

Combining physics-based and machine-learning methods for de-novo drug design

A Thesis

submitted to

Indian Institute of Science Education and Research Pune

in partial fulfillment of the requirements for the

BS-MS Dual Degree Programme

by

Venkata Sai Sreyas Adury



Indian Institute of Science Education and Research Pune

Dr. Homi Bhabha Road,

Pashan, Pune 411008, INDIA.

May, 2023

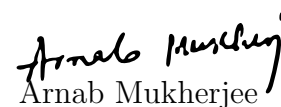
Supervisor: Arnab Mukherjee

© Venkata Sai Sreyas Adury 2023

All rights reserved

Certificate

This is to certify that this dissertation entitled Combining physics-based and machine-learning methods for de-novo drug design towards the partial fulfilment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research, Pune represents study/work carried out by Venkata Sai Sreyas Adury at the Indian Institute of Science Education and Research under the supervision of Arnab Mukherjee, Professor, Department of Chemistry, during the academic year 2022-2023.


Arnab Mukherjee

Committee:


Arnab Mukherjee

Bhimalapuram Prabhakar

This thesis is dedicated to my parents, who have always given me the freedom to explore the world of science in my own way.

Declaration

I hereby declare that the matter embodied in the report entitled Combining physics-based and machine-learning methods for de-novo drug design are the results of the work carried out by me at the Department of Chemistry, Indian Institute of Science Education and Research, Pune, under the supervision of Arnab Mukherjee and the same has not been submitted elsewhere for any other degree.



Venkata Sai Sreyas Adury

Acknowledgments

The work presented in this Thesis is inspired by the ideas of many people. Firstly, I would like to sincerely thank **Prof. Arnab Mukherjee** for giving me the opportunity to work on such an interesting project. Throughout my time under his supervision, he has always encouraged me to freely and scientifically explore the topics I worked on. He was very accepting of my mistakes and it is his constant inspiration and dedication to research that have trained me in the ways of scientific research. I would also like to thank my TAC member, **Prof. Prabhakar Bhimalapuram**, for his valuable insights on my project work.

I would also like to acknowledge **Rituparno Chowdhury**, who originally developed the physics-based drug-design algorithm. Collaborating with him on this project has been a wonderful journey. Discussions with him are always fruitful and even today I come away cheerfully optimistic whenever I talk to him.

I also express my deepest gratitude to my entire lab, who provided me with the most comfortable environment to work in, and have always been there to help me in not just my project-work, but also any personal problems. In specific, I would like to thank **Amal Vijay**, who has dedicated a lot of his time to rigorously validating the algorithm. I am also thankful to all the discussions I have had with **Sarathchandran J.**, who shares many of my interests in computer algorithms. I also gratefully acknowledge **Bikirna Roy**. He and Sarath have always lent me an ear whenever I was stuck with something, and were always full of helpful suggestions that have changed the way I have looked at many problems.

I am extremely grateful to **IISER Pune** for providing me with a friendly environment and ingraining in me the spirit of interdisciplinary research. My discussions with many course instructors has always left me pleasantly thrilled and inspired to know more.

Last but not least, I am thankful to my parents - my father **Prasad Adury**, and mother - **Jyothsna Adury** for their constant support and motivation. Without their care and love, I could never have reached this point so easily. No words can capture my heartfelt gratitude towards them.

Abstract

This thesis presents a proof-of-concept for a novel de-novo drug design algorithm that uses forcefield parameters to generate molecules in 3D space directly in the active site of a target. The algorithm efficiently samples possible molecules and their bound conformations using an approach inspired by Configurational-Bias Monte Carlo (CBMC). It is wholly atomistic and strings together atoms to construct the final molecule and uses forcefield interaction parameters to find the optimal binding partner for the target. The atom types used are parameterized in CHARMM-27 and are well-established. We have previously validated the algorithm’s accuracy in predicting strong binders through rigorous free-energy calculations. Adding to this physics-based approach, we use reinforcement learning to bias the atom type selection towards making molecules synthesizable using SYBA, an established classifier for predicting whether a molecule is synthesizable. The program shows good results by generating a diverse set of synthesizable molecules for streptavidin and HSP90, which are our test systems. The algorithm can also suggest modifications to existing ligands, thus allowing it to inspire ligand affinity improvement through minor modifications.

Contents

Abstract	xi
1 Introduction	1
1.1 Structure-based drug-design (SBDD)	1
1.2 Use of computers in SBDD	2
1.3 De-novo drug design: How and where is it used?	2
1.4 The scope of this project	3
1.5 Comparison with existing de-novo drug design algorithms	4
2 Theory	5
2.1 Role of proteins and competitive inhibition	5
2.2 The concept of forcefields and atom types	7
2.3 Metropolis Monte-Carlo	8
2.4 Configurational-Bias Monte Carlo (CBMC)	8
2.5 Graph Representations for molecules	10
2.6 Machine Learning on Graphs	12
2.7 Reinforcement Learning	12
3 Methods	17
3.1 The physics-based generation algorithm	17

3.2	Our Reinforcement Learning (RL) Model	27
4	Results	33
4.1	Vacuum (Receptor-free) Results	33
4.2	Real systems	41
5	Conclusion and Future Directions	49

List of Figures

2.1	Competitive Inhibition	6
2.2	<i>A depiction of a model linear homo-octamer placed on a 2D grid.</i>	9
2.3	<i>Growing a polymer one monomer at a time.</i>	10
2.4	<i>A graph representation of a small molecule</i>	11
2.5	<i>Schematic depiction a graph convolution operation.</i>	13
2.6	<i>A sample 3-state reinforcement learning problem</i>	14
3.1	<i>Oscillations in the interaction energy of MMC accepted molecules</i>	19
3.2	<i>Atom-type definitions require certain rules to hold</i>	20
3.3	<i>Position generation for positions in 3D space</i>	21
3.4	<i>An example of recoil due to poor atom-placement</i>	23
3.5	<i>An example of DeNovo generating a molecule atom-by-atom</i>	24
3.6	<i>Our modified implementation of a graph convolution policy network</i>	29
4.1	<i>The self-similarity distribution for 300 molecules.</i>	34
4.2	<i>Generating molecules in vacuum following the ChEMBL size distribution.</i>	35
4.3	<i>Molecules after initial training.</i>	36
4.4	<i>Change in the self-similarity after training.</i>	36
4.5	<i>The reward function during training.</i>	38

4.6	<i>The self-similarity after training with regularization.</i>	39
4.7	<i>The synthesizability as a function of size.</i>	40
4.8	<i>The molecules generated after training with regularization.</i>	41
4.9	<i>The contribution of specific atom types to the final molecules.</i>	42
4.10	<i>Self-similarity for molecules generated in the streptavidin binding pocket</i> . . .	43
4.11	<i>Sample molecules generated for streptavidin without any learning</i>	44
4.12	<i>Sample molecules generated for streptavidin after reinforcement</i>	45
4.13	<i>Ligands made by our algorithm sit tightly in the binding site</i>	46
4.14	<i>Comparing the Tanimoto similarity to the original template as a function of the retention fraction.</i>	47
4.15	<i>Molecules are reconstructed by breaking the molecule and regrowing it.</i>	47

Chapter 1

Introduction

Modern medicine has revolutionized the treatment of most common ailments that have plagued humanity for centuries. As our understanding of both disease and the human body grows, we tend to get better at treating the disease.

Some of the most successful methods for such design involve a combination of rational design of libraries of compounds and brute-force validation techniques such as high-throughput screening (HTS). Such techniques involve screening a large library of small molecules against the target, looking for signs of activity, and this is an expensive process. As a result, computational methods tend to be an attractive area of research as they can cut down the expenses by having a much better hit-rate than traditional HTS methods^[1].

1.1 Structure-based drug-design (SBDD)

The working principle used to design many drug molecules that are in use today is structure-based drug design (SBDD), which is a rather recent development^[2].

In SBDD, we use information of the 3D structure of the protein target. We expect a drug molecule to be a small molecule with the capacity to *specifically* interact with this protein and cause a noticeable change in its activity at minute concentrations. Specificity is necessary to ensure that the drugs have as few side effects as possible. Low concentrations are required to ensure that the human body can tolerate the required dosage of the drug.

The protein data bank (PDB)^[3] has now grown to contain an extensive collection of 3D structures of proteins, many of which are found in the human body, allowing the use of these

structures for SBDD. Meeting all the above requirements is rather tricky, and the dream of being able to design a suitable ligand (drug molecule) for any given protein is still a complex problem to solve.

1.2 Use of computers in SBDD

The field of computational drug design is itself very vast. Standard methods in computational drug design involve screening a large library of known drugs *in-silico* - known as virtual screening. This screening has the advantage that most compounds screened are known drugs with complete dosage studies, but it limits us to finding molecules from a pre-decided list for repurposing.

Molecular docking is another commonly used method. Molecules could be designed by any method, including manual inspection of the binding pocket and the chemist's intuition. Docking is the process of searching through possible binding conformations of the molecule in the active site to find the one that has the best possible interactions. Docking is also helpful to rank potential binders in the order of their interaction strength - a prerequisite for a high success rate of virtual screening techniques that use docking as their screening protocol. For example, a study looked at 18 million drug-like compounds and analyzed them by docking them to the Dengue virus' NS3 protein^[4].

However, this only solves some problems. Docking does not capture the motion of the receptor, and more importantly, docking cannot produce new molecules as potential candidates - it can only screen existing ones. These limitations build up to one common requirement:

We need a method capable of producing potential binders for a given protein target

1.3 De-novo drug design: How and where is it used?

De-novo stands for "anew", meaning that the molecules are designed completely from scratch, with no knowledge of existing ones. This starkly contrasts with ligand-based design, where new molecules are predicted based on known molecules.

While the ligand-based design does have its advantages in that it is better guided and has a greater success rate^[1], there are cases where de-novo drug design is unavoidable.

1.3.1 Drugging the undruggable

Many proteins in the human body are in dire need of small-molecule drugs, which, if found, could revolutionize medicine for certain ailments. However, by contemporary methods, these regions have been labelled *undruggable*, which implies (using heuristic metrics) that designing a ligand to bind to the required region is unlikely to succeed owing to weak/poor interactions, and the requirement of particular interactions whose satisfaction is too restrictive to allow the design of suitable candidates^[5].

Ligand-based design is not possible in these cases and other such cases where very few (if any) non-natural binders are known. De-novo drug design requires no such information and is likely to produce new candidate molecular scaffolds beyond human conception.

1.3.2 Antibiotic Resistance

The use of antibiotics started soon after the discovery of Penicillin in the 1900s. Penicillin and its derivatives were a rapid success in the field of antibiotic treatment, but this success was soon met by resistance. For a multitude of reasons^[6] (including the frequent use of antibiotics and the exposure of wild bacteria to antibiotics through poor waste management), many strains of bacteria started developing resistance to common antibiotics.

While there have been (and still are) constant efforts towards developing newer antibiotics to which resistance has not yet been developed, many methods still rely on modifying existing antibiotics, which are quickly met with resistance. De-novo design has the potential to produce molecules that have never before been used as antibiotics, giving us the edge in this constant arms race.

1.4 The scope of this project

Having sufficient motivation for de-novo drug design, this thesis focuses on presenting a new algorithm for such de-novo drug design. The project has two parts:

1. As will be explained in the Theory section, this algorithm is designed to use atom-types (a parameterized representation of atoms for computational modelling) and an approach inspired by Configurational-Bias Monte-Carlo^[7] to generate molecules atom-

istically (atom-by-atom) in a binding pose at the (user-specified) active site of the target protein.

