Structure-Based Drug Design Using Distance Prediction With Graph Neural Networks

A thesis

submitted by

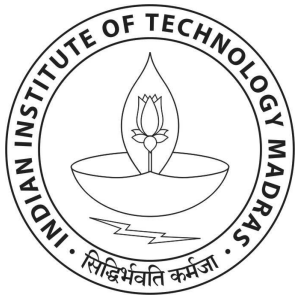
HEMANTH RAM G K

EE18B132

in partial fulfilment of the requirements

for the award of the degree of

BACHELOR OF TECHNOLOGY
in ELECTRICAL ENGINEERING
&
MASTER OF TECHNOLOGY
in DATA SCIENCE



DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

MAY 2023

THESIS CERTIFICATE

This is to certify that the thesis titled **Structure-Based Drug Design using Distance Prediction with Graph Neural Networks**, submitted by **G K Hemanth Ram**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor & Master of Technology**, is a bonafide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Balaraman Ravindran
Research Guide
Professor
Dept. of Computer Science and Engineering
IIT Madras

Prof. Karthik Raman
Research Guide
Professor
Dept. of Biotechnology
IIT Madras

Place: Chennai

Date: 15 May, 2023

ACKNOWLEDGEMENTS

Throughout my time in the institute and this project, I have received a lot of support and assistance from multiple people.

First and foremost, I express my gratitude to **Prof. Balaraman Ravindran** and **Prof. Karthik Raman** for accepting my collaboration request and for their constant support. Their valuable feedback and encouragement played an important role in shaping my ideas and enhancing my knowledge. I would also like to thank **RBCDSAI** for providing access to computing resources.

Additionally, I would like to thank my friends at **IIT Madras** for their invaluable support and contribution to my knowledge and understanding of research and life. Finally, I would like to thank my parents, grandparents, and brother for the support and encouragement they have provided me throughout my years of study and through the process of researching and writing this thesis.

G K Hemanth Ram

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

IITM	Indian Institute of Technology Madras
ML	Machine Learning
DL	Deep Learning
RL	Reinforcement Learning
MLP	Multi-Layered Perceptron
GNN	Graph Neural Network
GCN	Graph Convolutional Network
PPO	Proximal Policy Optimization
MSE	Mean Squared Error
RMSE	Root Mean Squared Error
QED	Quantitative Estimate of Drug-likeness
SA score	Synthetic Accessibility Score

CHAPTER 1

Introduction

1.1 Drug Design and Discovery

Drug discovery is a pharmacological process through which new chemical compounds with desired properties are identified, using a combination of computational, experimental, and clinical models. The process of drug discovery consists of multiple stages. The first stage aims to identify the right biological target molecule (most commonly a malfunctioning protein) that can either be activated or inhibited using an external influence to treat the desired medical condition. The next stage is the screening of millions of chemical molecules to select candidate molecules that show some kind of interaction with the target. Further analysis and optimization of the candidate molecules lead to a smaller set of "lead" molecules. These molecules then undergo extensive optimization to improve potency and pharmacokinetic properties. The pre-clinical stage involves testing on animals and characterization of various properties, post which clinical trials are performed. The entire process takes 12-15 years with a capital of around $2.6 billion for the development of a single drug molecule [1]. With over 10^{60} potential molecules, the chemical space is enormous and discrete. The molecules must also meet a number of desired properties, such as efficacy, stability, safety, low toxicity, etc., thus turning it into a complex multi-objective optimization problem. Successful applications of computational approaches to speed up the process have always been constrained by these critical issues.

1.2 Structure-based Drug Design

Structure-based drug design is the process of designing molecules that bind to a specific protein binding site [2]. The first step involves cloning, purification, and structure determination of the target protein. A large database of compounds is then screened by positioning them on the target site. They are then ranked based on electrostatic and steric interactions with the target site. The lead molecules are then synthesized and

tested for micromolar inhibition in solution. After further optimization of the target and the molecule's structure and orientation to improve potency and interactions, clinical trials are done. Structure-based drug design is one of the most challenging tasks in drug design because of the conformational degree of freedom of both protein and compound structures.

1.3 Molecular Docking

The goal of protein-ligand docking is to explore the binding modes (the pose) of a ligand when it binds to a protein of known structure. The possible poses of ligands generated are evaluated using different kinds of scoring functions that determine the optimal poses. The scoring functions play a major role in the screening of molecules to determine leads in the process of structure-based drug design.

1.4 Structure of the Thesis

The next three chapters form the background for the thesis. Chapter 3 discusses some relevant literature that gives an idea of the current models and algorithms being used. Chapter 4 illustrates an attempt at using reinforcement learning for structure-based drug design, and its drawbacks. In Chapter 5, the distance prediction based generative process is introduced, and the problem of planarization is discussed. The final results of the discussed models are presented in Section 5.6.

CHAPTER 2

Preliminaries

2.1 Graph Neural Networks

Graphs are used to represent different kinds of relational information between entities. Molecules also can be represented as graphs with atoms as nodes and bonds as edges. Graph data is significantly different from its Euclidean counterpart and hence requires a different approach for processing. Graph neural networks are used to perform various tasks on graph data.

Graph Data

In simpler terms, graph data is a collection of objects and relationships among those objects. The objects present in the graph are usually referred to as nodes (or vertices), and the relationships between the nodes are represented by edges. Edges also have properties of their own. Edges in the graph can either be directed or undirected. The edges may also have an associated weight (or cost) value with them. Let a graph **G** contains set of vertices $\mathbf{V} = \{v_1, v_2, .., v_n\}$ and contains set of edges $\mathbf{E} = \{(i, j) : i, j \in V\}$, then the graph is represented as,

$$\mathbf{G} = (\mathrm{V}, \mathrm{E})$$

Usually, graphs are represented by the following two methods:

- Adjacency Matrix: Graphs can be represented in 2-D matrix form. The matrix **A** has dimensions NxN, where N is the total number of nodes present in the graph. Each entry of the matrix, $A[i][j]$, denotes the status of the edge from node i to node j (there exists an edge from node i to j if $A[i][j]! = 0$). If every edge of a graph is undirected then the adjacency matrix would be symmetric.
- Adjacency List: Graphs can also be represented using an array of lists. Each element of the array is a list containing nodes connected with that particular node (i.e.) i^{th} element of the Adjacency List contains a list which has all the nodes with which the i^{th} node has an edge with.

Learning on Graph Data

A different kind of approach must be adopted while working with graphical data as they are structured differently when compared with traditional data, used for machine learning and deep learning. The relationship between various nodes in the model should be taken into account. Graph Neural Networks (GNN) are used for processing such data.

Now, let us consider a graph **G** = (V,E) with undirected and unweighted edges. Let N be the number of vertices/nodes and M be the total number of edges and each vertex is denoted by a feature vector x ($x_i \in R^d$ denote the feature vector of i^{th} node). Concatenation of all the feature vectors be $X = [x_1^T, x_2^T, .., x_N^T] \in R^{N \times d}$.

The main idea behind a GNN is to change each node's current state, represented by x_i into a hidden state, $h_i^{(K)}$, with potentially varying the dimensions, so that the new state captures the data about the surrounding area and the node's connection to the rest of the graph. Through the concepts of message forwarding and neighbourhood aggregation, this is accomplished iteratively.

At k^{th} iteration, the message $m_i^{(k)}$ is computed for each node $i \in V$, as follows

$$m_i^{(k)} = f^{(k)}\left(h_i^{(k)}\right)$$

where $h_i^{(k)}$ us the hidden state of node i in the k^{th} step, and $f^{(k)}$ is a function (parametrized linear/non-linear function) that computes the message given a hidden state of a node at the k^{th} step. The next step is to aggregate the messages across neighbours of a node, as follows

$$s_i^{(k)} = g_{AGG}^{(k)}\left(\left\{m_j^{(k)} : j \in \mathcal{N}(i)\right\}\right) \forall i \in V$$

In other words, the aggregator function $g_{AGG}^{(k)}$ is used to carry the aggregated messages from the nodes that are directly adjacent to the node i. The aggregation function is typically selected to be invariant to the permutation of the nodes in the graph, meaning that it is independent of the input order of the nodes (e.g., summation or mean). $\mathcal{N}(i)$ denotes the node's immediate neighborhood. The messages from the source node and the aggregated entity are fed into the nonlinear function $\sigma(.)$, which updates the nodes' hidden state.

$$h_i^{(k+1)} = \sigma\left(m_i^{(k)}, s_i^{(k)}\right) \forall i \in V$$

To put it another way, $h_i^{(k+1)}$ aggregates the messages from the source node i and mixes them with the message of itself to update the node's hidden state. Initially, in most of the cases, when $k = 0$ then $h_i^{(0)} = x_i$. The hidden state $h_i^{(k)}$ would have gathered data from the node i's k^{th} neighborhood at some random step k. We finally arrive at the hidden representation $h_i^{(K)} \forall i \in V$ after k iterations.

In the recent five to six years, a wide range of GNNs have been developed to address various graph problem types. Graph Convolutional Network (GCN), Graph Attention Network (GAT), and Message Passing Neural Network (MPNN) are a few of the most well-known graph networks.

2.2 Reinforcement Learning

Reinforcement learning is an area of machine learning that deals with sequential decision-making wherein an agent is trained to take actions with associated rewards in an environment. The goal of the agent is to maximize the cumulative reward and it does so by reinforcing itself through experience and subsequently picking better actions.

Markov Decision Process

To solve reinforcement learning problems, one of the important assumptions made is the assumption of the Markov decision process (MDP). An MDP (environment) is characterized by the following:

- set of states **S**
- set of actions in each state **A**
- state transitions given an action **T**
- reward for the state transitions **R**

The assumption of a Markov decision process is that rewards and the transition probabilities depend only on the current state and not on the historical states. We have to find an optimal policy $\pi(a|s)$ that would maximize the cumulative reward.

Approaches

Reinforcement learning algorithms can be broadly divided into the following categories:

- **Value-based** algorithms involve optimizing the value function of each state $V^{\pi}(s)$, which is the expected return following the policy π starting at state s. Q-learning and value iteration methods are examples.
- **Policy-based** algorithms involve optimizing the policy π and coming up with the optimal $\pi(a|s)$ directly, which is the action to be performed at each state s, and example being the REINFORCE algorithm.
- **Actor-critic** algorithms combine the above methods by using training an actor to optimize policy and a critic to evaluate the state values simultaneously. Proximal policy optimization and soft actor-critic are examples.

Actor-critic Algorithms

The base for all the policy gradient and actor-critic algorithms is the policy gradient theorem which gives an expression for the derivative of the performance $J(\theta)$.

$$\nabla J(\theta) \propto \sum_{s} \mu(s) \sum_{a} q_{\pi}(s,a) \nabla \pi(a|s,\theta) \tag{2.1}$$

where μ is the state distribution and $q(s,a)$ is the Q value for state-action pairs. After changing into expectations and adding scaling factors, we get:

$$\nabla J(\theta) = \mathbb{E}_{\pi}[G_t \frac{\nabla \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}] \tag{2.2}$$

where G_t is the return at step t. After adding a baseline and using a separate approximator to calculate the state value, we get performance $J(\theta)$ in the actor-critic model as:

$$J(\theta) = (R_{t+1} + \gamma v(S_{t+1},\omega) - v(S_t,\omega)) log(\pi(A_t|S_t,\theta)) \tag{2.3}$$

where ω is the set of parameters used to estimate the state values. Both θ, which constitute the actor's (policy) parameters, and ω, which constitute the critic's parameters,

are updated simultaneously in each iteration.

Proximal Policy Optimization (PPO)

PPO is a type of policy gradient algorithm which aims to improve the training stability of the policy by limiting the change in the policy in each iteration. We prefer to avoid rapid updates in the policy as it might lead to a bad policy. In PPO, we see how much the policy has changed when compared to the previous iteration. We then clip this ratio in a range $[1-\epsilon, 1+\epsilon]$ and prevent the policy from changing a lot (keeping it *proximal* to the previous policy). The modified objective function in PPO is:

$$L^{CLIP}(\theta) = \mathbb{E}[min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)] \tag{2.4}$$

where A_t is the advantage function which is the difference between the return calculated from the step reward and the predicted state value. $r_t(\theta)$ is the importance ratio which is the ratio between the probability under the new and old policies.

