dataset as well. Figure 3 shows the fragmentation of a random molecule and the fragment connectivity. Figure 4 shows a random order of molecule generation from constituent fragments.

A.8 Anchor point predictor

We trained a standalone neural network (AnchorGNN) to predict the anchor points during sampling. We used graph convolutional layers (GCL) to predict the probability of each scaffold atom being an anchor point. The molecular scaffold and protein pocket atoms are the model inputs. We use one-hot encoded atom types as scaffold atom features. For pocket atoms, the features are atom types, amino
acid types, and whether the atom belongs to the backbone or sidechain. We use inter-atomic squared distance $d_{ij}^2 = \| r_i - r_j \|^2$ as the edge feature.

The update for feature $h$ and coordinates of node $i$ at layer $l$ are computed as follows:

$$m_{ij} = \phi_e(h_i^l, h_j^l, d_{ij}^2), \quad h_i^{l+1} = \phi_h(h_i^l, \sum_{j \neq i} m_{ij}), \quad r_i^{l+1} = r_i^l + \phi_{vel}(r_i^l, h_i^l, i)$$

with $d_{ij} = \| r_i - r_j \|$ and $\phi_e, \phi_h$ being learning functions. We perform a sequence of $l$ Graph Convolutional Layers ($l = 4$). Finally, node embeddings for the scaffold molecule $h^M$ are linearly transformed to a single number and passed through a sigmoid function to compute the probabilities. A binary cross-entropy loss is used to train the model. During sampling, we take the anchor point with the highest probability. A learning rate of $5 \times 10^{-4}$ was used to train this model.