2. Given that the molecules we are generating are potential drug candidates, the molecules generated must satisfy some properties, such as being synthesizable and drug-like. We use pre-existing models for this project and show how a reinforcement algorithm is capable of biasing a completely random atomistic generation at each step to end up with a collection of synthesizable molecules.

1.5 Comparison with existing de-novo drug design algorithms

Many algorithms are already present that have similar goals. However, this method differs from most of them in some key ways.

Some methods^{[8][9][10]} tend to generate molecules independently of the protein (sometimes biasing for certain molecule-level features), followed by docking to screen for hits. This method relies both on the accuracy of docking and results in many generated molecules failing the screening phase, causing the algorithms to likely be very slow or inefficient. Additionally, to our knowledge, this is the only method to implement reinforcement learning on an algorithm that generates molecules in the protein’s active site.

Many methods of de-novo drug design are not atomistic in that they use fragments^{[8][11][12][13]} instead of atom-level construction of molecules. While these methods are likely to have better accuracy and hit rate, they are likely to fall into the same trap of producing molecules with scaffolds similar to known molecules^[14], defeating the purpose of de-novo design in the first place.

Putting all these points together, having a trainable, atomistic, generative model that produces 3D bound structures as a part of the generation itself could prove to be a valuable tool in the field of Computational SBDD. The program (unimaginatively titled “**DeNovo**”) presented in this thesis achieves these goals as a proof-of-concept.

Chapter 2

Theory

The program derives its theory from multiple different concepts. This section will briefly cover the necessary background theory. There will be references pointing to works that go into the theory in greater detail. How exactly the various parts of the theory described here come together to make a de-novo drug design program will be discussed in the next chapter.

2.1 Role of proteins and competitive inhibition

Proteins very carefully regulate almost every biological process in our body. Since proteins play such a key role in maintaining healthy conditions, any imbalance or disruption in their regular activity causes diseases. Even diseases caused by pathogens usually have stages in which some proteins from the pathogen interact with the human body (such as when SARS-CoV-2 infections were found to be mediated by the Spike-ACE2 interactions^[15]).

As a result, if there were some way for us to manually interfere with the malfunctioning proteins, resulting in the modification of their activity, we could potentially treat most diseases. While many forms of interactions are possible between a protein and a ligand, computationally, the problem of finding competitive inhibitors is comparatively well-defined. This is because most other interactions (such as activation or allosteric inhibition) require a certain *kind* of interaction between the protein and the ligand and relies heavily on the protein's capability to respond to this ligand binding, which makes it situational at best. However, competitive inhibition is achieved when the drug molecule competes with (rather outcompetes) the natural binding partner of the protein and forces a far lower frequency of actual

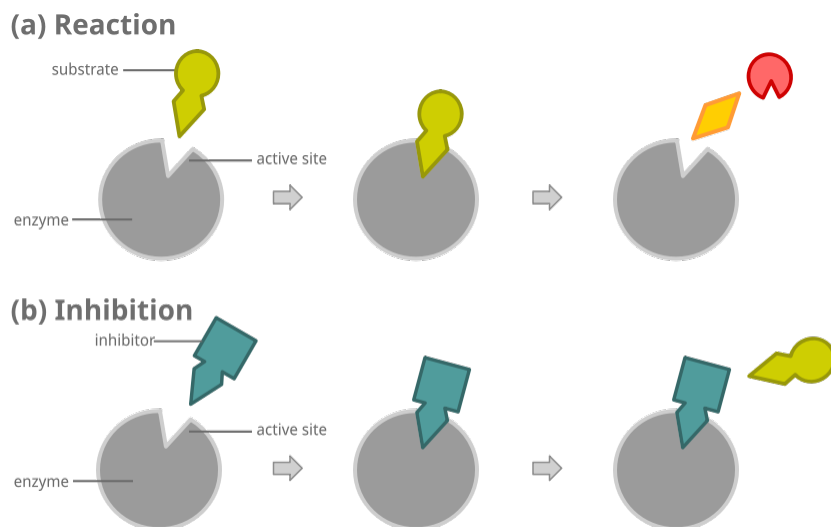


Figure 2.1: *Competitive inhibition*^[1]: The drug **forcibly** replaces the natural ligand

binding to this partner. In most cases, the actual active site of an enzyme is likely to be well-known. In the case of competitive inhibition, the sole requirement is that the drug molecule binds as strongly as possible to the active site (so long as it is specific to the target protein only). Competitive inhibition is also quite well sought after, as many diseases require us to prevent a certain protein-ligand or protein-protein interaction (for example, the Spike-ACE2 interaction mentioned above).

Calculating binding interaction is a much more tangible (although still challenging problem) to solve computationally. With the help of generic small-molecule forcefields (explained in the next section), we can produce a numerical measure of interaction between a ligand and a protein, given its bound 3D configuration. A short derivation to explain how binding strength (or binding free energy) relate to the strength of a competitive inhibitor is shown below:

Let E be the enzyme/protein and S be its natural substrate/ligand. Let D be the drug. Consider these competing binding reactions and their corresponding equilibrium constants:



This gives us (by definition of equilibrium constants):

$$K_s = \frac{[ES]}{[E][S]} \quad \text{and} \quad K_d = \frac{[ED]}{[E][D]} \quad \implies \quad \frac{K_d}{K_s} = \frac{[ED][S]}{[ES][D]} \quad (2.2)$$

For proper inhibition, we have these two conditions:

$[ED] \gg [ES]$ (we want to override the natural substrate with the drug)

$[S] \gg [D]$ (the natural ligand is abundant, while the drug will only be present in traces)

This means that we want to have $K_d \gg K_s$ (By using above conditions in eq. 2.2), which requires that we have the protein-drug interaction be as strong as possible (from the relation $-RT\ln(K_d) = \Delta G_{\text{bind}}$, where low ΔG_{bind} represents strong interactions).

With this goal in mind, we optimize the interactions between the protein and ligand during the drug generation phase.

2.2 The concept of forcefields and atom types

In recent years, performing computer simulations of chemical systems has become rather common. However, these systems are rarely simulated to complete perfection due to obvious computational limitations. Even in a numerical simulation of the system, performing full-scale quantum calculations at each step is also not feasible for most simulations with long enough timescales.

Forcefields^[16] model the chemical systems using simple parameters, reproducing bulk properties as best as possible and developing these parameters in a generic way so that they can be extended to most molecules. This modelling allows simulations to proceed by simply solving the classical laws of motion under extremely simple potentials. Most forcefields have an additive potential, i.e. the net potential energy of the system is merely the sum of the potential energy of interaction computed between all pairs of atoms, and the potential for each pair is independent of the positions of all the other particles^[16].

This property and the fact that forcefields also provide atom geometry information allow us to generate molecules in the protein active site. Atom types are atom classes for forcefields. Forcefields classify interactions into basic categories: Bonds, Angles and Dihedral angles model **bonded** interactions, while electrostatic interactions and van-der-waals' interactions are modelled by partial charges, and a distance-dependent dispersion force such as the LJ-Potential, which are classified as **non-bonded** interactions.

2.3 Metropolis Monte-Carlo

Every de-novo drug design method needs two things apart from an algorithm capable of generating chemically rational molecules - a scoring function and an optimization algorithm. Some methods use Genetic Algorithms as their optimization algorithm, while others prefer learned models. DeNovo primarily uses Metropolis Monte-Carlo criteria to effectively span the vast chemical space of potential binder molecules, allowing it to spend more time in regions of chemical space with strong binders. Metropolis Monte-Carlo is a Markov-Chain Monte-Carlo (MCMC) algorithm that allows us to efficiently sample points from a particular underlying distribution and is applied to statistical mechanics problems where it was found to be a computationally tangible method for exploring large phase spaces^[17].

2.4 Configurational-Bias Monte Carlo (CBMC)

CBMC started by trying to sample the distribution of lengths of “growing molecular chains” (i.e. polymers) to compute the length, size, and other parameters for growing polymers^[7]. The algorithm was later extended by Siepmann and Frenkel in their work on a sampling scheme for flexible chains^[18]. This is where the method used in this project takes its inspiration.

Consider a polymer that is composed of N monomers placed in sequence. Suppose we were to sample all the configurations of this polymer in 2D space (2D is for the simplicity of visualization. Extension to 3D is relatively straightforward). In computational models, this is commonly achieved by placing the atoms on a grid, where each unit distance on the grid corresponds to the distance between two monomers (see figure below): Most randomly picked coordinates would end up causing the polymer to clash with itself (and this problem worsens as the polymer’s length increases). In order to avoid this problem, a growth algorithm was devised by which the generated configurations have a high chance of being stable. This method ensures that the configurations of the polymer are sampled as they would occur in nature - according to the Boltzmann distribution of their energy.

Let there be a symmetric global energy function $E(i, j) \equiv E(j, i)$ that computes the interaction energy between two non-bonded monomers in a polymer based on their positions in

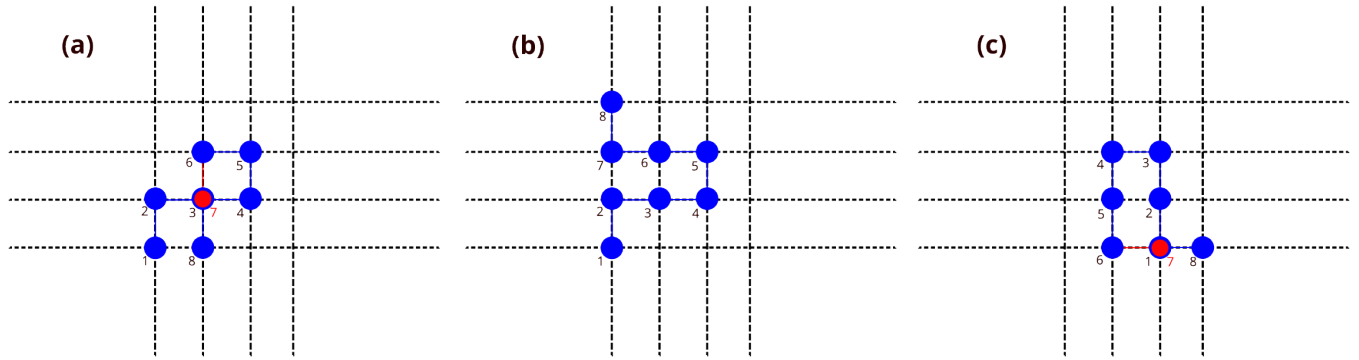


Figure 2.2: A depiction of a model linear homo-octamer placed on a 2D grid. *a* and *c* show examples of clashing (disallowed) configurations while *b* is an allowed configuration.

space. We further assume:

$$E_{\text{polymer}} = \sum_i \sum_{j>i} E(i, j) \text{ where } E(i, j) = 0 \text{ if } i \text{ and } j \text{ are connected monomers}$$

The polymer is repeatedly “grown” in a computer by adding one monomer at a time. The next monomer can be added in any of the 4 (6 in 3D) directions from the endpoint (See figure below). The key to this algorithm is the *weights* given to each of the n allowed positions for each added monomer:

$$w_j = \exp(-\beta \Delta E_j) \text{ and the normalization constant } A_i = \frac{1}{n} \sum_{j=1}^n w_j \quad (2.3)$$

The number of allowed positions may differ at each step (such as if there is a blockage - see part (c) of the figure below). This variation is why n is part of the normalization, and this can be dropped if n is known and fixed. The same chain is grown multiple times, and for a sequence of N monomers placed, we define the Rosenbluth weight to be:

$$W := \prod_{i=1}^N A_i \text{ (} A_i \text{ is as calculated in eqn. 2.3)} \quad (2.4)$$