2.3 Context Encoder

The molecular data we have in hand has atom type, bond, and spatial information of atoms. We model the molecules as fully-connected graphs with the atoms as nodes and the pairwise distance between the atoms as the edge weights (Note: bonds are not taken into account). Embeddings for the nodes have to be generated. To do so, an MPNN (message-parsing neural network) is employed. To make the embeddings rotation and translation invariant, only the pairwise distances between atoms are taken into account, and not their absolute locations.

Instead of directly using the pairwise distances, the distances are Gaussian-smear to get the distance feature vectors. This is then passed through an MLP to get a hidden layer which is then used as the weights when aggregating messages in the MPNN. For instance, the initial embeddings are h_i^0 for the i^{th} node. Specifically, the formula of message passing takes the form:

$$\mathbf{h}_i^{l+1} = \sigma\Big(\mathbf{W}_0^l \mathbf{h}_i^l + \sum_{j \in N_k(i)} \mathbf{W}_1^l \mathbf{w}(d_{ij}) \odot \mathbf{W}_2^l \mathbf{h}_j^l\Big) \tag{2.5}$$

In the above expression, d_{ij} represents the distance between the i^{th} and j^{th} atom and $N_k(i)$ represents the k nearest neighbors of the i^{th} node. The $\mathbf{w}$ represents the Gaussian-smearing, and the $\mathbf{W}_1^l$ represents the MLP for the distance feature in the l^{th} message parsing layer.

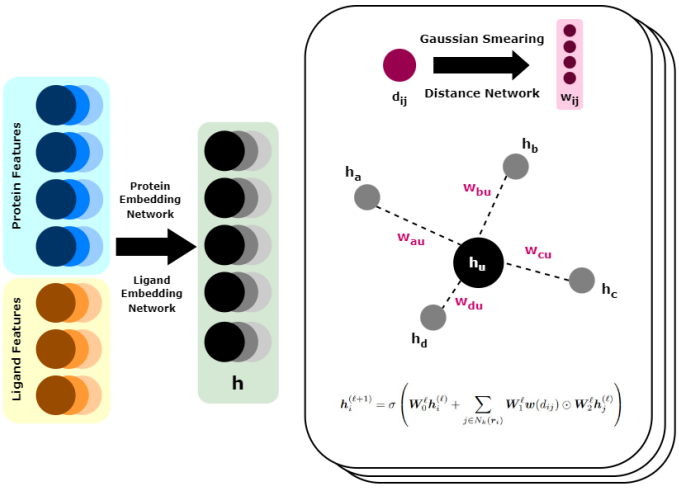


Figure 2.1: Visualization of the context encoder

2.4 Datasets and Tools

Following are some databases that are widely used in training drug discovery models:

- **ChEMBL** is a curated database of biologically active and drug-like chemical molecules.
- **Protein Data Bank (PDB)** is a large database containing the three-dimensional structures of multiple proteins.
- **PDBbind** provides a collection of experimentally measured binding affinity data of biomolecular complexes in PDB.
- **BindingDB** contains binding affinity data of drug-like molecules with proteins.
- **CrossDocked2020** [3] contains 22.5 million docked poses of protein-ligand pairs as specified by *Pocketome*, which groups structurally similar proteins and ligands into *pockets*. Complexes in the same *pocket* are then cross-docked.

Useful tools and modules which are used are listed below:

- **RDKit** [4]: Open source chemoinformatic toolkit.
- **OpenBabel** [5]: A chemical toolbox.
- **PyTorch** [6] [7]: To build and train the models required.
- **AutoDock Vina** [8]: A molecular docking software program used in computational chemistry to predict how small molecule ligands bind to receptors.

CHAPTER 3

Literature Survey

Recent advancements in data-driven and machine-learning algorithms are playing a major role in the development of various aspects of drug discovery. Most of the approaches use deep generative models that are either SMILES/string-based or graph-based. Very few approaches that have been proposed, take into account spatial information of the concerned molecules into account. Following are literature based on molecule generation, property prediction, and structure-based drug design depicting different ways to model molecules and evaluation metrics.

3.1 GCPN - Graph Convolutional Policy Network [9]

This approach integrates the idea of the graph convolutional network with proximal policy optimization, a policy gradient algorithm used in reinforcement learning. The graph generation process is posed as an MDP (Markov Decision Process) where the state at each step is the graph generated so far, and the action is the addition and removal of bonds and atoms. This also uses adversarial training through a GAN to model the stepwise rewards to incorporate prior knowledge about the molecules in the dataset. The valency and bond constraints are also imposed through the stepwise rewards. The pharmacological properties are computed as the final reward. A graph convolutional network is used as the policy, and proximal policy optimization is used for training.

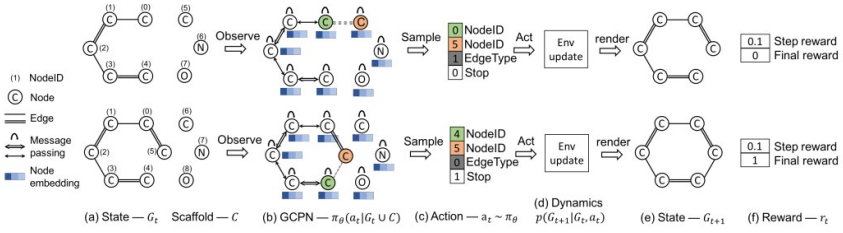


Figure 3.1: Overview of the iterative process proposed by 3.1 [9]

3.2 3-Dimensional Generative Model [10]

Structure-based drug design is posed as an iterative generative process in which an atom is added into the 3-dimensional space at each step. Bonds are generated using OpenBabel and not the model after the atoms have been placed. The model uses only the pairwise distance between atoms and hence is rotation and translation invariant. The are two components in the model. The context encoder is a graph convolutional network to generate encodings for the atoms from the structure using pairwise distances and atom features.

The spatial classifier predicts the density of each type of atom at each point in the space using the encodings computed and the distances of the point in consideration to the neighboring atoms. Starting with the target protein as the context, ligand atoms are added using auto-regressive sampling. A frontier network is employed to determine termination. To train the model, atoms in the ligand are masked, and the model is used for prediction. The loss function consists of the log losses, which check if the model places the atom at the right place and is of the right type.

$$L_{binary\_entropy} = -\mathbb{E}_{\mathbf{r}\sim p+}[log(1 - p(Nothing|\mathbf{r}, C))] - \mathbb{E}_{\mathbf{r}\sim p-}[log(p(Nothing|\mathbf{r}, C))] \quad (3.1)$$

$$L_{categorical} = -\mathbb{E}_{(e,\mathbf{r})\sim p+}[log(p(e|\mathbf{r}, C))] \quad (3.2)$$

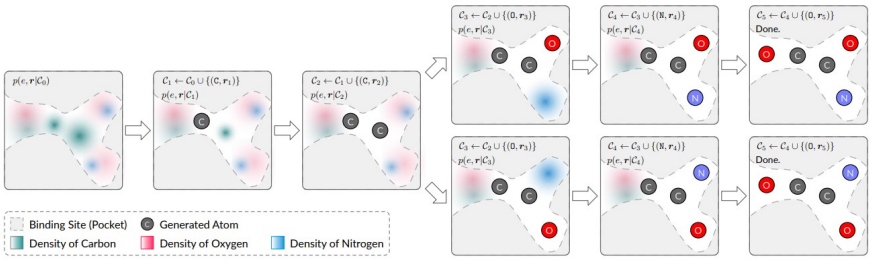


Figure 3.2: Visualization of the auto-regressive sampling proposed in 3.4 [10]

3.3 ChemProp [11]

An approach using Directed Message Parsing Neural Networks (directed NMPNNs) for the prediction of different chemical properties of molecules from their structures

is proposed here. The directed MPNN proposed performs better than the conventional MPNN by overcoming the totter's problem. The key difference between directed MPNNs and conventional MPNNs is the computation of edge-wise embeddings rather than node-wise embeddings.

3.4 Pafnucy Model [12]

A deep learning model for protein-ligand binding affinity prediction. The molecules are treated as 3-dimensional images where each pixel will have 19 channels with the required features of the atoms and bonds. A 3-dimensional convolutional neural network is employed for the prediction. Data was augmented with rotated versions of the structures to impose translational and rotational invariance.

3.5 Continuous Representation of 3D Molecular Structures [13]

A model that generates 3D structures of molecules. The molecule is modeled as atom probability densities in the 3-dimensional space. Every atom is treated as a center of 3-dimensional Gaussian distribution. A variational auto-encoder model is employed to generate new atom distributions in space. A novel fitting algorithm is then used to fit the continuous densities to a discrete molecular arrangement. OpenBabel is then used to add bonds to the structure.

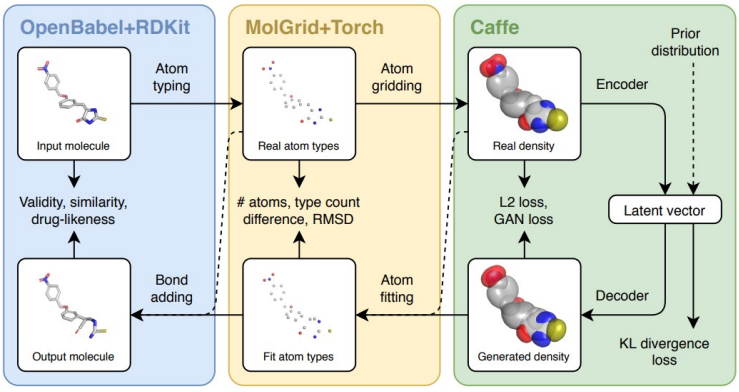


Figure 3.3: Visualization of the continuous representation proposed in 3.5 [13]

CHAPTER 4

A Reinforcement Learning Approach : An Attempt

The problem at hand is to generate molecules with some optimized chemical properties (like binding affinity) and ensure that the agent learns the structural patterns shown by the molecules in the data set (incorporating prior knowledge). The agent is expected to follow rotation and translation invariance. The former can be imposed through the formulation of the stepwise and final rewards, and the latter through the design of the models being used.

4.1 Why Reinforcement Learning ?

When compared to learning a generative model across a dataset, a reinforcement learning approach to molecule generation offers a number of benefits. First off, because they are complicated and non-differentiable, desired molecular characteristics like drug-likeness and molecule restrictions like valency cannot be easily included into the objective function of graph generative models. Contrarily, through the design of the environment dynamics and reward function, reinforcement learning is capable of explicitly imposing hard limitations and desired qualities. Secondly, active exploration of the molecular space outside of the samples in a dataset is made possible by reinforcement learning. Alternative deep generative model approaches show promising results in reconstructing given molecules, but their exploration ability is restricted by the training dataset.

4.2 The Environment

Drawing inspiration from here, only atoms are taken into consideration in the generation process, and the bonds will be added using OpenBabel after the atoms have been placed in space appropriately.

The 3D space in consideration is partitioned into cells (like a grid), and in each cell, an atom of any type can be placed. The state space is now the set of all possible

arrangements of atoms. The initial state will be the arrangement of atoms in the binding site specified. As atoms in the ligand are generated, the state changes and the atoms of the ligand also get added. The appropriate features of the atoms of the binding site and the ligand are also part of the state.

At any state, the action space consists of the addition of any type of atom in any empty cell of the 3D grid in consideration. So, in each step, any empty cell in the grid can be chosen and any kind of atom can be added to it.

4.3 Data

The dataset and the features used are being drawn from here. A refined subset of the CrossDocked2020 dataset (around 25,000 data points) which is filtered according to the binding pose RMSD of the docked pairs, is used. The dataset provides protein-ligand pairs with the coordinates of the atoms and the bonds. Depending upon the location of atoms, for every protein atom, the following features are considered :

- The element of the atom (atomic number)
- 3D coordinates of the atom
- Whether the atom belongs to the backbone (N, C or α-C)
- Amino acid to which the atom belongs, in the chain

Similarly, for ligand atoms, the following features are considered:

- The element of the atom (atomic number)
- 3D coordinates of the atom
- Atom family information (from RDKit)

4.4 The Agent

Given the atoms of the binding site and the set of atoms added previously, the agent has to select where to add an atom in the 3D grid and then choose the type of atom. To do so, the probability density of each type of atom in the space is estimated, as done here. In order to make the model rotation and translation invariant, only the relative distances are taken into account, so that structural information is also considered.