So when a *new* conformation is grown using this algorithm, it is picked with a probability of acceptance given by:

$$P_{\text{acc}} = \min \left(1, \frac{W_{\text{new}}}{W_{\text{old}}} \right) \quad (2.5)$$

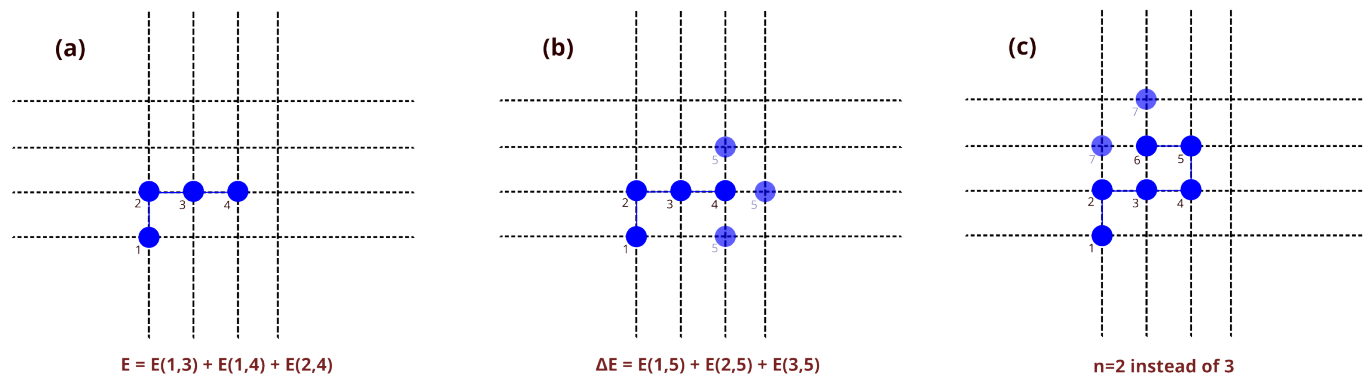


Figure 2.3: *Growing a polymer one monomer at a time.*

(a) shows how the energy function is implemented. (b) shows how multiple possible positions are considered and how ΔE is computed, which is needed for computing the weights (w_i s). (c) shows how the number of trial positions may vary.

This reweighting is necessary to reproduce the Boltzmann distribution of conformations in the final ensemble of accepted configurations, which is explained in detail in the original paper^[18] and is not particularly relevant here.

Before we conclude the theory of CBMC, it is important to recall that our drug design algorithm requires the space of allowed positions to be continuous and not gridded. So an extension of this algorithm to continuous space^[19] is used. In this extension, a fixed number of trial positions are generated subject to any known constraints, directly in real space. We then compute ΔE for each of these positions, and select the i^{th} configuration with probability proportional to the boltzmann factor of the energy change for each:

$$P(X_i) \propto \exp(-\beta \Delta E_i)$$

In the limit of sampling a large enough number of points in the continuous space, this sampling produces the correct ensemble of configurations, but in continuous space. An explicit use of this continuous sampling is explained along with the algorithm in section 3.1.1.

2.5 Graph Representations for molecules

Throughout all the discussions, we have referred to molecules as atoms stringed together. This depiction is a handy way to represent molecules to computers. Two common represen-

tations^[20] are the *graph representation*, and the *SMILES*^[21] representation (which we will not discuss here). A graph is constructed with the atoms as the nodes (vertices) of the graph and connections representing the bonds.

To feed a graph into computer programs, we commonly use two matrices: A node matrix, which is essentially a list of vectors - one for each node. Each vector contains all the necessary information to represent its corresponding node. In this figure is an example molecule in its

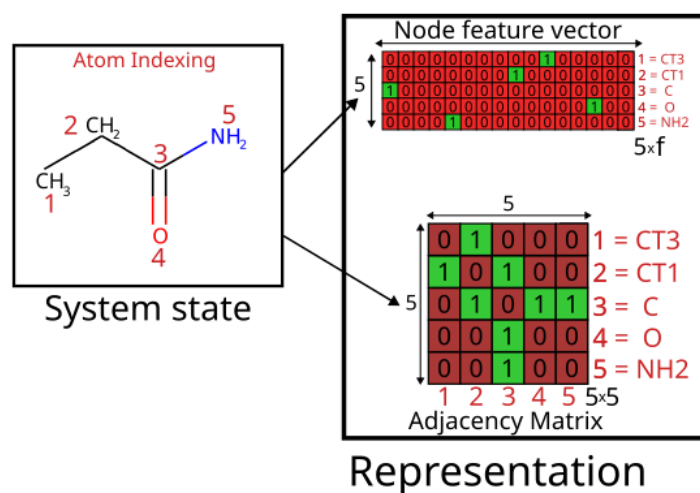


Figure 2.4: A graph representation of a small molecule

The feature vectors used here are one-hot encodings, which are described later. Hydrogen atoms are omitted from the matrices for clarity (treat CH_3 as one unit). The feature vector for each atom is of size f .

graph form. There are five atoms, and the first “matrix” is a set of node feature vectors. These “feature vectors” have predetermined size (f as denoted in figure 2.4).

The adjacency matrix is a description of connections. It is an $n \times n$ matrix (n is the no. of atoms), where all entries are 0, except at index (i, j) if atoms i and j are connected by a chemical bond, in which case it is 1. Notice how this definition also makes the adjacency matrix symmetric.

2.5.1 About feature vectors

A feature vector is the representation of a complex object using a vector of numbers. The vector is usually of fixed size, and each position in the vector represents one *feature* of the object (hence the name). For example, one can represent an atom type uniquely with five entries: Atomic number (to distinguish the element), Atomic mass (to distinguish isotopes),

number of connected atoms (to judge hybridization), partial charge (a measure of how electronegative the neighbours are) and a boolean value to tell if it is part of a cycle or not. Usually, feature vectors are much more elaborate with how features are presented to facilitate machine-learning applications.

2.6 Machine Learning on Graphs

Machine Learning refers to the broad scope of a computer learning (picking up useful patterns) from data presented to it. We have already seen how to represent molecules to computers as graphs in section 2.5. Using these graphs as inputs, models can be trained to use the structure provided by the graphs to make meaningful predictions.

As a starting point, one can imagine a machine-learning model to be a function that takes in a set of numbers (a vector) and produces another sequence of numbers as output. The output can then be interpreted based on how the model was trained.

2.6.1 Graph Convolution

Graph convolution is the process of incorporating the information from the neighbouring nodes into each node. Essentially, this provides knowledge of the “environment” for each node. It transforms the information at every node first, usually through a linear filter (essentially multiplying the input vector by a learnable matrix). After the transformation, figure 2.5 shows how the merging of information is achieved. When studying chemical structures, knowing about the local neighbourhood of an atom can be important. Carbon atoms connected to oxygen atoms will behave differently from those connected to nitrogen atoms. While atom types usually capture this information, some are generic and require this information to be incorporated from the neighbour.

2.7 Reinforcement Learning

Reinforcement Learning is a subfield of machine learning where instead of learning from labelled data, the computer learns actions through a series of steps. A reward is given to the

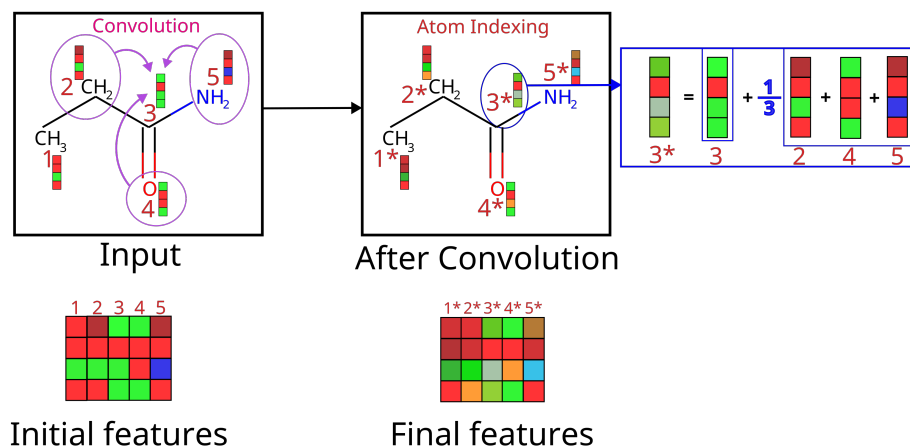


Figure 2.5: Schematic depiction a graph convolution operation.

Graph convolution combines the information from the neighbouring nodes with the existing information to provide a richer representation of each node. Vectors numbered 1-5 denote the original feature vectors. Vectors numbered 1*-5* denote the (final) convolved feature vectors.

computer after each step, and the goal of the learning algorithm is to learn to pick actions that maximize the net reward.

A simple example is that of a 3-state system. Imagine three tiles, as shown in Figure 2.6, where the computer always starts at the same point and has two choices: It can choose to move right (which will cause it to fall into a pit) or down (which will take it to its goal). These are arbitrarily defined positions, but the rewards can also be tuned for real scenarios. Now the agent is forced to pick a direction (only right or down are allowed). If it picks the correct side, it is rewarded. If it picks the wrong side, it is given a negative reward (punishment). The goal is to get the largest possible reward.

The machine learns by repeatedly playing the same game. Say it starts by randomly picking either side with equal probability. Every time it gets a positive reward, it is motivated to make the same choice next time, biasing the odds that it will again pick the same direction in the next run of the game. Similarly, it is discouraged from picking the wrong side by a negative reward.

The math of this selection is implemented as the REINFORCE algorithm^[22] (there are other reinforcement learning algorithms, but this is by far the simplest and is what is used for this project). The description of REINFORCE here is quite brief.

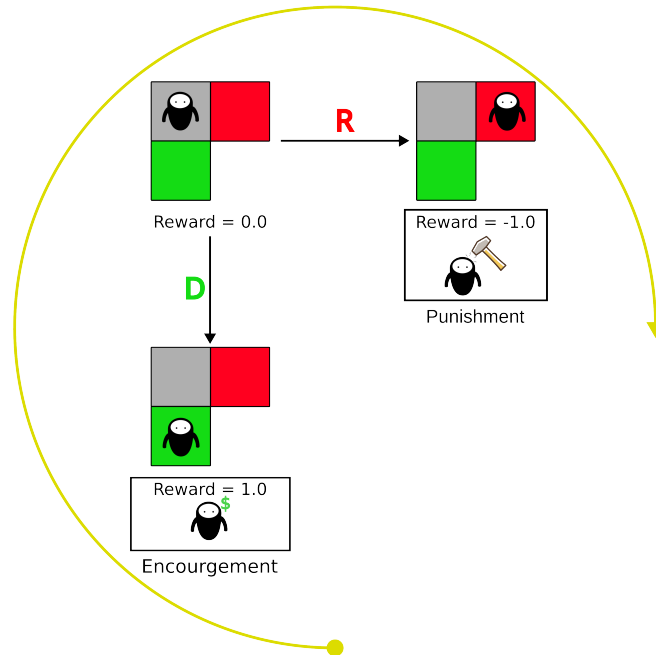


Figure 2.6: *A sample 3-state reinforcement learning problem*
The average of the feature vectors of all the neighbours is taken to be the “information” from the neighbourhood. This is added into the original feature vector.

2.7.1 The REINFORCE algorithm

The REINFORCE method with policy gradients optimizes our policy network. In this project, we use a neural network as our policy function. A policy function is essentially the learnable function that determines the machine’s next action, given its current state. A policy network is this choice function described as a neural network. In the previous example, this would be the function that gives the probability of picking right (or down) from the starting point. The inputs to this function would be all the information available to the system at the initial position (which could be the colour of the tiles on either side - red and green).

If the policy function is differentiable (as for neural networks), the reinforce algorithm takes a gradient descent form, hence the name *policy gradients*. The following derivation assumes some familiarity with the notation of reinforcement learning:

Let π_θ be the policy function. An episode is an entire sequence of states and actions until the “game” ends (in reinforcement-learning terminology, every such learning setup is called a game, where the actions the computer picks are analogous to a human’s interaction with a joystick controlling the game). Let us say we want to learn from a given episode.

The first hurdle we hit is that of credit assignment. A decision taken in the first step might

affect the reward five steps later. So we cannot assume that changing the decision just one step before a particularly bad reward solves anything. This problem, however, usually sorts itself out through multiple runs, as the decision that leads to poor rewards consistently will eventually be correlated to those poor rewards after many runs of the game. However, this raises an interesting point. The decision at step 0 will influence *all* the rewards received. The decision at step 5, however, can only influence rewards received *after* step 5. An estimate for the reward received by taking action a_i in state s_i is essentially approximated after multiple runs by:

$$R_{s,a} \approx \frac{1}{N} \sum_{\text{run}=1}^N v_t^{\text{run}} \text{ where } v_t^{\text{run}} := \sum_{i \geq t} r_t^{\text{run}} \quad (2.6)$$

Here, r_t^{run} is the reward at the t^{th} step in the runth run. Now we can estimate the expected reward for a policy π_θ from a state s as:

$$E[R_\theta^{\text{net}}(s)] = \sum_{a \in A} \pi_\theta(s, a) R_{s,a} \text{ (where } A \text{ is the set of all allowed actions)} \quad (2.7)$$

Here $\pi_\theta(s, a)$ is the probability of picking action a at state s with our current policy. To optimize, we can use gradient descent as we assumed π was differentiable with respect to its parameters:

$$\nabla_\theta E[R_\theta^{\text{net}}(s)] = \sum_{a \in A} \frac{\partial \pi_\theta(s, a)}{\partial \theta} R_{s,a} = \sum_{a \in A} \pi_\theta(s, a) \frac{\partial [\log \pi_\theta(s, a)]}{\partial \theta} R_{s,a}$$

Now in a set of episodes under policy π_θ , it can be assumed that the actions for any state s are picked according to the probability $\pi_\theta(s, a)$. So we can approximate the gradient for every state s_i by:

$$\nabla_\theta E[R_\theta^{\text{net}}(s_i)] \approx \sum_{e \in E, s=s_i} \frac{\partial [\log \pi_\theta(s_i, a)]}{\partial \theta} R_{s_i, a} \quad (2.8)$$

We can approximate $R_{s,a}$ from equation 2.6, and since we have an estimate for the gradient, we can modify the parameters by gradient ascent to maximize the reward for each state. A clean derivation showing convergence of the algorithm and a more practical setting can be found here^[23].

Chapter 3

Methods

The algorithm is built by combining multiple ideas, most of which have been discussed in the previous section. This section describes how everything comes together.

3.1 The physics-based generation algorithm

This form of a novel physics-based atomistic drug-generation algorithm was first envisioned by Dr. Arnab Mukherjee. The first implementation of the algorithm was developed by Rituparno Chowdhury by using atom type definitions from GROMACS' ^[24] force-field parameters. Venkata Sai Sreyas later improved this method by introducing rule based constraints and tuning atom type parameters thus enforcing chemical variability and consistency. The algorithm itself has been previously tested on real systems and has shown to be computationally consistent ^[31].

3.1.1 Growing a molecule atom-by-atom

DeNovo uses an *atomistic* drug generation algorithm, meaning that the molecule is generated one atom at a time. The algorithm needs to make chemically rational molecules. So along with the list of parametrized atom types, we also provide forcefield parameters. These parameters include the Lennard-Jones (LJ) parameters (σ - the atomic diameter and ϵ - the attractive strength) and partial charges (to calculate dipoles and estimate charge-based

interactions such as hydrogen bonds and dipole-dipole interactions).

We also provide the expected valency for each atom type. For example, *CT3* is a tetrahedral carbon atom bound to 3 hydrogen atoms. So it has four neighbours. *CA* is an aromatic benzene-like carbon atom. It has 3 neighbours (because it is sp^2 hybridized, there will be one double-bond). This means that the atom types determine the local geometry. How DeNovo incorporates these requirements is elaborated below.

User Inputs:

Target Size

A program can go on stringing atoms together forever. To avoid this, the user must input a target size for how big a molecule should get. The program will stop adding atoms after reaching the target size unless the addition is required for satisfying atom-type definitions (see section 3.1.1).

If the program is being applied to a target protein for the first time (or if no target size is known), it is possible to sweep over all possible sizes starting from 10 atoms to almost 45-50 atoms. One instance of the program is run for each size.

The target size counts only *heavy atoms* - i.e. non-hydrogen atoms.

The hotspot

The *hotspot* or *active-site* is the region of the protein to be targetted. If we already know where the natural substrate binds, we can use that knowledge to determine the target region. The target region is usually specified by providing the program with the key residues in the active site that are expected to interact with any new ligand. However, this selection is only suggestive of the binding site. While the program starts making molecules from that region, it is not constrained to keep the molecule there unless specifically requested by the user.

Seed Count The generation of a molecule must start with one atom. This single atom is placed randomly at the active site and is called the *seed atom*. DeNovo generates molecules by randomly spreading out seed atoms, then picking one seed at random and using it as a starting point. The user is expected to input the seed count. The seed count tells the program how many random positions to generate as starting points before generation. In general, we expect better coverage of the target region but longer convergence times with more seeds.

Oscillation Count As the molecules are selected by a Metropolis Monte-Carlo (MMC) scheme, there are expected to be oscillations (ups and downs) in the interaction energy of the new molecules as they are generated. The program is not built around sampling a

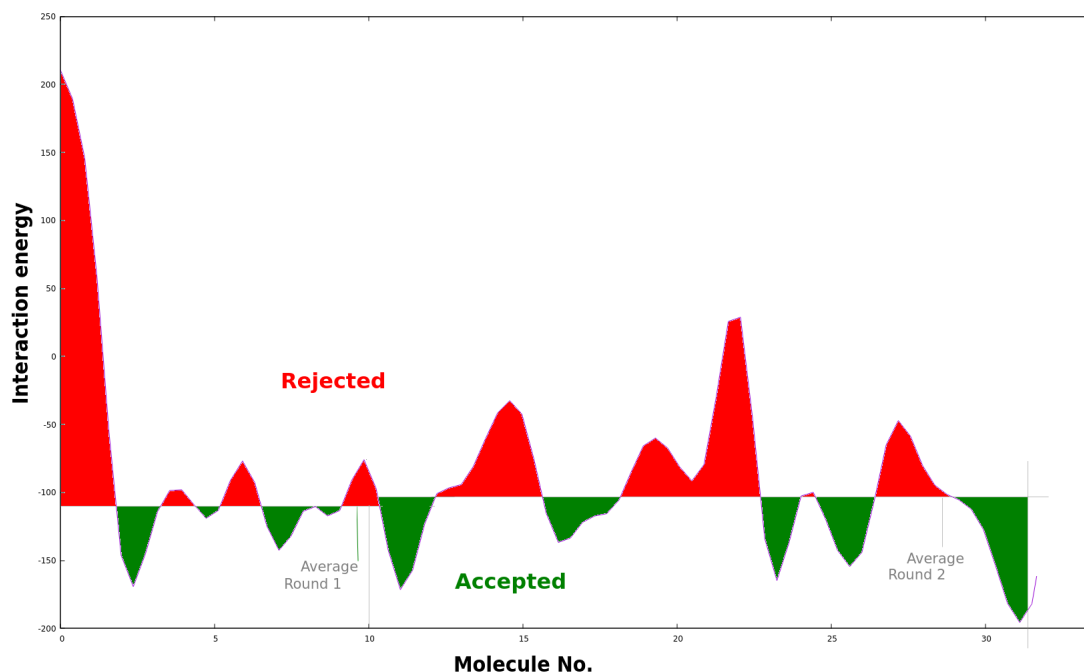


Figure 3.1: *Oscillations in the interaction energy of MMC accepted molecules*
There are two batches shown in the figure. The average line is drawn for each batch.
Molecules above the mean are rejected, and those below are accepted

Boltzmann distribution of molecules but rather on finding strong binders, so we don't take all the accepted molecules. Instead, after a few oscillations (decided by the user), we average out the energy of the last batch. Only molecules with lower (thus stronger) interaction energy to the protein (compared to the mean) are selected. This process is schematically shown in Figure 3.1

A larger oscillation count usually results in more consistent results and better molecules, but it makes the program take longer to finish.

Seed atom positioning

The program starts by generating a large number (determined by the user) of starting points from where the molecule generation will begin. At each such random position, a *seed atom* is placed, ensuring that the seed atom is not unstable at that point (i.e. does not physically clash with any of the protein's atoms). For every molecule that will be generated, one of these is the starting point.

The seed positions are generated as spherical shells around a fixed point. Usually, the fixed point is taken to be the centre of geometry of the active site (as marked by the user). A

large number of seeds are generated before each generation to ensure proper spanning of the 3D space.

Selection of atom-types

When seed atoms are generated, after choosing the positions, the program randomly determines which atom the seed will be. There is freedom for the user to choose which atoms will be picked and how frequently, but most commonly, they are carbon atoms. Sometimes there will also be oxygen and nitrogen atoms.

From here on, the molecules must be generated bottom-up. So, deciding the atom type is necessary for each step (i.e. whenever an atom is supposed to be added). By definition, atom-types have “rules” as demonstrated in figure 3.2. Beyond this, however, the atom-type

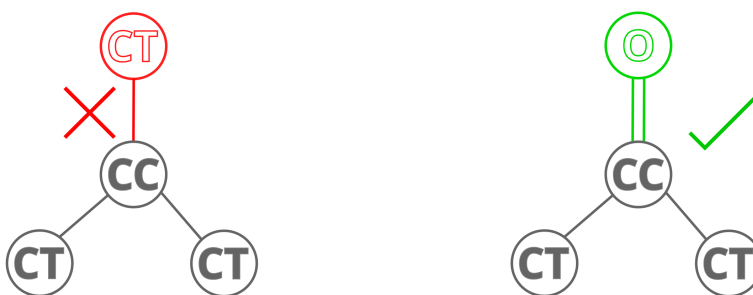


Figure 3.2: *Atom-type definitions require certain rules to hold*
In this example, the atom marked “CC” is a carbonyl carbon. It must necessarily be connected (by a double bond) to an oxygen atom.

choice is completely arbitrary. There are very generic atom types (like *CT*), which just require that the carbon atom have near-tetrahedral geometry. So all possible single-bonded neighbours are valid. In this case, one atom type is randomly picked due to the lack of information.

This selection is where reinforcement learning comes into play. The program is developed to use an external model to determine which atom type to pick when multiple options are available. This model can be trained to achieve any necessary goal that is only molecule-dependent. These could be common drug-likeness properties such as the ADMET properties^[25] or a metric of synthesizability.

As a proof-of-concept, this project shows the results obtained by training the model to generate *synthesizable* molecules. Generated molecules are predicted to be synthesizable or not based on a previously developed, publicly available classifier known as SYBA^[26].