Atom Embeddings

On observation, the feature dimensions for atoms of protein and ligand are different. To come up with embeddings of the same dimension for further processing, 2 fully connected hidden layers are introduced. These are now used as embeddings for atoms.

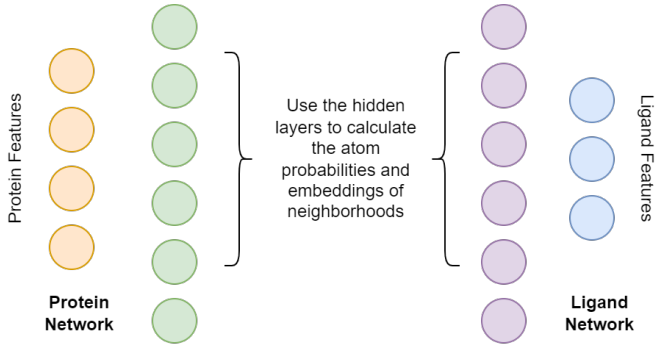


Figure 4.1: Visualization of computing atom-embeddings

The Model

Given the receptor atoms and the atoms added so far, a fully-connected graph is constructed with them as the nodes and the pair-wise distances as the edge weights. A context encoder, as discussed here, is then used to get the embeddings for the atoms. Now, given a point in space, the k nearest atoms to this point and their embeddings are considered. Again Gaussian-smearing the distance of the atoms to the point and using the distance feature as weights to aggregate the embeddings, an embedding for each point in space is obtained. A separate network then processes this to predict the probability density of each type of atom. After the probability densities are calculated, the type with the maximum probability gets added to the appropriate cell with the maximum probability.

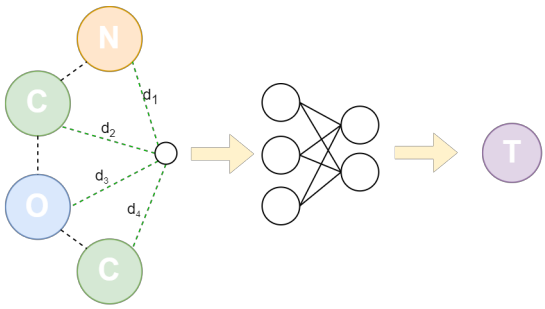


Figure 4.2: The model

Training

The above-described model can be considered to be the actor. A new context encoder can be used to generate embeddings to be used for computing the value of the state. The embeddings of the ligand atoms can be aggregated and passed through an MLP to get the value of the state. The above-mentioned components now constitute the critic. Actor-critic algorithms Monte-Carlo actor-critic and PPO can now be used to train the agent.

Generation of Bonds

The termination is predicted separately by an MPNN and once all the atoms have been placed, the spatial arrangement of the ligand atoms is fed to OpenBabel, which gives us the optimal bonds between the atoms.

4.5 The Reward Structure

Neighborhood

The neighborhood of an atom is defined as the collection of the k-nearest atoms in the space to it. The protein and ligand atoms in the neighborhood are considered separately. The embedding of the neighborhood is then defined as an aggregate of the features of the atoms in the neighborhood. The aggregate can be a weighted mean of the features with the distance of the neighborhood atoms as the weights so that the structure is taken into account. Then, the neighborhood embedding for all the ligand atoms in the dataset is calculated.

Neighborhood based step-wise reward

The agent is expected to learn the structural patterns of the molecules in the dataset through step-wise rewards. The agent will be rewarded based on how similar the neighborhood of the atom it placed in its previous action, is, to the neighborhoods of that

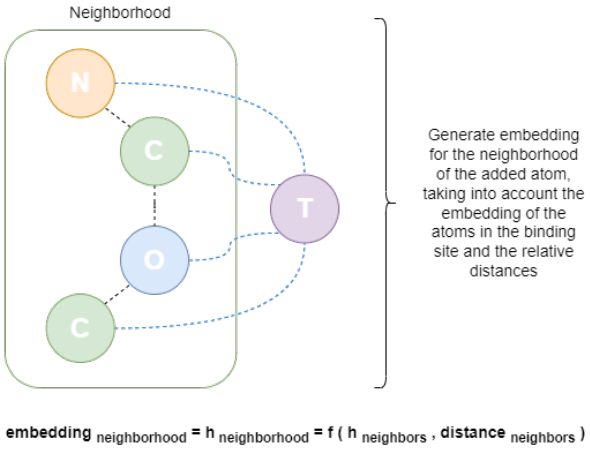


Figure 4.3: Visualization of neighborhood

type of atom in the dataset. After taking an action and placing an atom in the grid, the embedding of the placed atom's (T for target atom) neighborhood is calculated. The step-wise is then computed by comparing this embedding with the neighborhood embeddings of that type of atom in the dataset using some similarity metric (like cosine similarity, mean squared distance).

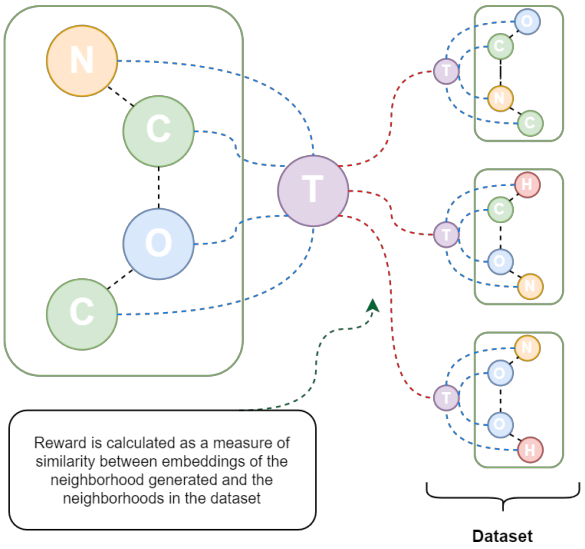


Figure 4.4: Visualization of neighborhood-based step-wise reward

Density based step-wise reward

The ligand atoms are considered as the center of 3D Gaussian distributions, and the density score is calculated as the Gaussian value of the closest atom. The density score provides an estimate of how close to the point chosen an actual ligand atom is present.

Once the agent chooses a point and the type of atom, the step-wise reward is calculated as the density score, calculated using the ligand atoms, of the type chosen.

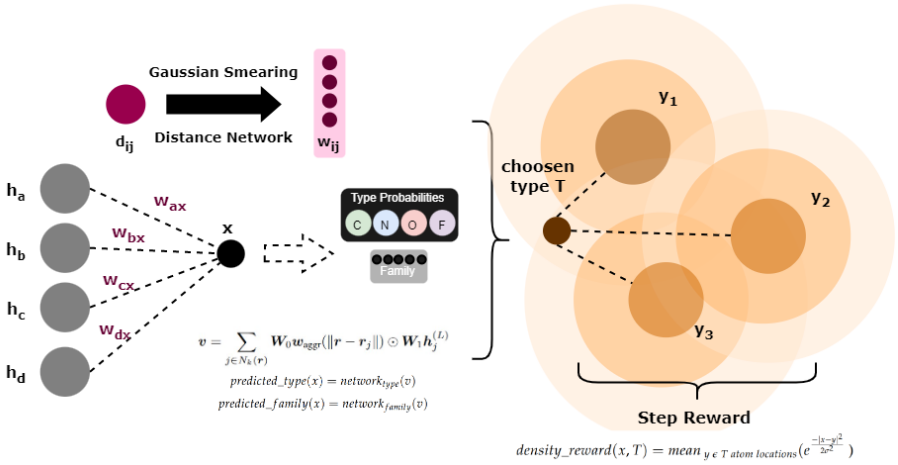


Figure 4.5: Visualization of density-based step-wise reward

Final Reward

To formulate the final reward, the scoring functions used for protein-ligand interactions in molecular docking [14] are assessed. Following are the types of scoring functions used:

Physics-based scoring functions include the scoring functions based on force-field, solvation models, and quantum mechanics methods. Physics-based SFs take into consideration enthalpy, solvation, and entropy and are hence appropriate to compute the binding free energy of the complex with relatively greater predictive accuracy.

Empirical scoring functions employ linear regression analysis over important energetic factors for protein-ligand binding like hydrogen bonds, hydrophobic effects, steric clashes, etc. to estimate the binding affinity of the complex.

Knowledge-based scoring functions estimate the desired pairwise potentials based on the inverse Boltzmann statistic principle from a large set of protein-ligand complexes. The advantage of these SFs is the balanced trade-off between computing cost and accuracy.

Apart from the classical ones mentioned, machine learning-based methods like the Pafnucy model, which directly predicts binding properties, are also in use. A combination of computationally feasible metrics mentioned can be used as the final reward.

4.6 Results

The density score based reward system was used for the step-wise rewards, and for the final reward, the sum of the difference between the number of atoms in the generated molecule and the actual ligand for each type was used. The Monte-Carlo actor-critic algorithm was used to train the agent.

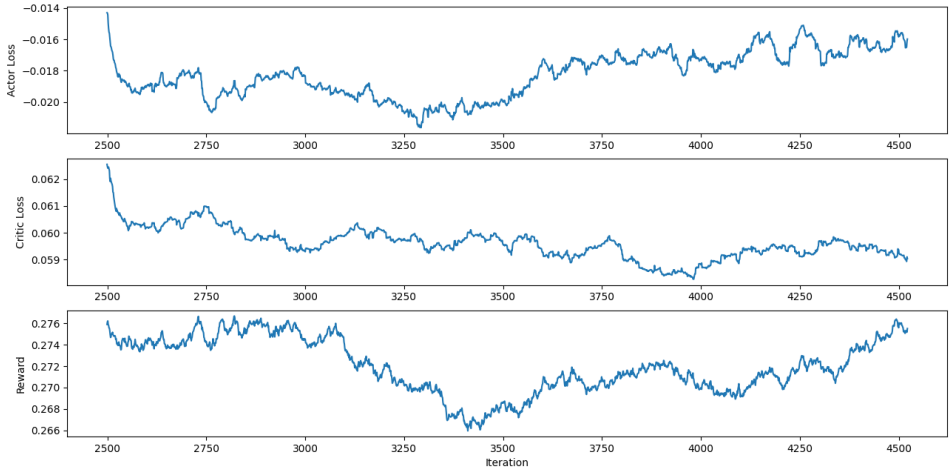


Figure 4.6: Training curve of Monte-Carlo actor-critic

The training was found to be very slow, and the atoms were getting placed very close to each other even after 10,000 iterations. This can be attributed to the very large action space. Even with a resolution of 0.05, the number of points in space is of the order of 10^6, and at each point, any type of atom can be placed. One additional observation was that only carbon atoms were being chosen, irrespective of the location, after a few iterations. The bias towards carbon was primarily because of the frequent occurrence of carbon. Carbon comprises around 70% of all the atoms in the dataset.

What next ?

The large action space, which was a result of the definition of the environment, is the primary reason for the poor performance. Different definitions of the action space or splitting the action into sub-actions might help overcome this problem. The main problem lies in the prediction of location because of the large number of points.

Instead of predicting the location and type together, the selection of location and type can be decoupled. The selection of location can be treated as a supervised learning problem in a variety of ways. But once decoupled, there is no point in treating the selection of the type of atom as a reinforcement learning problem as it is a classification problem in an obvious way.

So, either the definition of the environment has to be changed such that the size of the action space is reduced or the selection of location and type should be treated as supervised learning problems to improve performance. Moving on, the latter will be followed. The selection of location and type will be decoupled. The location of the atom to be added is first predicted, and then the type is predicted, given the location. The first part is analyzed as a supervised learning problem in two different ways, and the second part is considered as a classification problem.

CHAPTER 5

Distance Prediction using Graph Neural Networks

In the previous section, it was seen that reinforcement learning with the defined environment led to very slow training and hence proved ineffective. Moving on, the framework used will be the same, but supervised learning will be used, and the prediction of the location and the type of atom will be dealt with separately.