Atom placement and inspiration from CBMC

Further atom placement proceeds similarly to the CBMC criteria. During generations, there are three possible situations we can find ourselves in. Following the CBMC criteria, a test

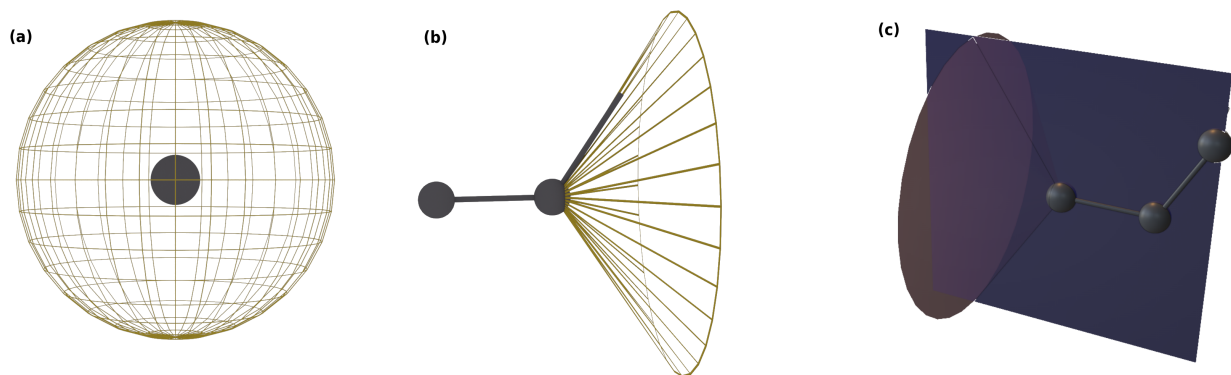


Figure 3.3: *Position generation for positions in 3D space*

(a) *When only a single atom exists in the molecule. The intersection points of the longitudinal lines and latitudinal lines represent the allowed points. (b) When two connected atoms are present. The rim of the cone drawn is the set of allowed positions. (c) The most common - after adding at-least three atoms. We determine the set of allowed dihedral angles. There is one plane corresponding to each. The points where the cone (as from case b) intersects the plane determine the allowed positions.*

atom is placed at all the positions (shown in figure 3.3). The potential energy is calculated. Because the potential from most classical forcefields is additive^[16], the ΔE value is computed by adding up the interaction energy (with the protein) of this single atom. Then, using the same biasing method (see equation 2.3), we place the atoms with probabilities that ensure that a more stable position is more likely to be picked.

The reweighting part:

DeNovo molecule construction borrows heavily from CBMC to ensure that the generated molecules are biased to be stable at the active site.

There is one key difference, however. The goal of DeNovo is not to straight-up produce a distribution of molecules from chemical space whose combined conformational and interaction energies form a Boltzmann distribution. We only want to focus on *strong* binders, not a distribution. Hence, the reweighting part is omitted in favour of directly using the scoring function (forcefield interaction energy)-based criterion for acceptance, similar to the classic Metropolis Monte-Carlo as explained in a later section.

Cyclization

Every time we add an atom, we attach it with a bond connecting it to a predetermined atom from the parent molecule. To make cycles, we need to attach an atom with *two* bonds. Because the generation is in 3D, we complete all possible cycles based on the 3D position of the atom. If the newly added atom is within bonding distance of another atom in the parent molecule, a cycle is formed unless that atom has no free valency to spare. In that case, the position is rejected as a “clash”.

Cyclization can also be part of an atom type’s rules. For instance, all aromatic carbon atoms must be part of a cycle. Similarly, atoms specifically defined to be cyclic atom types (such as carbon from cyclopentane) are also forced to be part of a cycle.

It would seem that randomly placing atoms would make it very unlikely to let them form a cycle. But this is mitigated by choice of angles and dihedrals from the forcefield data, making it far more likely, especially for small cycles (up to sizes 5-6). Larger cycles are more commonly made as fusions of multiple such small rings.

Interaction energy

The main reason for performing a search directly in 3D space rather than generating molecules first and then docking them is to optimize the molecule *in the generation phase* to have strong interactions with the protein. The interaction energy to the protein is computed from two different parameters that are part of the forcefield atom types. DeNovo uses the CHARMM-27^[27] atom types as its foundation. These atom types are slightly modified into specialized categories to allow for the diversification of atom types. For example, the *CA* atom-type stands for aromatic carbon like in benzene, but the charge on this carbon atom depends on what its substituent is. So DeNovo uses *CA* for normal aromatic carbon and *CAS* for positively polarized aromatic carbon.

The exact force constants for all the different bonds, angles, and dihedrals are present in the forcefield data but are unnecessary as we usually only consider the local minima of the structure (i.e. the equilibrium values) and not allowing large deviations. However, they are still used where available.

It is to be noted that this interaction energy is only a rough estimation of the actual value, which will be expected to change once the entire molecule is completed and parameterized as a whole.

Dead-ends and recoil

Sometimes when generating a molecule, it is possible to run into a dead-end before reaching the target size. This usually happens if there is no place to add another atom without clashing with the existing molecule or the protein. This problem also occurs in classical CBMC, and the growth is scrapped in favour of a retry. DeNovo does something similar, but instead of restarting the generation from scratch, it recoils the last “unsatisfied portion”, i.e. the set of atom-types whose forcefield rules are not satisfied yet. DeNovo adds atoms if

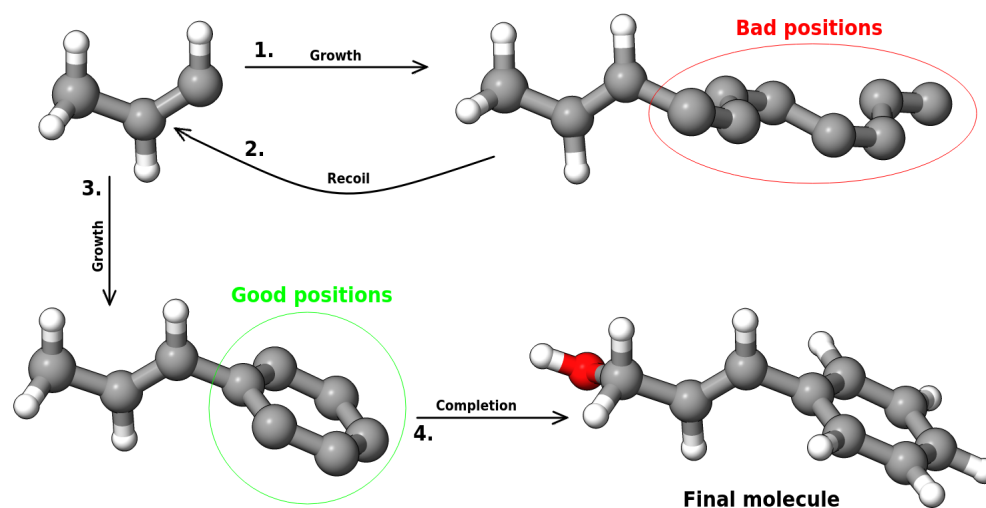


Figure 3.4: An example of recoil due to poor atom-placement

In this figure, we see recoil after many poorly placed atom positions - aromatic atoms must be placed circularly, but they were placed in an extended configuration.

either of the two criteria is met: (a) The target size has not been reached or (b) Some atoms need to be added to ensure that atom-type rules are satisfied (recall the case of carbonyl carbon and oxygen from figure 3.2).

The program first tries to ensure that all instances of unsatisfied atom types are resolved. If so, it marks that point to be the last checkpoint. Then it adds atoms only if the target size has not been reached. If the program runs into a dead-end at any point, it goes back to the last checkpoint instead of redoing the whole generation. The number of times every checkpoint can be returned to is limited, ensuring that the program can avoid being trapped by a poorly oriented checkpoint. If it fails despite the multiple tries, the entire molecule is discarded.

A special situation may occur when the molecule is out of free valencies before reaching the target size. In this case, the entire generation is scrapped. Figure 3.5 shows the stages in

which a sample molecule generated by this protocol.

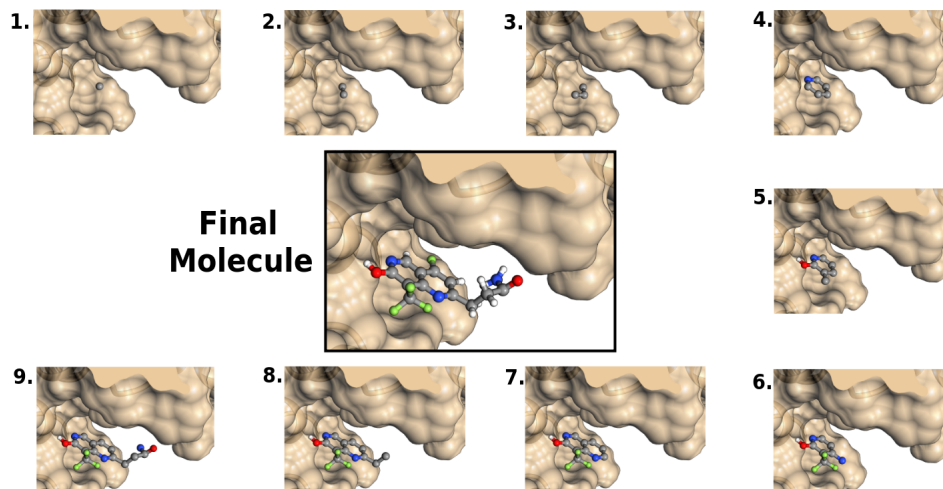


Figure 3.5: *An example of DeNovo generating a molecule atom-by-atom. The atoms are placed directly in 3D space, with their positions determined similar to the 3D implementation of foldable polymers by CBMC*

3.1.2 Extending the Forcefield

DeNovo uses atom types based on the CHARMM27^[27] forcefield. Most of the generic atom types have been preserved. However, many atom types specific to certain residues (such as nitrogen atoms belonging to nucleic acids) were removed as they are too specific for a de-novo drug generation program.

Connectivity rules

As discussed, atom types have rules that need to be followed in order for them to preserve their identity. For example, a benzene atom must be part of an aromatic cycle. These rules are phrased to the program using a simple sequence of instructions. By default, the bond-length information tells the program which atom type pairs *can* be connected. If atom types A and B have no bond-length information, $A - B$ bonds are not valid/parameterized, so they are never made.

Of all the bond information present, most bonds are completely optional. Two atoms connected to a benzene carbon atom must necessarily be aromatic themselves, but the third

atom (the benzene substituent) is completely random. Even having a substituent is completely optional.

This information is presented to the program via “rule entries”. Every rule has four components:

1. The focal atom: The central atom for which the rule is defined.
2. The target group: A subset of the allowed atom types that can bond to the focal atom. The rule will impose constraints on this group.
3. A minimum number: *At-least* these many bonds from the focal atom must end in an atom-type from the target group
4. A maximum number: *At-most* these many neighbours of the focal atom can belong to the target group

Sample rules would look like this:

CA [aromatic] 2 3

(CA, an aromatic carbon atom must connect to at-least two aromatic atoms (which are expected to cyclize. It can have all aromatic neighbours, too, like in the case of fused rings).

Charge recalculation

The CHARMM27 forcefield comes with charges already defined for its atom types. However, generalizing its atom types to different molecules without explicit reparameterization would mean these charges lose their accuracy.

The partial charge on a carbon atom connected to three fluorine atoms will obviously differ from that of a CH3 group. In order to deal with this problem, extremely small representative fragments were made with each of the newly introduced atom types wherever there was a need to recalculate charges. Then, Gaussian 09^[28] was used to perform quantum calculations to obtain ESP charges. These charges were then scaled to align with the CHARMM27 atom types. The scaling ratio was determined by the ratio of gaussian ESP charges and the CHARMM atom-type charges for known atom types (we used benzene as the benchmark to find this ratio by looking at the charge on carbon atoms).

Anyhow, we realize that the charges from static forcefield parameters may not always be very accurate due to the on-the-fly molecule generation. Our program allows users to use charges from an external source to recalculate the interaction energy with the protein once the entire molecule is made using modified charges. Any external program capable of taking SMILES^[21] as input and producing charges as output (in a specific JSON format that is predecided) can be used. This may involve using established methods like Gasteiger Charges^[29]. We are experimenting with different techniques to do the same.

3.1.3 Traversing Chemical Space

Metropolis Monte-Carlo allows us to sample points in accordance with any probability distribution - in our case - the Boltzmann distribution. A vague justification for enforcing such a distribution is this:

If we have two molecules (say A and B) with different interaction energies (say ΔE_A and ΔE_B) competing for binding to the same target location, the probability of finding molecule A is proportional to $\exp(-\Delta E_A)$. The ratio of probabilities is given by:

$$\frac{p_A}{p_B} = \exp(-\beta(\Delta E_A - \Delta E_B)) =: \exp(-\beta\Delta(\Delta E)) \quad (3.1)$$

Taking our default scoring function to be an estimate for the binding energy, we accept a newly generated molecule by comparing its binding energy estimate (according to DeNovo) to that of the previous molecule. Then:

$$P_{acc} = \min(1, \exp(-\beta(E_{new} - E_{old}))) \text{ with } \beta := \frac{1}{k_B T} \text{ (Using } T=300\text{K)} \quad (3.2)$$

Temperature as a variable parameter

In some cases^[30], optimization algorithms using Metropolis Monte-Carlo treat temperature as a tuning parameter to balance variety with optimization. Such an interpretation is possible with DeNovo as well. Increasing the temperature corresponds to quicker oscillations, resulting in less optimal but more varied molecules (useful when other key properties not captured by interaction strength are also necessary). Lowering the temperature reduces the variety and makes the algorithm slower (and may also lead to non-convergence) but will lead to better optimization.

A temperature of 300 K has still been found to be quite reasonable and successful in previous use-cases^[31]. However, temperatures up to 1200 K have also been shown to produce potent molecules in preliminary trial generations.

3.2 Our Reinforcement Learning (RL) Model

Putting atoms together to make molecules has many advantages, such as allowing greater exploration of chemical space (the ideal space of all stable molecules) and a lower chance of introducing human bias. However, it does have disadvantages. The molecules made by this method are guaranteed to have the right valency on each atom and geometry, but this alone is not enough to make a molecule a drug. It must be synthesizable (otherwise, it cannot be a commercially available molecule) and possess certain “druglike” properties.

While methods to estimate drug-likeness do exist (e.g. QED^[32]), they are likely not very accurate at differentiating the different requirements of drugs based on their targets. Owing to this ambiguity, we also believe that for a generic program, an estimate of synthesizability is more useful as ensuring that the molecules are easily synthesizable makes it more likely that they make it into the experimental (usually *in-vitro*) trial phases early.

3.2.1 The reinforcement setup

Our RL setup is inspired by very similar work on molecular graphs and is called a Graph Convolution Policy Network (GCPN)^[33]. A policy is the set of rules the computer will follow during its task (in our case, molecule generation). A good policy will make the program pick atom types to make the final molecule synthesizable. This policy is learnt through reinforcement. In this case, the input is a graph representing a half-made molecule.

There are a few differences from the original GCPN implementation that are worth noting that divide the job of generation between the “program” (the previously described rule-based algorithm) and the “policy” (the machine learnable model that tries to make molecules synthesizable):

1. The program determines the focal atom - the atom to which the newly added atom will be connected. In the original program, this was also selected by the policy.
2. The program provides a list of valid atom types to choose from. The model must

assign weights to each choice, and one atom type will be sampled with those weights. This pre-decided set of choices means that the number of allowed actions can change depending on the state, which is rare in standard RL implementations.

3. All bonds (including cycles) are determined by the program. The original GCPN implementation makes cycles and adds additional bonds optionally. It also handles the formation of double or triple bonds) Here we use atom types instead of simple elements for atoms, so handling this is unnecessary.

Given the simplicity of our requirement compared to the original GCPN model, we decided it would be useful to have a relatively simple model. Note that this model is subject to change depending on the goal. In this project, we present a proof-of-concept to show that the program can generate molecules such that the fraction of synthesizable molecules significantly increases while using it.

3.2.2 Using GCPN in DeNovo

We can now understand the construction of molecules as a sequential decision-making problem, where at each step, the program is presented with a focal atom and a set of allowed atom types from which it must pick one to attach. How the set is provided or how the focal atom is decided is beyond the scope of reinforcement. To the model, these are predetermined. We, however, know that these are decided based on atom-type definitions as given by the forcefield using look-up tables for each atom type.

The molecule graph

The program is supplied with a graph as input. The graph contains node feature vectors and an adjacency matrix as discussed in section 2.5. From this information, graph convolution is used to incorporate the information from the neighbours into each node. Only information from the focal atom is used in deciding the next atom. The focal atom gives the model context, telling it *where* the new atom is being added in this molecule.

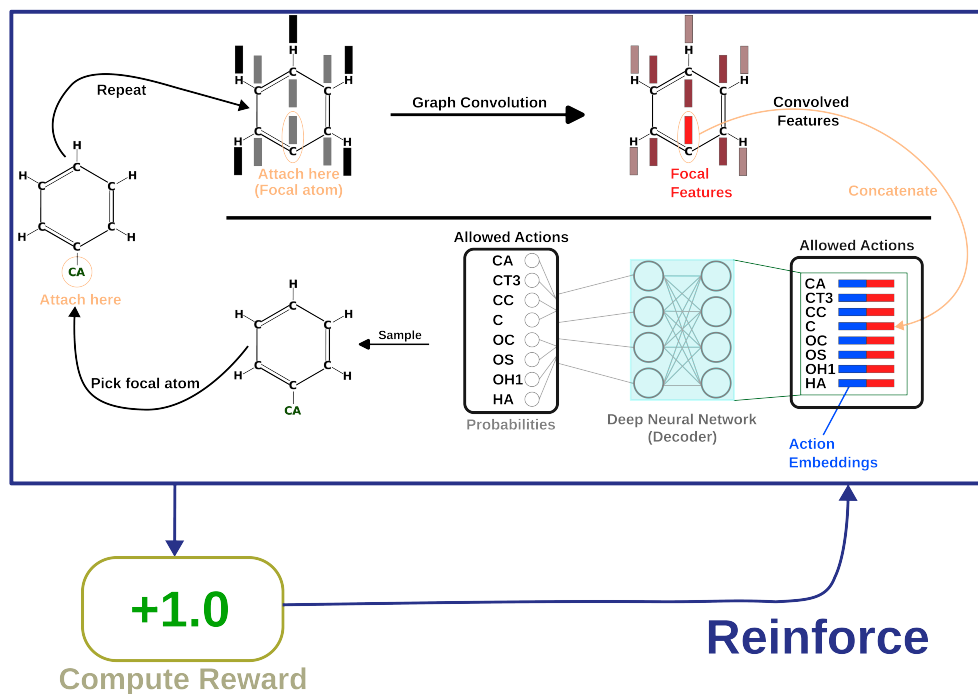


Figure 3.6: *Our modified implementation of a graph convolution policy network. The focal atom is highlighted with the caption “Attach here”. These steps are repeated multiple times - once for each newly added atom.*

The implementation

The model is built in two parts: the “encoder” and the “decoder”. The encoder encodes necessary information into an intermediate vector, first achieved through graph convolution (see subsection 2.6.1 of Machine Learning) over the input graph. The final convolved representation of the focal atom (say v_0) is taken. The focal atom is determined entirely independently of the model and is usually the last added atom.

The program can already determine a list of atom types which can bond to the focal atom. This list is a subset of the atom types in the data file and depends purely on the rules to satisfy the requirements of each atom type. Each of these individual atom types in the data file also has a feature vector. We used one-hot encoding^[34] (described briefly below) here, but we acknowledge the scope of a more potent representation that maps similar atom types to more similar feature vectors.

v_0 is concatenated to every atom type’s feature vector from this list. So we now have a list of combined feature vectors capturing the information of the focal atom and *one* potential connection partner. This list is passed to a decoder which outputs one number for each input

vector. We use a simple 2-layer fully-connected layer^[35] as the decoder.

Quick note on one-hot encoding:

One-hot encoding is a way of encoding different objects with bit vectors. Every object is given an ID based on some arbitrary ordering. The IDs will never change. The feature vector for each object is just a vector full of 0s, except at one index where it is a 1. This index is the ID of that object. So every object has a unique vector representing it, and all the vectors are equidistant from each other. This method of encoding makes sure that no bias enters the system.

Feature and model sizes:

The intermediate layer size for the decoder is 128 neurons. We have 166 atom types, so that is the feature vector size for each node. During graph convolution, we preserve the size (the transformation is done by a learnable 166×166 matrix). This size makes the input size for the decoder $2 \times 166 = 332$ because it uses two concatenated vectors.

Discount Factor and Reward Scheme:

A classic $+1, -1$ reward scheme is used. At the *end* of the molecule’s generation, it receives a reward of $+1$ if the molecule is synthesizable and -1 if not. There is a special penalty of -6.25 if the molecule does not reach the target size but cannot grow further (all valencies are complete).

Initially, no rewards were given in the middle, but we later changed this. We notice that some atom types have complex satisfaction conditions, which causes the model to take many more steps near these atom types, thus causing the rather unfair damping of any reward obtained from these atom types as the reward is received after far too many steps. We introduced a small negative intermediate reward of -0.01 . This penalty ensures that the model tries to reach its goal as fast as possible and limits the benefits it receives from delaying an inevitably negative reward.

The program also has rollback options. When a molecule ends up in a position where no atoms can satisfactorily complete it, or if it cannot complete it after many tries, the program returns to the oldest state where all atom-type requirements are met. To get the program not to add and remove too many atoms redundantly, a negative reward of -0.05 is given for every rollback that it performs.

A discount factor (γ) value of 0.995 is used.

Optimizer:

We used the Adam^[36] as the optimizer as implemented in PyTorch^[37]. We use a learning rate of 9.5×10^{-4} and default parameters. The training was done in multiple cycles of optimizing the same model while tuning the learning rate to slowly reduce. This tuning hopes to get an early quick optimization phase, followed by small local optimization for convergence. The complete procedure used for training is explained as part of the results. Some of this tuning had to do with the feedback from our results.

3.2.3 The SYBA model

For deciding the reward for reinforcement, we take SYBA’s prediction of the synthesizability of a molecule as the ground truth. SYBA is an established pre-trained model with the ability to predict if any given molecule is synthesizable or not.

The model is based on a naïve Bayes classifier that scores individual fragments of the input molecule as “ES” (easy-to-synthesize) or “HS” (hard-to-synthesize).

As the training data, SYBA used molecules from the ZINC15 database as ES molecules. HS molecules were artificially created using other algorithms. It has been shown to be a simple, consistent and interpretable model for predicting synthesizability.

Chapter 4

Results

4.1 Vacuum (Receptor-free) Results

This program can make a sample of molecules even in the absence of any target (proteins or other biomolecules). In this case, the molecules are made in a vacuum, with the only energy terms coming from self-energy (intramolecular interactions). The usual Metropolis Monte-Carlo (as described above) is not used for traversing chemical space. Instead, we accept every successfully generated molecule.

This process allows us to study the construction of the molecules directly without any bias creeping in from a protein target. It is crucial to train the reinforcement model unbiasedly so that it does not optimize molecules for only some particular protein targets.