Framework

Given the atoms of the binding site and the spatial information, the ligand is generated by adding ligand atoms sequentially in the space. To do so, taking the binding site atoms and the ligand atoms added so far as the context, the location and type of the next ligand atom are predicted. Separate models are employed for the prediction of location and type. The location is first predicted by a model, and then the type is predicted, assuming the location is known by a different model. Along with the type, the termination probability is also predicted to determine termination. Note that the bonds of the molecules are not considered. The bonds are added using the location of the ligand atoms added, with the help of OpenBabel and RDKit.

5.1 Type Prediction

The prediction of the type of atom takes as input the location where the new atom is being added and the current context. A feature vector is generated from the input to predict the type. It contains the following components:

- Aggregate of the ligand atom features
- Aggregate of the surrounding protein atom features
- Encoded spatial information of protein atom features

The first component is to account for the number of each type of atom added so far. The second component is to give an estimate of what the neighborhood of the ligand molecule generated so far looks like. The third component is generated by using a spatial encoder. To do so, the distances of the point chosen, to the k nearest protein atoms are computed. The distances are Gaussian smeared and passed through an MLP to get the distance feature vectors. These are then used as weights to aggregate the protein atom features. The aggregate is finally passed through an MLP to get the third component.

The feature vector generated using the three components by concatenating them is then passed through an MLP with a softmax output layer to get the atom-type probabilities. To predict termination, only the first two components are concatenated and passed through an MLP with a sigmoid output layer.

Training

The ligand molecules in the dataset are partially built, and the location of the next closest atom is given as input to the model. The atom-type probabilities are obtained from the softmax output. The loss is the *log* of the softmax output of the actual type of the atom. For termination, the binary cross entropy loss function is employed for training. The training curve is shown in Figure 5.1.

$$L_{type=k} = \log\ (\ softmax\_output_k\ ) \tag{5.1}$$

$$L_{termination} = -\log(t)\ if\ termination\ else\ \ -\log(1-t) \tag{5.2}$$

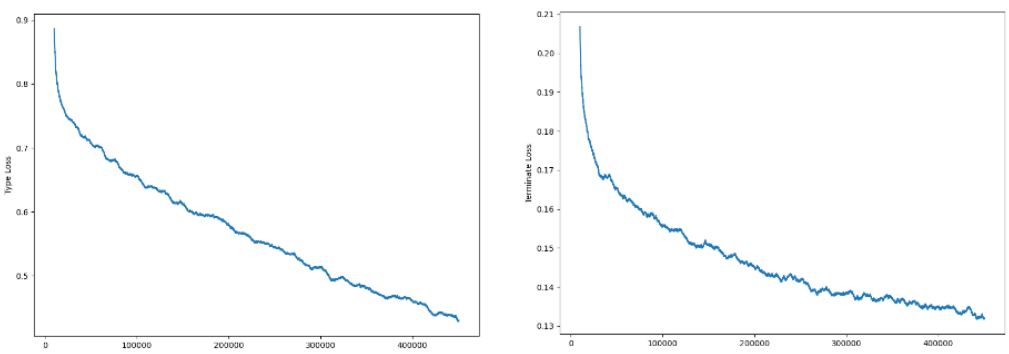


Figure 5.1: Training curve for type prediction

5.2 Density Prediction for Location

The ligand atoms in the actual molecule are considered as the center of 3D Gaussian distributions, and the density score is calculated as the Gaussian value of the closest atom as discussed here. In this method, the density score is predicted, and the point with the maximum predicted density score is chosen. To do so, first, a context encoder as discussed here is used to generate embeddings for the atoms. To predict the density score at a point, similarly, the distances of the point to the k nearest neighbors are used to generate distance features. These are then used as weights to aggregate the context encoder's embeddings and finally passed through an MLP to get the predicted density score.

Training

The ligand molecules in the dataset are partially built, and the density score at each point in space is calculated using the remaining ligand atoms. The points are sorted in ascending order of the score, and every 10 points (10% of all points to reduce computation) are taken. The model is then used to calculate the predicted scores at these points. The RMSE of the density scores with the predicted density scores is used as the loss function for training. The training curve is shown in Figure 5.2.

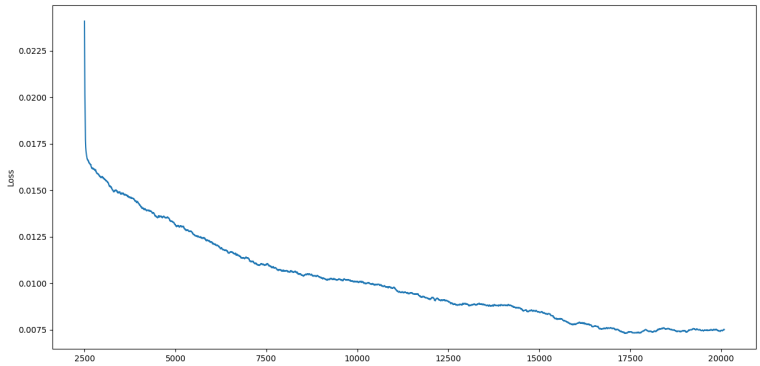


Figure 5.2: Training curve for density prediction

Results

To judge the performance of the model, the distance of the point chosen by the model to the nearest present ligand atom is computed. Figure 5.3 shows the distribution of this distance. It is observed that most of the points are 1.5-3 Åfrom an actual ligand atom (bond lengths vary from 1.3-1.8 Å). This is a slightly significant difference and should be improved.

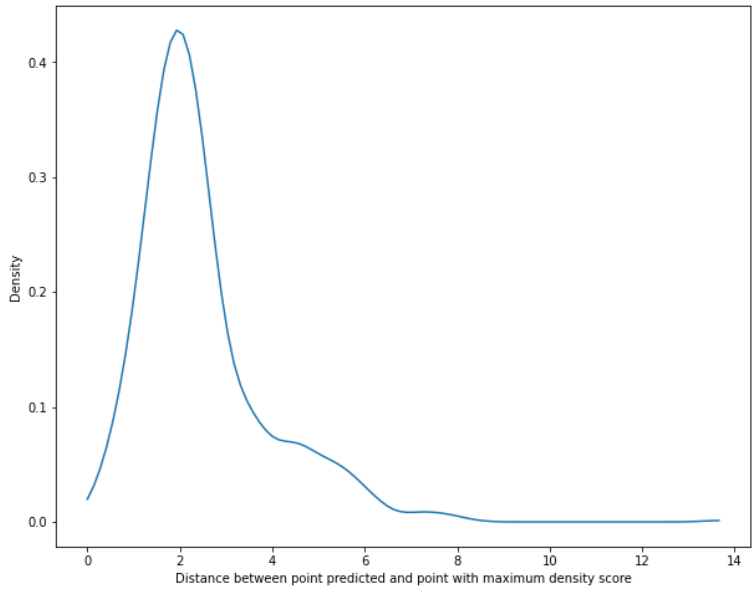


Figure 5.3: Closest distance distribution for density prediction

5.3 Distance Prediction for Location

As mentioned before, the ligand atoms are added sequentially to build the molecule. To find the location of the atom, the distances of the new atom from the previously added ligand atoms are predicted. The point in space which satisfies these distance constraints with the least error (RMSE of the distances) is chosen as the location for the next atom. To predict the distances, the embeddings generated by the context encoder are used. The embeddings are passed through an MLP to get the predicted distances. In this method, there is no necessity to look at all the points in space. Once the distance constraints are obtained, the RMSE of a point $r(x, y, z)$ can be optimized to obtain the coordinates. The following figure illustrates the method.

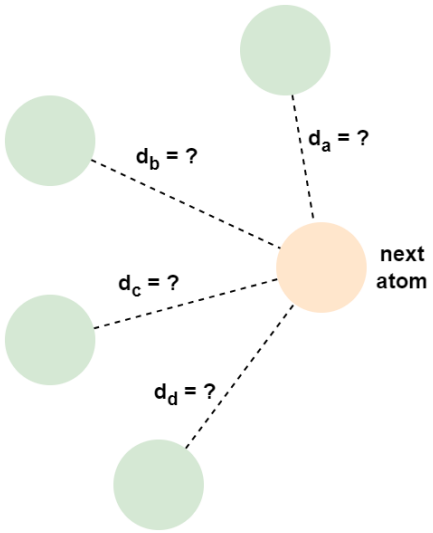


Figure 5.4: Illustration of distance prediction

Training

The ligand molecules in the dataset are partially built, and distances of the next nearest ligand atom to the previously added atoms are computed. The model then computes the predicted distances. The MSE of the actual distances and the predicted distances is used as the loss function to train the model. The training curve is shown in Figure 5.5.

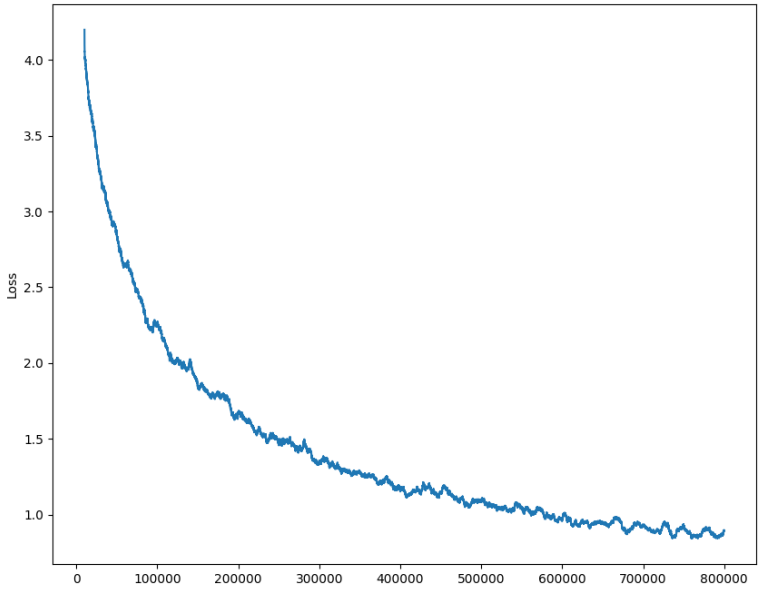


Figure 5.5: Training curve for distance prediction

Results

To judge the performance of the model, the distance of the point chosen by the model to the nearest present ligand atom is computed. Figure 5.6 plot shows the distribution of this distance. It is observed that most of the atoms are 0.5-2 Åaway from an actual atom. This is significantly better than the results of using density prediction. Moving on, the distance prediction to choose the location of the next atom will be employed in the process.

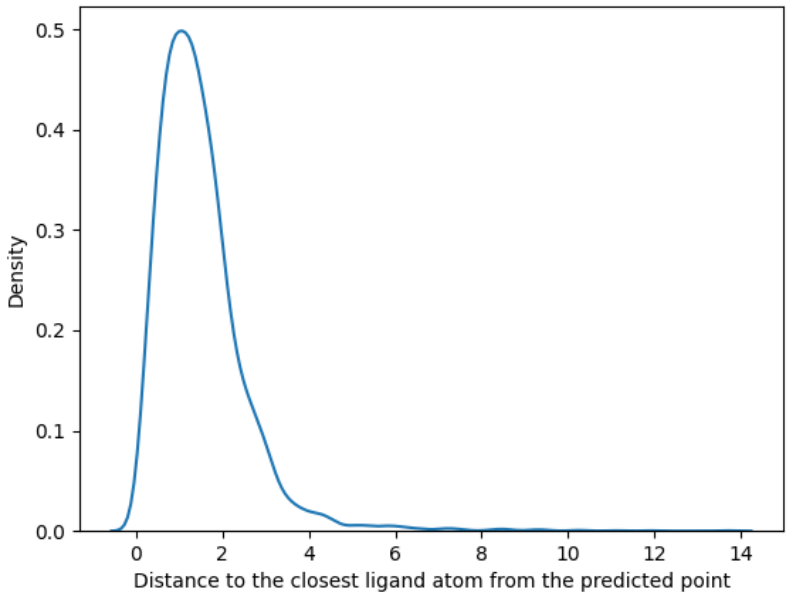


Figure 5.6: Closest distance distribution for distance prediction

5.4 Evaluation Metrics

It is necessary that a drug candidate is optimized in terms of several biological and chemical properties. There are multiple ways to computationally evaluate the properties of the molecules generated by a deep learning model. These metrics help characterize the performance of the algorithms in generating molecules that exhibit drug-like properties.