4.1.1 Baseline (Synthesizability before reinforcement)

Before explaining how reinforcement improves anything, we need to establish how the program performed initially. We use a set of metrics to convey its quality. Self-similarity has always been a good measure of the diversity of molecules within a set. We want our program to generate a diverse array of molecules without pigeonholing itself into a very narrow region of chemical space. As we proceed with the discussion, the importance of this point will be further evident. We also require that as many of the molecules as possible be synthesizable. At this point, about 16% of the molecules are synthesizable - a significantly small portion. If we filtered only synthesizable molecules, we would lose over 80% of our molecules to the

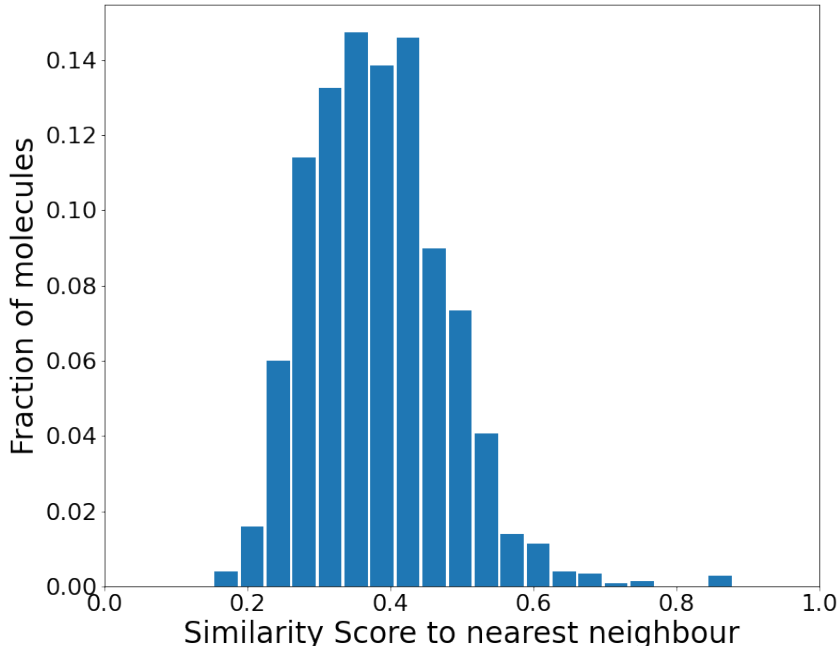


Figure 4.1: *The self-similarity distribution for 300 molecules. The mean value is 0.38 ± 0.01 , with no significant peaks above 0.9, indicating that the molecules are fairly diverse. Around 16% of the molecules are synthesizable.*

screening, making the generation process inefficient.

Computing the self-similarity distribution:

We define the distribution of self-similarity (SS) of the set of size N by defining SS as follows:

$$SS := \{ \max_{j \neq i} \{ sim(m_i, m_j) \}; 0 \leq j \leq N \}; 0 \leq i \leq N \} \quad (4.1)$$

Here, $sim(m_i, m_j)$ denotes the similarity between the i^{th} and j^{th} molecules. Any metric can be used here. We use Tanimoto similarity, which has shown to be a good measure for molecules^[38]. The standard fingerprints from RDKit 2019^[39] were used for this step. As this metric depends on the set size, standardizing the plot with a set size is important. For a larger set, just by the sheer size of the set, we expect there to be a molecule that is more similar to a given molecule. For each set used in computing the self-similarity throughout this thesis, we use a set size of 300 molecules unless specified otherwise.

This distribution does not capture the entire story, however. We find that there is a strong dependence of synthesizability on the size of the molecule. We generated a large distribution of over 1 million molecules in a vacuum to study such molecule-level properties for those generated using this algorithm. The size distribution closely followed the ChEMBL^[40] database, against which we make some comparisons. Figure 4.2 shows how the size affects the syn-

thesizable fraction. This strong dependence on target size is important to remember while analyzing reinforcement results. It is interesting to see a deviation from the trend around

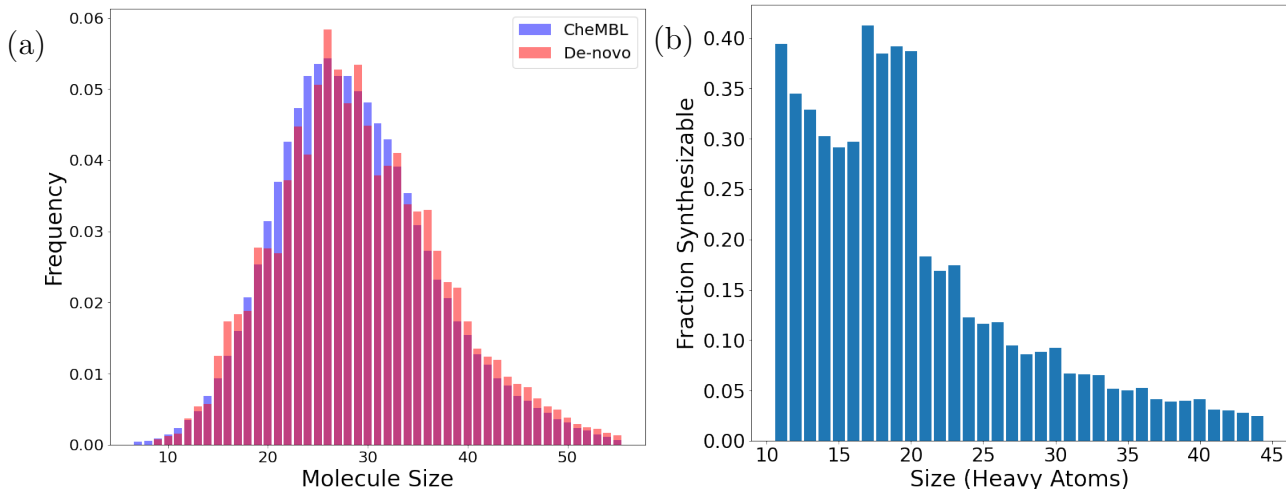


Figure 4.2: Generating molecules in vacuum following the CheMBL size distribution. (a) *Reproducing a size distribution similar to CheMBL. This size distribution was manually targeted.* (b) *The effect of size on the synthesizable fraction*

sizes 20-24. This is likely due to the large number of known synthesizable molecules that naturally fall within this size range. However, noticeably, this fraction declines incredibly quickly with increasing target size. As we will show later, our model achieves great success even at these larger sizes.

4.1.2 Results after training - Mode collapse

While training the program, we noticed something peculiar after 1500 cycles (1500 molecules generated). The program had technically solved the problem, but it was not the solution we wanted. The program had figured out an efficient way to get synthesizable molecules. It realized that stringing benzene fragments together was a sure-shot way to get synthesizable molecules. However, this is a great detriment to our goal of exploring chemical space. We have ourselves a mode collapse! Mode collapse is a problem seen with generative models where they learn a solution that technically does minimize the loss, but miss “modes”, resulting in the generation of a very narrow band of results from the complete space available to them^[41].

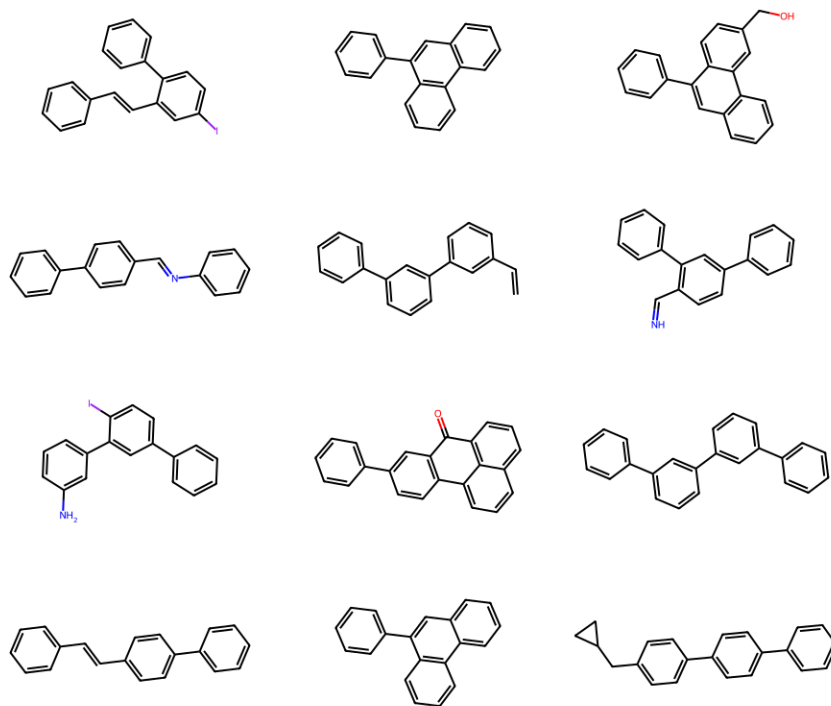


Figure 4.3: *Molecules after initial training.*
 Notice the extremely similar scaffolds and repeated chains of benzene rings. 91% of these molecules are synthesizable, but there is little diversity.

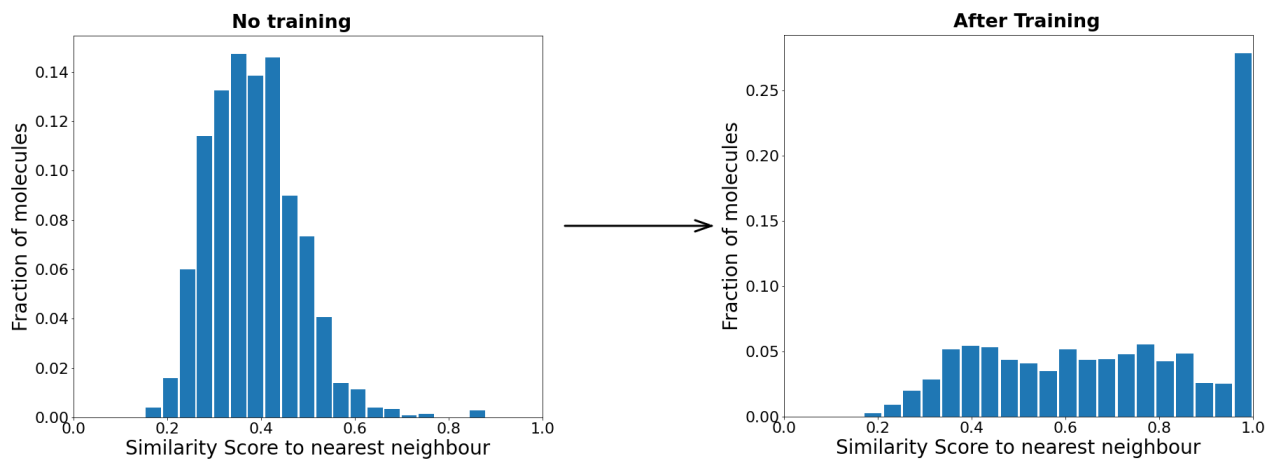


Figure 4.4: *Change in the self-similarity after training.*
 The sharp shift to the right indicates loss of variance, and the peak at 1.0 indicates mode collapse.

The peak at 1.0 indicates that there is actually a duplicate molecule present in the same

set for a significant fraction of the molecules generated. Given the actual size of drug-like chemical space even with just 16-20 atoms^[42], we do not expect any such degree of duplication in just 300 molecules. In fact, the original distribution has no significant presence after 0.9, thus backing this point.