Octanol-water partition coefficient (logP)

The octanol-water partition coefficient (P_{OW}) is a measure of the drug's hydrophilicity and lipophilicity. It is defined as the ratio of the concentration of a solute in a water-saturated octanolic phase to its concentration in an octanol-saturated aqueous phase.

$$P_{OW} = \frac{[X]_{octanol}}{[X]_{water}} \tag{5.3}$$

And logP is defined as the *log* value of P_{OW}. Although logP is an experimentally deduced quantity, computational methods to estimate the quantity based on the contributions of functional groups and fragments of a molecule have been derived [15]. A lower value of logP is generally favorable as it indicates that the molecule is polar and hydrophilic. It is usually considered that molecules with a logP value greater than 1 or less than 4 are more likely to have optimal drug-like properties.

Quantitative Estimate of Drug-likeness (QED)

The quantitative estimate of drug-likeness (QED) [16], which was proposed in 2012, is an index of drug-likeness modeled using information available on marketed drugs and is widely used in current small-molecule drug discovery for computational methods and to evaluate drug-like properties. The QED index models these properties using data available from 771 orally administered drugs already approved by the U.S. Food and Drug Administration (FDA). QED values lie between 0 and 1, with a higher value indicating better drug-likeliness.

Synthetic Accessibility Score

Synthetic accessibility defines the ease of synthesis of chemical compounds. It is characterized by a metric called synthetic accessibility score [17] (SA score), which can be computationally determined in a rule-based manner. A lower SA score indicates easier synthesizability. Compounds that have a score of less than 4.5 are generally considered easy to synthesize.

Binding Affinity

Apart from evaluating the generated molecules as a separate entity, it is also necessary to evaluate the receptor-ligand binding properties. The binding affinity of the receptor with the generated ligand is calculated after performing docking using AutoDock Vina. It uses a scoring function to estimate the binding affinity between the small molecule and the target protein, and it also employs a search algorithm to explore the conformational space of the ligand and receptor and find the best possible binding pose. It has also been validated against experimental data and shown to perform well in predicting ligand binding affinities and binding poses.

5.5 Planarity

Aromaticity is a very common phenomenon in chemical molecules, and almost all the molecules in the dataset have at least one aromatic ring present in them. One of the spatial constraints for a ring to be aromatic is planarity. Only if all the atoms in the ring are planar can aromaticity exist. But, many of the molecules generated by the model lacked aromaticity, primarily because atoms were not in the same plane. Rings were also not formed due to a lack of planarity in some cases. The atoms were almost in the same plane in many cases. The presence of aromaticity also results in better chemical properties at times. The SA score particularly improves most of the time, while the QED gets better not so frequently.

Figure 5.7 shows molecules generated by the model being planarized using RDKit, along with the evaluation metrics mentioned. It is observed that the SA score reduces and the QEDs increase. So, it can be concluded that planarization will indeed improve the quality of molecules generated occasionally. Two different ways are proposed to impose planarity in the molecules generated.

1. Planarization after generation
2. Addition of atoms in a plane

The former involves using RDKit to planarize the molecule after generation. The latter is the addition of atoms in a plane defined by the first three ligand atoms added. The point in the plane which best satisfies the distance constraints in that plane is chosen.

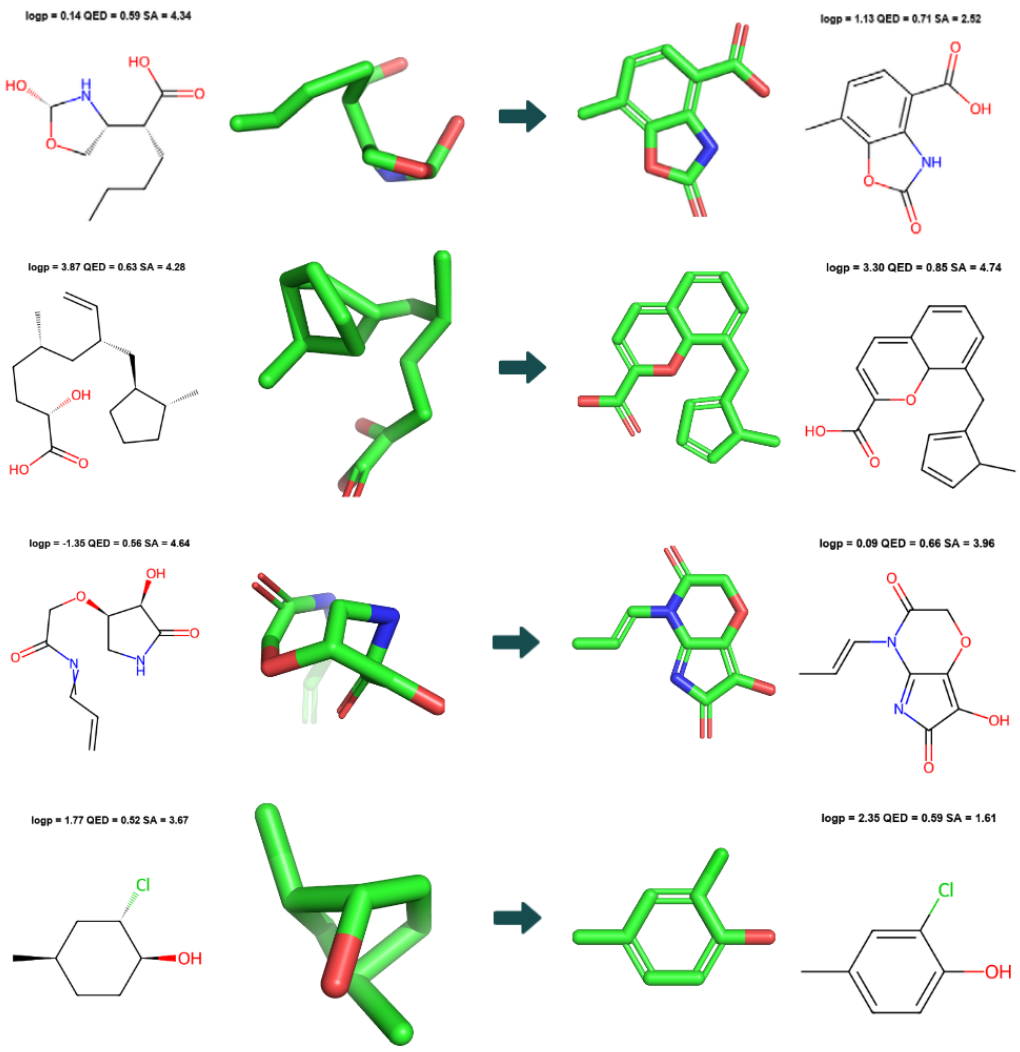


Figure 5.7: Examples of planarizing molecules

5.6 Results

The distance-based prediction is used to choose the location of the next atom to be added. The type prediction model discussed is used for predicting the type and termination. The first three atoms are added before generation to give the model prior information about the receptor site. This is also useful when adding atoms in a plane since the plane is implicitly defined by the first three atoms. The addition of the first three atoms is in itself a different problem and will not be dealt with for now.

It was found that the molecule at termination didn't necessarily have better properties than the intermediate molecules. So, the intermediate molecule which has the highest QED is chosen as the final molecule. In all the analyses done, only generated ligand molecules that have at least 7 atoms are considered in order to avoid too small molecules.

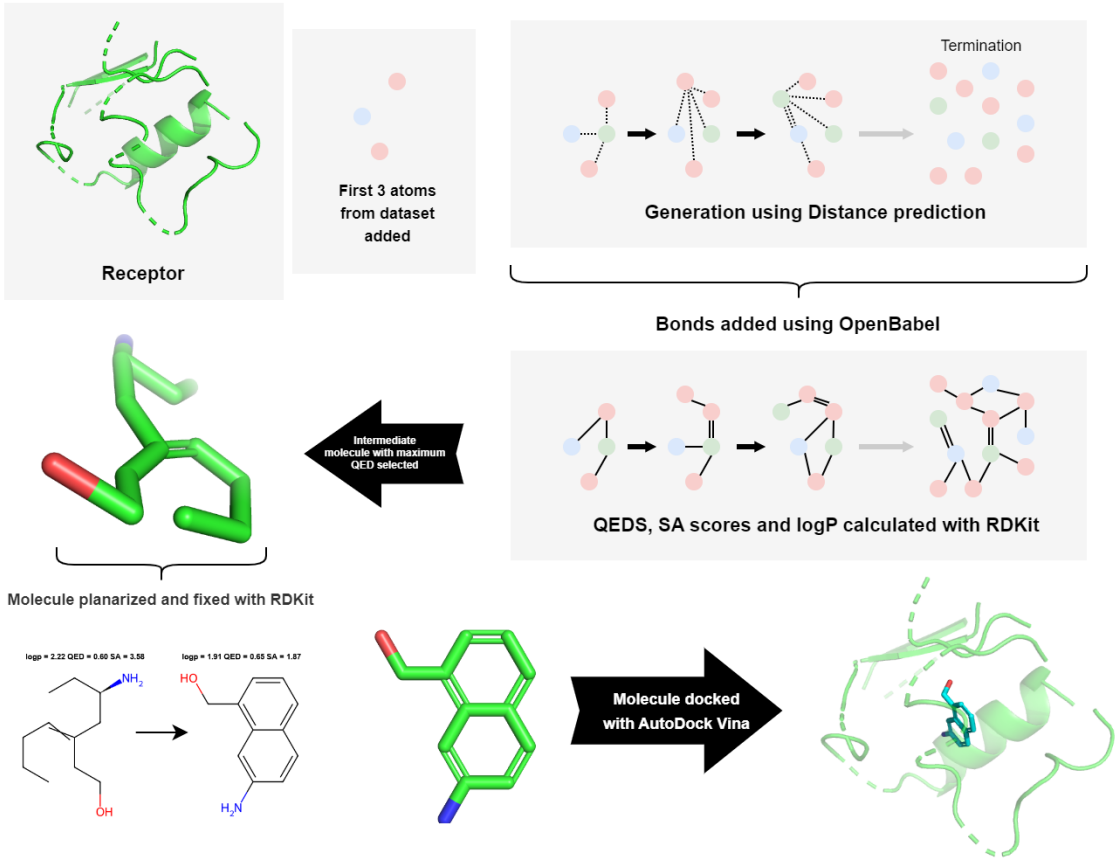


Figure 5.8: Summary of the generation process

A summary of the entire generation process is shown in Figure 5.8. The implementation of the described method can be found here.

Does Planarization Help ?

Two methods of planarization of molecules were suggested. The first method was to planarize the molecule after the generation was complete. One drawback of this method is that after planarization, the structure of the molecule is completely different, and docking has to be done again. To analyze if planarization improves the molecule, the QED of the molecule is checked, and the planarized molecule is considered if the QED is better. Figure 5.9 shows the distribution of the QED before planarization and after checking for improvement in QED. It was found that planarizing improved the QED for 35% of the molecules.

The second method suggested was to add atoms in the plane during generation. Though the atoms are in the same plane, the model tends to add atoms in a collinear fashion at times. So, RDKit was used to alter them to form rings accordingly and add

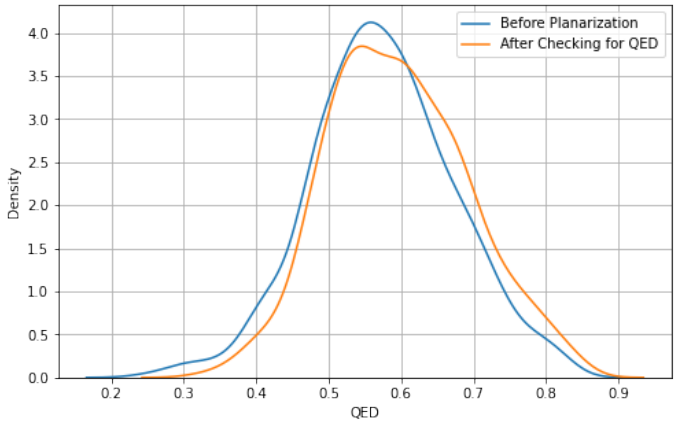


Figure 5.9: Effect of planarization after generation on QED

aromaticity. As done previously, one among the generated molecule and the altered molecule was chosen based on their QED. Figure 5.10 shows the QED distribution before alteration and after checking for improvement in QED with the alteration. It was found that using RDKit to alter the molecule improved the QED for 30% of the molecules.

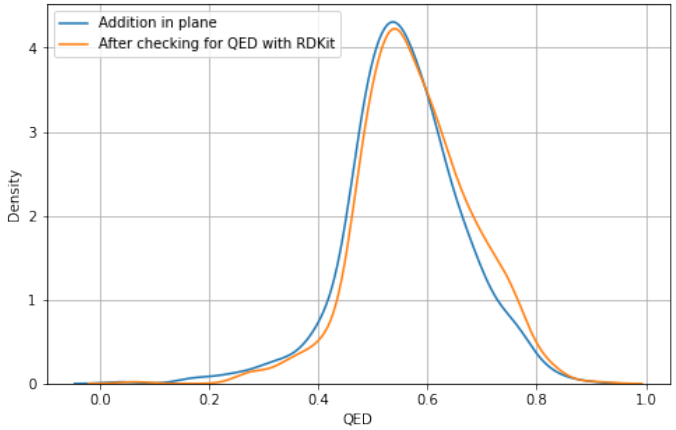


Figure 5.10: Effect of using RDKit on molecules generated in a plane

In both methods, the molecule with the maximum QED after the corresponding changes, planarization in the first method and modification with RDKit in the second method, is considered for further analysis.

Comparison of Planarization Methods

In this section, the two methods of planarization are compared. Figure 5.11 shows the distributions of logP, QED, and SA scores for both methods respectively. It can be seen that the logP distribution is better for the addition of atoms in a plane as the values are concentrated between 1 and 3. The QED distribution is slightly worse for the addition

of atoms in a plane, but again, the SA score distribution is better.

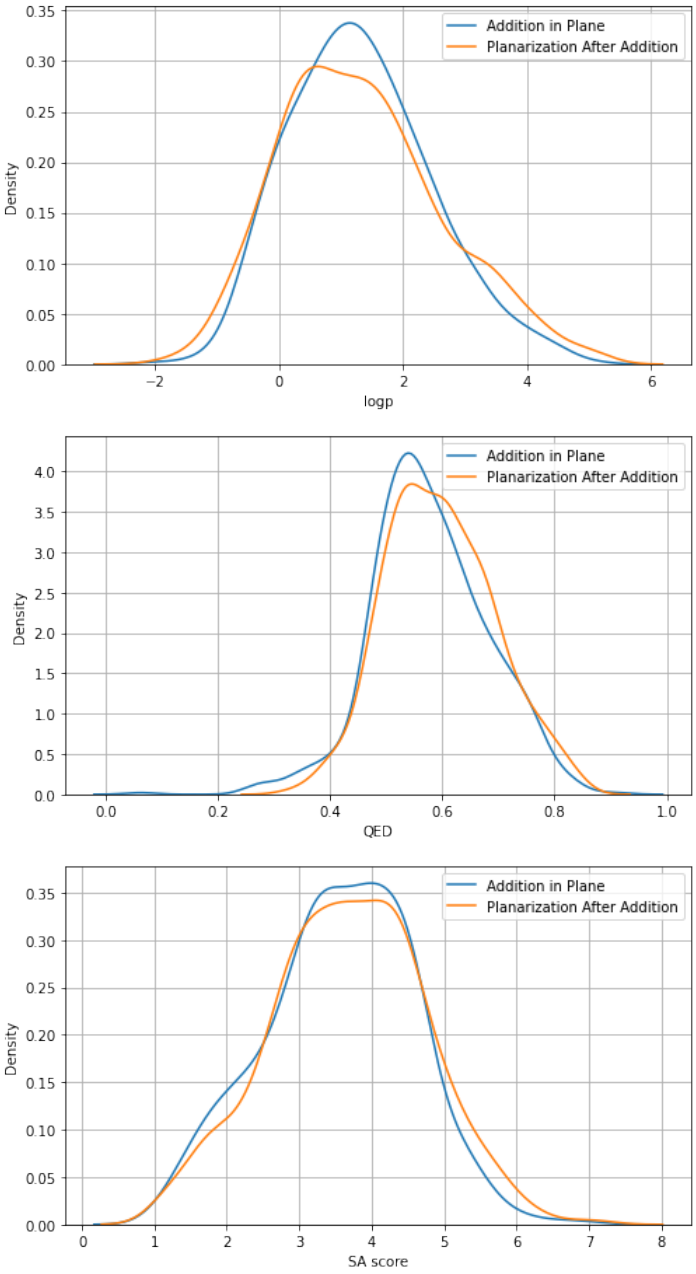


Figure 5.11: Comparison of planarization methods

A few molecules generated using both methods are also shown in Figure 5.12. One advantage of the addition of atoms in the same plane is that the structure is not changed much when modified with RDKit, but in the case of planarization after addition, the structure is changed drastically, and hence docking should be done again. Henceforth, the addition of atoms in a plane will be used for the following reasons: better SA score and logP distributions, almost the same QED distribution, and the less significant change in structure when modified.

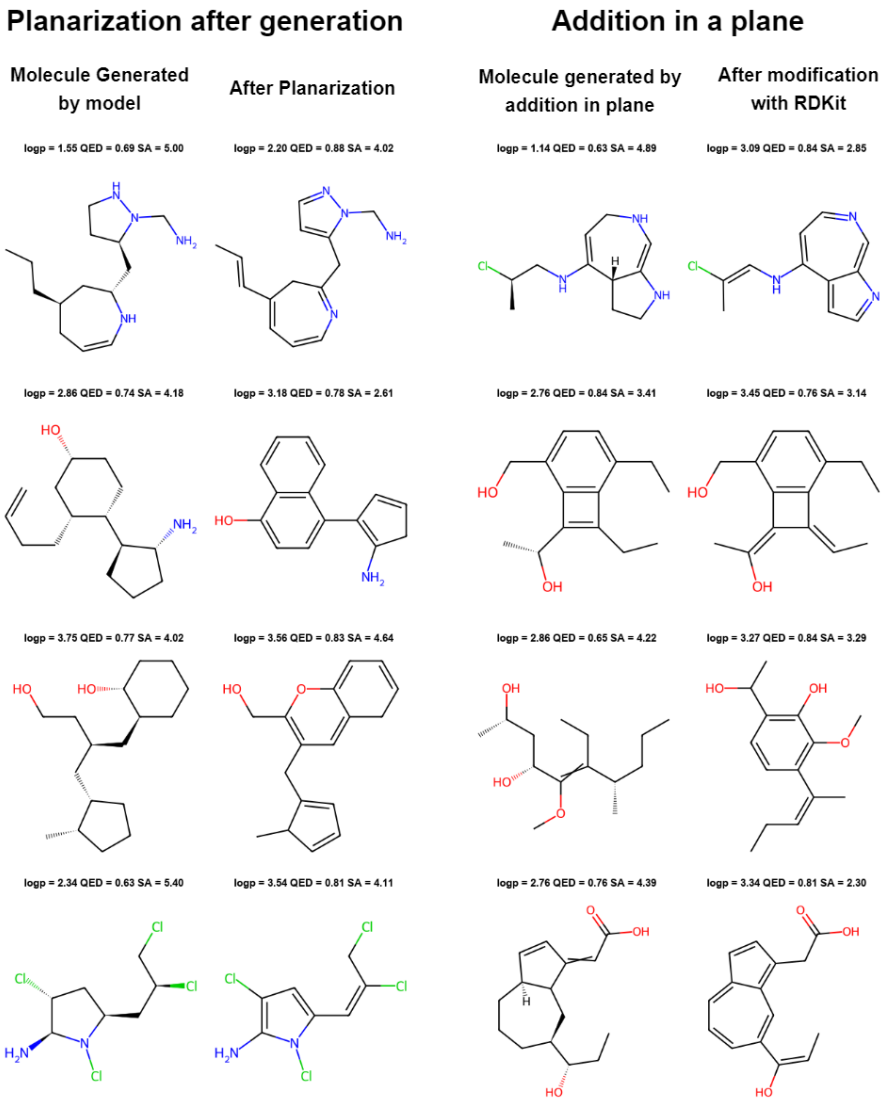


Figure 5.12: Example molecules obtained by the planarization methods

Comparison with Data

Figure 5.13 shows the distribution of the properties of molecules generated by the model in comparison to the molecules from the dataset. The SA score distribution of the model can be seen to closely follow the distribution of molecules in the dataset, though the number of molecules with SA scores less than 3 is a little less. The QED values of the molecules lie in the range of 0.2 - 0.8, while the generated molecules have QED values concentrated in the range of 0.4 - 0.7. The logP values of the generated molecules are in the range -1 - 4, which is also acceptable since the dataset has molecules with logP values ranging from -10 to 10.

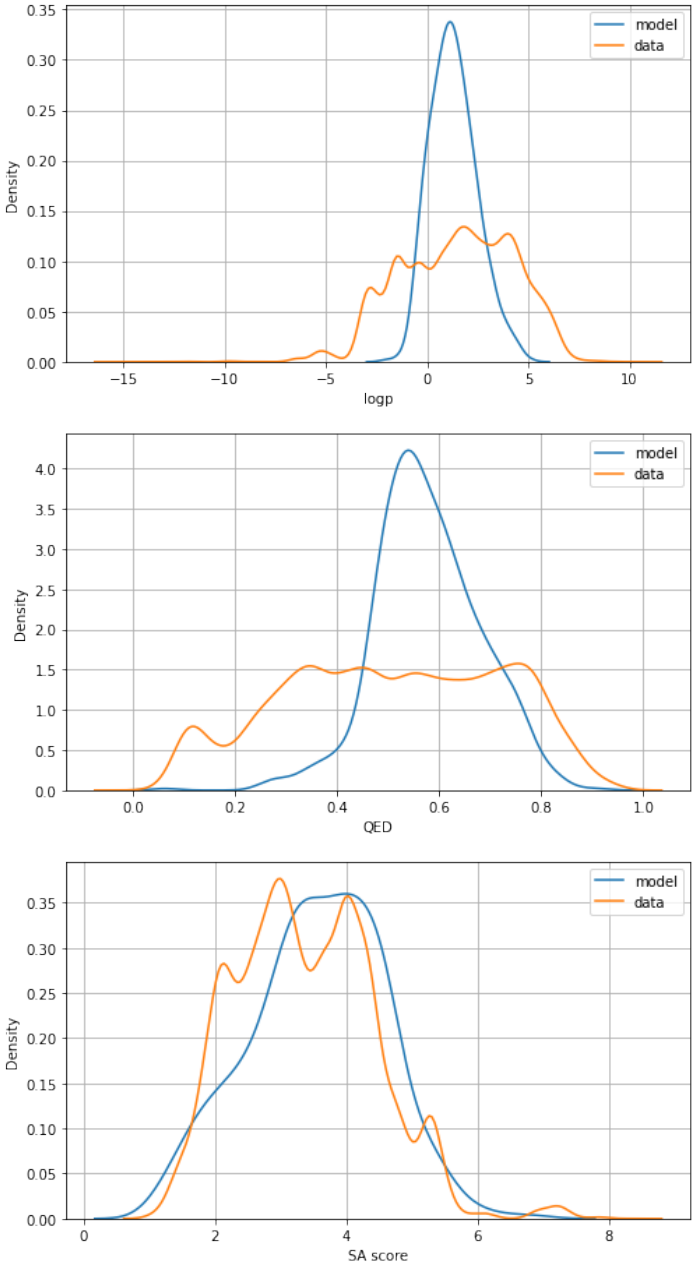


Figure 5.13: Comparison of metrics of molecules generated by model and dataset

Similarity to Data

The objective of the model is to generate new molecules which would bind to a receptor site. To see how similar the generated molecules are similar to the ones in the dataset, we use the Tanimoto similarity [18] metric to find the molecule which is most similar. Ideally, generated molecules should be completely new, but it would be convincing if some kind of substructure from the data is being replicated in the generated molecules. Figure 5.14 shows the distribution of the highest Tanimoto similarity (calculated with molecules in the dataset).

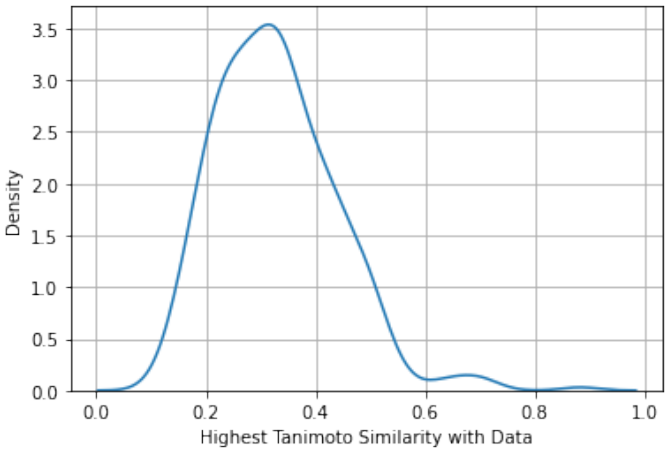


Figure 5.14: Distribution of the Tanimoto similarity with the molecules in dataset

We can see that the Tanimoto similarities lie in the range of 0.2 - 0.5, which implies that the generated molecules are not very similar to the ones in the dataset. The molecules with considerably high Tanimoto similarity (> 0.5) are shown in Figure 5.15 along with the molecule in the dataset which is most similar to it. It can be observed that a significant structure of the molecule in the dataset is getting replicated in the generated molecule, and hence it can be concluded the model does indeed learn and impose the pattern in the molecules.

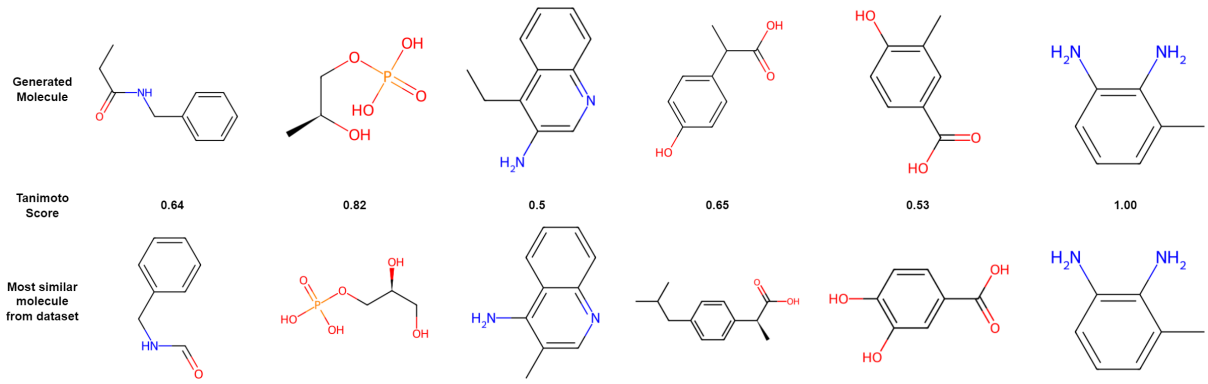


Figure 5.15: Examples of generated molecules along with the most similar (using Tanimoto score) molecule from the dataset

Outputs

Figure 5.17 shows examples of molecules generated without using the planar constraint and without any processing. Figure 5.18 shows examples of molecules generated by the final model chosen, i.e., distance prediction using the planar constraint and using RDKit for further modification.

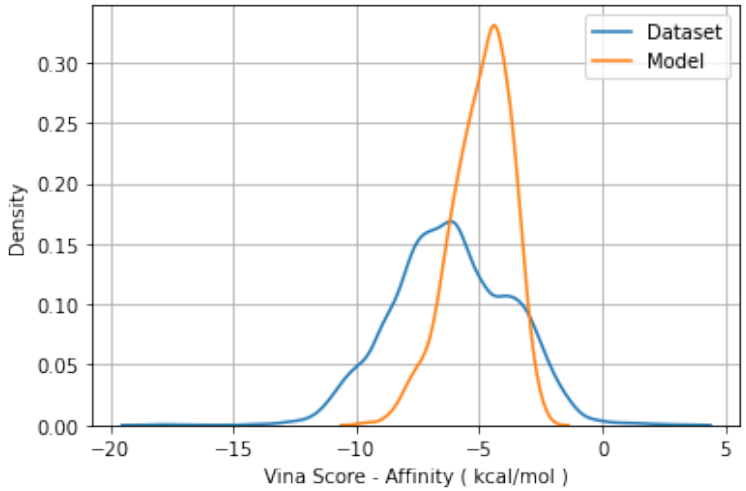


Figure 5.16: Distributions of Vina score for model and dataset

Docking Results

The generated molecules after planarization were docked using AutoDock Vina, and the distribution of the Vina scores (in kcal/mol) for the generated molecules, and the protein-ligand pairs in the dataset are shown in Figure 5.16. It was found that the generated molecules with the receptors had an average Vina score of around -5 kcal/mol, while the mean score of the pairs in the dataset was -6.1 kcal/mol. It was also found that the Vina score of most of the generated molecules before docking was way higher (worse). This can be attributed to the planarity imposed. Since the bonds are allowed to be rotated during docking, the molecule is rendered non-planar, and the Vina score gets better with docking. Though the mean value of the Vina score for molecules in the model is worse than the mean value for the molecules in the dataset, molecules with a Vina score less than 8 were generated and are shown in Figure 5.19.

Comparison

The comparison of the results with the data, the CVAE model, the auto-regressive model, and Pocket2Mol (current state of the art) is shown in the following table. The QEDs of the molecules generated outperform all the models. The SA scores outperform the CVAE model and are at par with the AR model, but not better. The logP values outperform the CVAE and the AR models and are at par with Pocket2Mol. The Vina scores are not able to outperform any of the models but are close to the CVAE model.

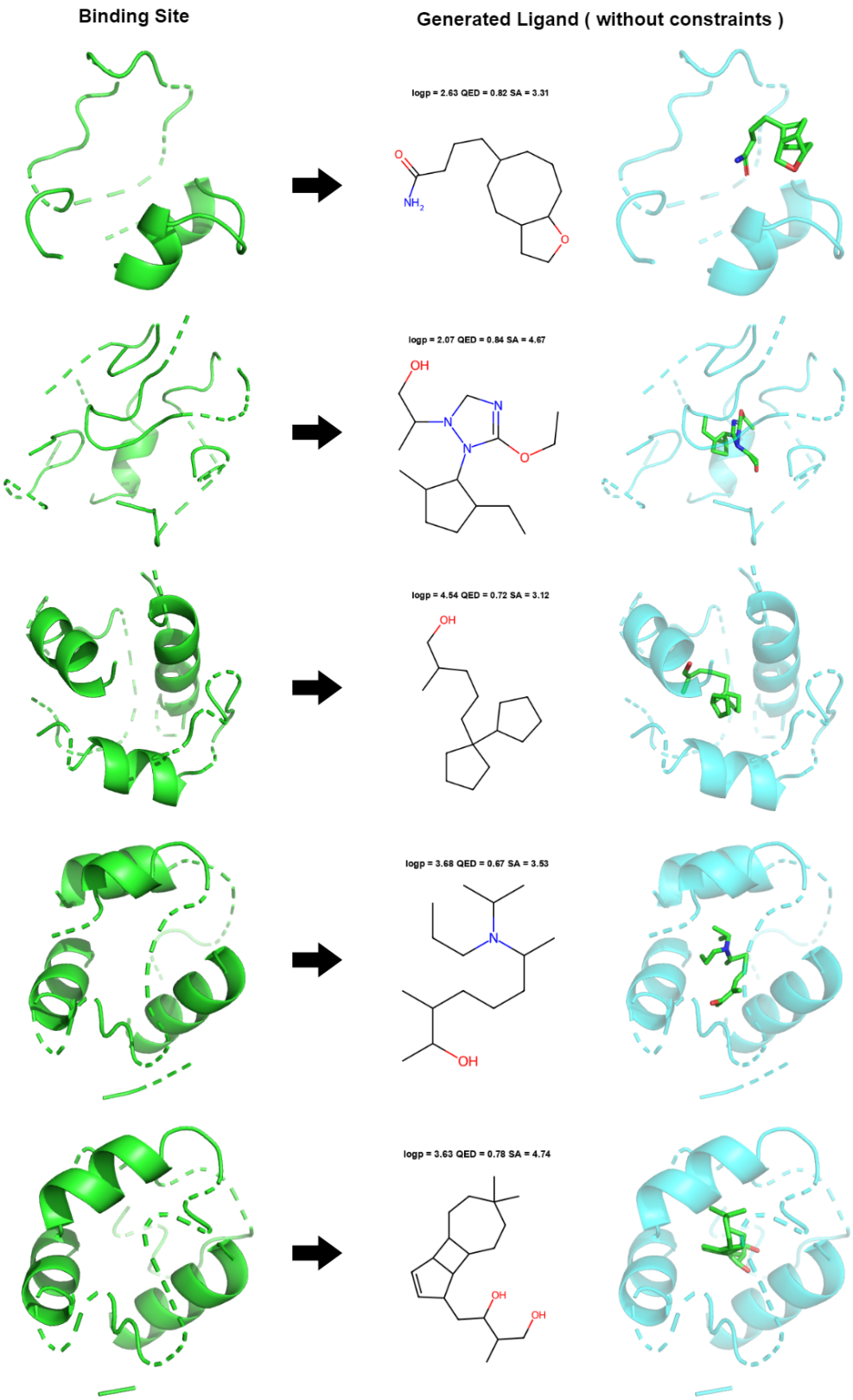


Figure 5.17: Example outputs of the model without any constraints

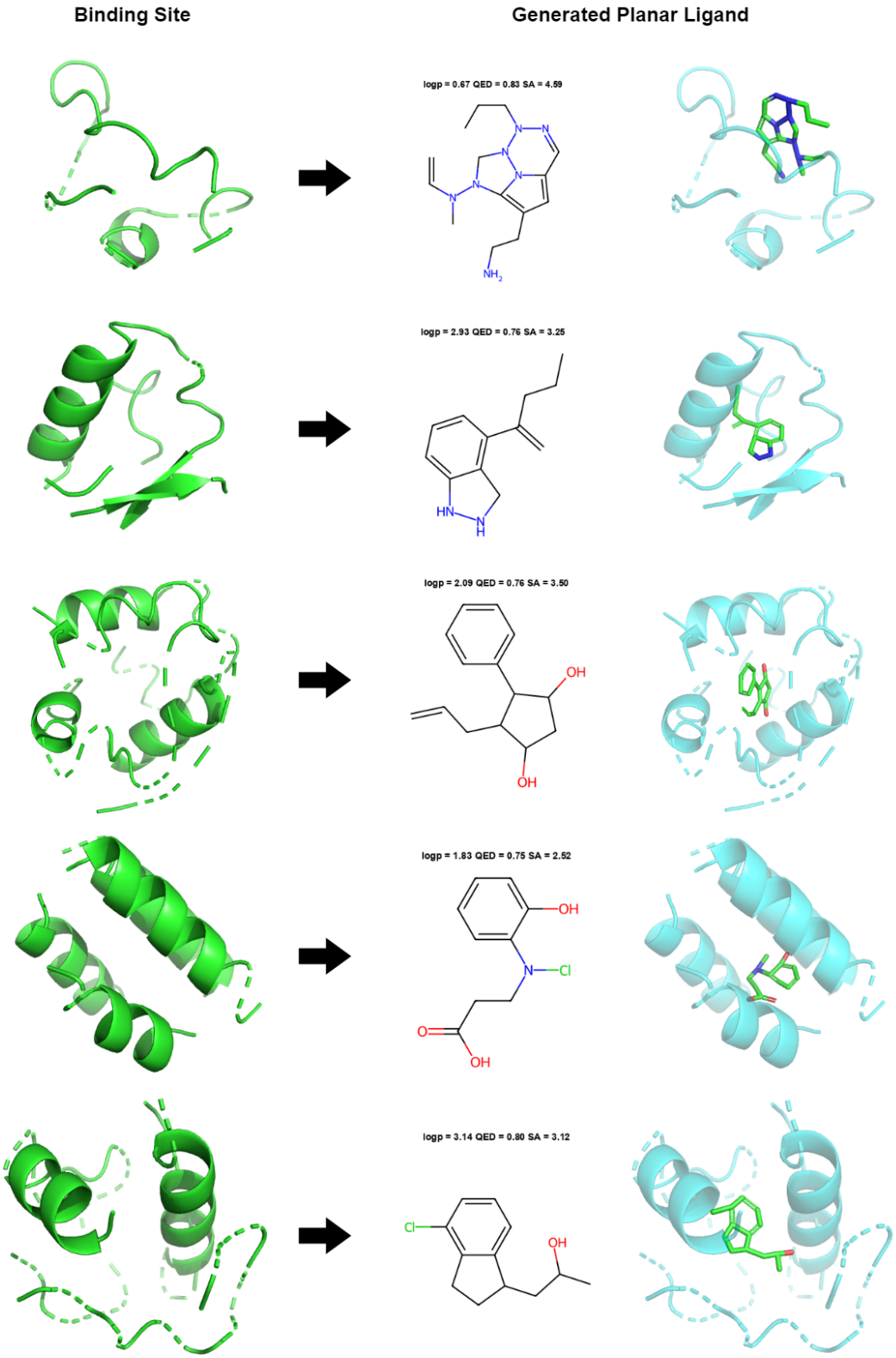


Figure 5.18: Example outputs of the planar model

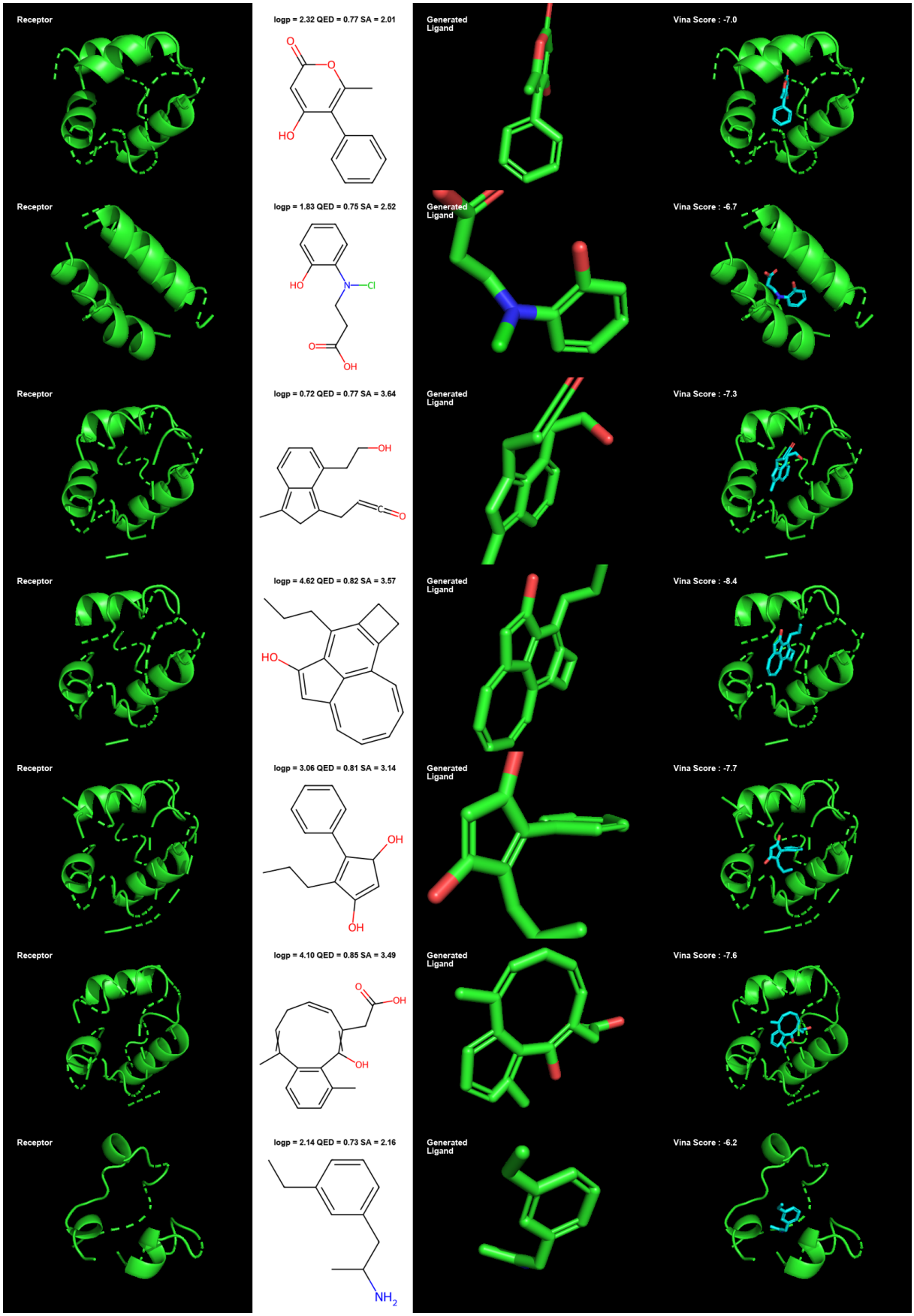


Figure 5.19: Example results from AutoDock Vina

Metric	Data	CVAE	AR	Pocket2Mol	Distance Model
QED	0.484 ± 0.21	0.369 ± 0.22	0.502 ± 0.17	0.563 ± 0.16	0.591 ± 0.1
SA	0.732 ± 0.14	0.590 ± 0.15	0.675 ± 0.14	0.765 ± 0.13	0.645 ± 0.10
logP	0.947 ± 2.65	-0.14 ± 2.73	0.257 ± 2.01	1.586 ± 1.82	1.56 ± 1.29
Vina	-7.158 ± 2.10	-6.144 ± 1.57	-6.215 ± 1.54	-7.288 ± 2.53	-5.547 ± 1.15

Table 5.1: Comparison of results

In conclusion, the molecules generated when considered as a separate entity have good chemical properties, but the interaction with the receptor needs to be improved.

CHAPTER 6

Conclusion

Structure-based drug design was treated as a generative process in which ligand atoms were added sequentially, and then bonds were added later. It was found that reinforcement learning did not give satisfactory results because of the large action space, which was a result of the definition of the environment. Different definitions of environments and rewards can be explored.

Supervised learning was then used to predict the location and type of atom at each step. Two different methods (density-based and distance-based) were explored for choosing the location, and it was found that the distance-based method gave better results. To impose planarity, two methods (planarization after generation and addition in plane) were explored, and it was found that addition in the plane during generation gave better results. It was found that the distributions of various metrics of the molecules generated were close to the distributions of the molecules in the dataset, but the similarity was less. This shows that the model indeed learns to generate new molecules given a binding site, with decent molecular properties.

Along with the individual properties of the molecules, properties related to the protein-ligand interactions were also analyzed. These properties are important to determine whether the generated ligand can actually bind to the receptor. The Vina scores of the generated ligands with the receptor, after docking, though found to be decent, were not very satisfactory when compared to the protein-ligand pairs in the dataset. In around 20% of the molecules generated, the Vina score didn't change much with docking. This suggests that the generated molecule was almost correctly placed by the model; in other words, the pose of the ligand was correctly predicted. In conclusion, the model, though poorly, does learn to predict the pose of the ligand.

Future Scope

In the current framework, only the atoms as such, and not the bonds, are considered during the generation phase. The bonds of the receptor site are also not being considered. The bonds of the ligand get added in the last phase by OpenBabel, which adds bonds that best fit the positions of atoms without any information about the receptor. There is scope for the development of models which would take into consideration, the bonds of the receptor and ligand during generation and generate bonds for the ligand by itself, without external help from OpenBabel.

One other important aspect of the current framework is that the first three atoms in the ligand are added from the data, and only then the generation phase begins. So, the model is incapable of generating ligands for receptors that have no known binding ligands. Different methods like PocketMiner [19] can be explored to find out the binding site in the receptor and add the first 3 atoms accordingly.

The framework being used now generates only planar molecules. Planarity is imposed as a hard constraint during the generation process. Planarity was imposed to bring out aromaticity, and for aromaticity, the entire molecule doesn't have to be planar. Therefore, a more implicit way to impose planarity to just parts of molecules instead of making the whole molecule planar can also be explored.

In the framework being used now, the distances from all the ligand atoms added so far are predicted, and then all the constraints are taken into account. When the number of atoms increases, the number of constraints also increases, and there arises ambiguity as to which constraints should be given more importance. This is one possible reason why the model is unable to generate larger molecules (with size > 30). Weighting the constraints or choosing the atoms to define the next distance can be explored in the future.

REFERENCES

[1] J. P. Hughes, S. S. Rees, S. B. Kalindjian, K. L. Philpott, Principles of early drug discovery, British Journal of Pharmacology 162 (6) (2011) 1239.

[2] A. C. Anderson, The process of structure-based drug design, Chemistry and Biology 10 (9) (2003) 787–797.

[3] P. G. Francoeur, T. Masuda, J. Sunseri, A. Jia, R. B. Iovanisci, I. Snyder, D. R. Koes, Three-dimensional convolutional neural networks and a crossdocked data set for structure-based drug design, Journal of Chemical Information and Modeling 60 (9) (2020) 4200–4215.

[4] G. Landrum, Rdkit: Open-source cheminformatics software (2016).

[5] N. M. O'Boyle, M. Banck, C. A. James, C. Morley, T. Vandermeersch, G. R. Hutchison, Open Babel: An Open chemical toolbox, Journal of Cheminformatics 3 (10) (2011) 1–14.

[6] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, Pytorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024–8035.

[7] M. Fey, J. E. Lenssen, Fast graph representation learning with pytorch geometric (2019).

[8] O. Trott, A. J. Olson, AutoDock Vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization and multithreading, Journal of computational chemistry 31 (2) (2010) 455.

[9] J. You, B. Liu, R. Ying, V. Pande, J. Leskovec, Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation, Advances in Neural Information Processing Systems 2018-December (2018) 6410–6421.

[10] S. Luo, J. Guan, J. Ma, J. Peng, A 3D Generative Model for Structure-Based Drug Design (3 2022).

[11] K. Yang, K. Swanson, W. Jin, C. Coley, P. Eiden, H. Gao, A. Guzman-Perez, T. Hopper, B. Kelley, M. Mathea, A. Palmer, V. Settels, T. Jaakkola, K. Jensen, R. Barzilay, Analyzing Learned Molecular Representations for Property Prediction, Journal of Chemical Information and Modeling 59 (8) (2019) 3370–3388.

[12] M. M. Stepniewska-Dziubinska, P. Zielenkiewicz, P. Siedlecki, Development and evaluation of a deep learning model for protein–ligand binding affinity prediction, Bioinformatics 34 (21) (2018) 3666–3674.

[13] M. Ragoza, T. Masuda, D. R. Koes, Learning a Continuous Representation of 3D Molecular Structures with Deep Generative Models (10 2020).

[14] J. Li, A. Fu, L. Zhang, An Overview of Scoring Functions Used for Protein–Ligand Interactions in Molecular Docking, Interdisciplinary Sciences: Computational Life Sciences 2019 11:2 11 (2) (2019) 320–328.

[15] S. A. Wildman, G. M. Crippen, Prediction of physicochemical parameters by atomic contributions, Journal of Chemical Information and Computer Sciences 39 (1999) 868–873.

[16] G. R. Bickerton, G. V. Paolini, J. Besnard, S. Muresan, A. L. Hopkins, Quantifying the chemical beauty of drugs, Nature Chemistry 2011 4:2 4 (2) (2012) 90–98.

[17] P. Ertl, A. Schuffenhauer, Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions, Journal of Cheminformatics 1 (2009) 1–11.

[18] D. Bajusz, A. Rácz, K. Héberger, Why is Tanimoto index an appropriate choice for fingerprint-based similarity calculations?, Journal of Cheminformatics 7 (1) (2015) 1–13.

[19] A. Meller, M. Ward, J. Borowsky, M. Kshirsagar, J. M. Lotthammer, F. Oviedo, J. L. Ferres, G. R. Bowman, Predicting locations of cryptic pockets from single protein structures using the PocketMiner graph neural network, Nature Communications 2023 14:1 14 (1) (2023) 1–15.