4.1.3 Results - Preventing Mode collapse

To avoid mode collapse, we introduce a regularization parameter. Here, "regularization" is used as a parallel to regularization commonly used in simple regression problems. A regularization term is an addition to the loss function to prevent the model from overfitting. In our case, to prevent mode collapse, we modify our reward scheme. Instead of allowing the agent the complete +1 reward for making a synthesizable molecule, we penalize it for making similar molecules. The reward is now:

$$R_{new}^{synth} = 1 - \lambda \sqrt{\frac{1}{K} \sum_{i=n-K}^{n-1} (\text{cosim}(m_{new}, m_i))^2}$$

Here, we used another proxy for similarity (denoted *cosim*). Instead of the standard Tanimoto similarity criterion, we took cosine similarities of a vector representation for each molecule. Each molecule was represented by a 166-dimensional feature vector (for example, m_{new} represents the newly constructed molecule), with each vector element corresponding to one atom type. The value at each index is the fraction of the molecule’s atoms that have that atom type. For example, benzene would have values of 0.5 and 0.5 for "aromatic carbon" and "aromatic-attached hydrogen" atom classes and 0 elsewhere.

This form of regularization looks at the similarity to each of the previous K molecules and uses the mean of squares of those similarities to calculate the final penalty. The squaring emphasizes avoiding higher values of similarity.

λ and K are tunable hyperparameters that measure the strength of the regularization, and the number of molecules to regularize with respect to. Higher values of λ promote more variance but reduce the synthesizable fraction.

We set $K = 50$ for our training. We observed that starting with a very high value of λ caused no learning. Even after 7500 molecules, the synthesizable fraction did not go above 18%. Instead, we started with $\lambda = 1.6$, which still results in mode collapse. We trained for around 7500 molecules and saved the model after every 50 molecules. We found the model with the best synthesizable fraction without a significant difference in the self-similarity dis-

tribution. This model was the one saved after 3500 molecules. We continued training with around 4200-4500 molecules each for $\lambda = 2.00$, then $\lambda = 2.35$ and finally $\lambda = 2.60$, starting with this half-trained model. For this follow-up training, we reduced the learning rate to 2.5×10^{-4} . The initial training phase brought the average reward from -0.8 to -0.45, which

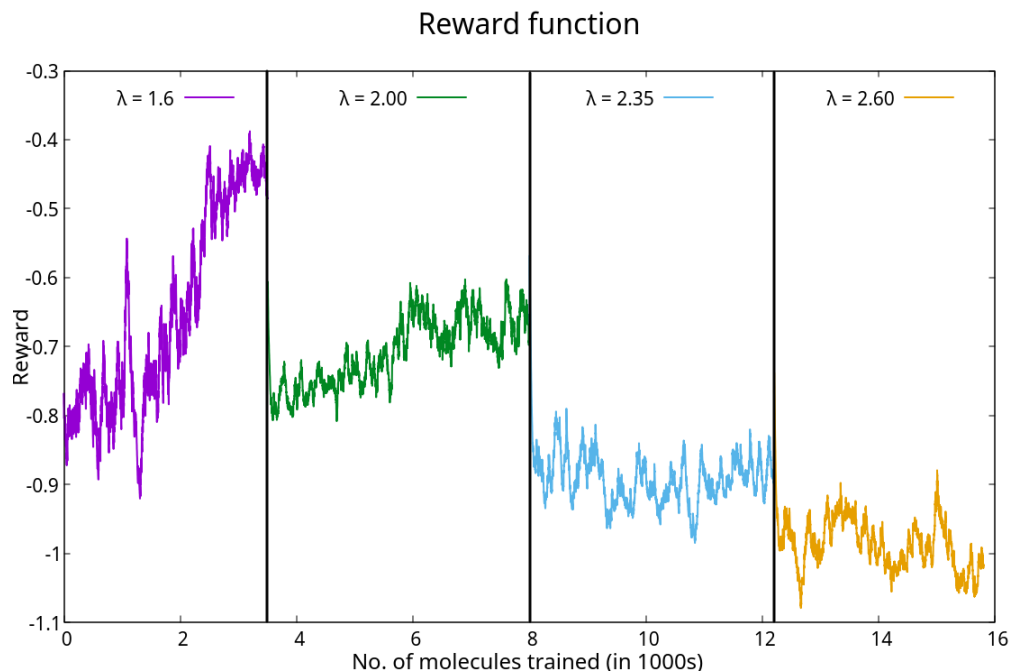


Figure 4.5: *The reward function during training.*

The jumps indicate the changes in regularization (λ) reflected as a drop in the net reward, even when making synthesizable molecules with the same frequency.

indicates how the model learnt to make synthesizable molecules. A slight improvement is again seen with the increase in reward at $\lambda = 2.00$. Beyond this, the reward function shows drops corresponding only to stronger regularization. We guess that this gradual increase of regularization weight ensured the model could quickly learn to make synthesizable molecules and then learn the changes necessary to make a diverse set of them.

Similarity Distribution

This penalization actually does the trick pretty well. Figure 4.6 shows an almost identical self-similarity distribution, with only a slight increase in mean similarity to the nearest neighbour. We can attribute this to the intrinsic bias of any model that picks a subset of an existing large set.

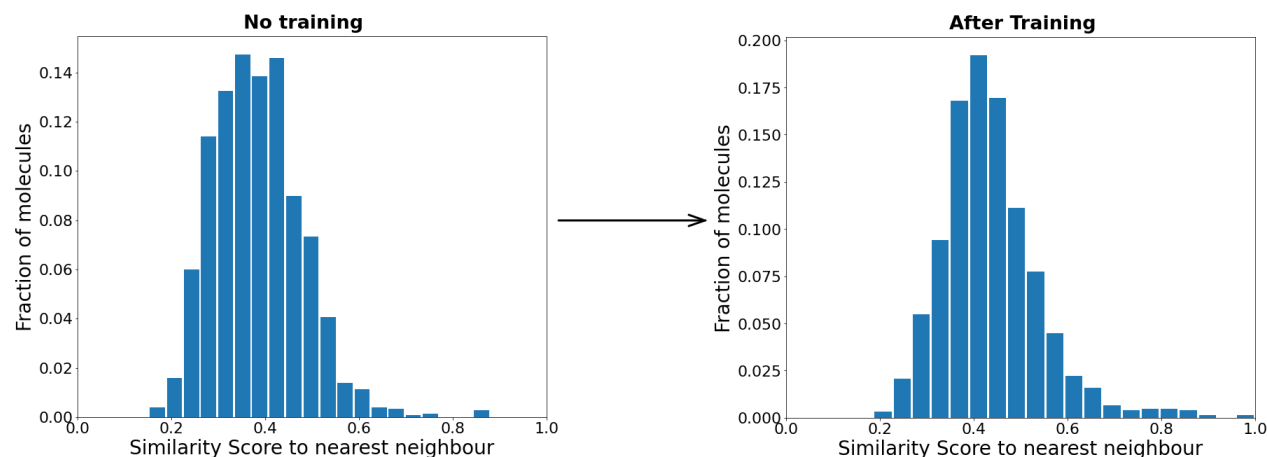


Figure 4.6: *The self-similarity after training with regularization.*

The mean similarity only shifts slightly from 0.38 ± 0.01 to 0.44 ± 0.01 . Again, there are no significant peaks above 0.9, indicating the model’s capability to make varied molecules.

Synthesizability boost

The main goal of the training was to get the program to make synthesizable molecules. The boost to synthesizability we observed is phenomenal. The trained model is capable of making over half the molecules synthesizable. Figure 4.7 compares the synthesizable fraction at different target sizes. The far lower drop in the synthesizable fraction on increasing size indicates that the model has actually learnt something about the “synthesizability” property that is globally applicable beyond its training purview (size ranges of 21 ± 5). The model can be deemed successful in most size ranges of relevance (size < 51). The net synthesizable fraction in the training size-range (16-26 heavy atoms) was 66.5%.

Atom type distribution analysis

The program now generates a diverse set of synthesizable molecules. Figure 4.8 shows the different kinds of molecules made by the program. A bias for aromatic rings is evident, but the variety in molecules is maintained through different kinds of linkages and substituents, slight modifications on the rings, and different ways of connecting the rings.

We have a few more questions to ask of the model. Does the distribution of atom types change? Does the program identify some atom types that are “more synthesizable” than others? Comparing the distribution of atom types between the trained and untrained molecules does provide some insights. Figure 4.9 shows the most changed atom type frequencies. The

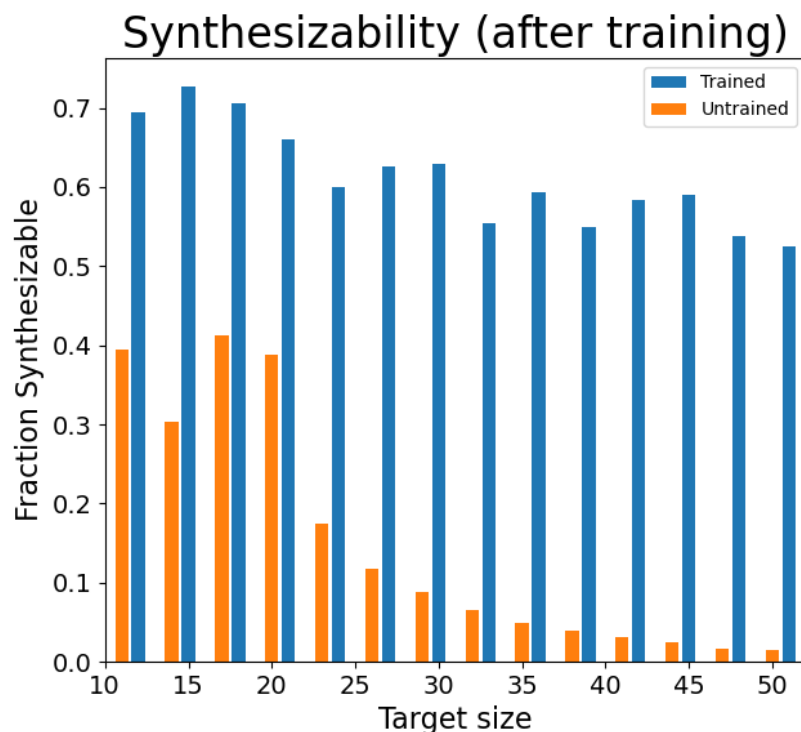


Figure 4.7: *The synthesizability as a function of size.*

Random search over chemical space yields abysmal results at large sizes. However, after training, the drop in synthesizability with the increasing size is far lower. Also, more than half the molecules are synthesizable for all sizes in this range.

most significant change is in the *CA* (aromatic carbon) atom type, with a 4x increase in benzene-like rings after training. Notably, other aromatic atom types such as *NX* (aromatic pyridine-like nitrogen) as *CAS* (benzene carbon atom with polar substituent) have reduced, indicating a bias of pure benzene-like aromatics over heterocycles and highly substituted benzene rings, which the model seems to have learnt to be “hard to synthesize”.

CT3 and *CT3x* represent the terminal methyl group. The steep drop in this atom type might have to do with the high penalty levied on molecules which fail to reach the target size. Terminal groups will likely leave less free valencies from which the molecule can grow. In compensation, *CT2* (aliphatic carbon connected to two hydrogen atoms) is the default aliphatic side-chain now.

CE1A and *CE1B* represented conjugated double bonds. These groups are difficult to synthesize unless placed very carefully, as are *NHZ* (hydrazine-like N-N linkage), *CC* and *O* (randomly placed carbonyl groups).

Finally, an increase in *NH3* atom type indicates that N-substituted amines are easy to syn-

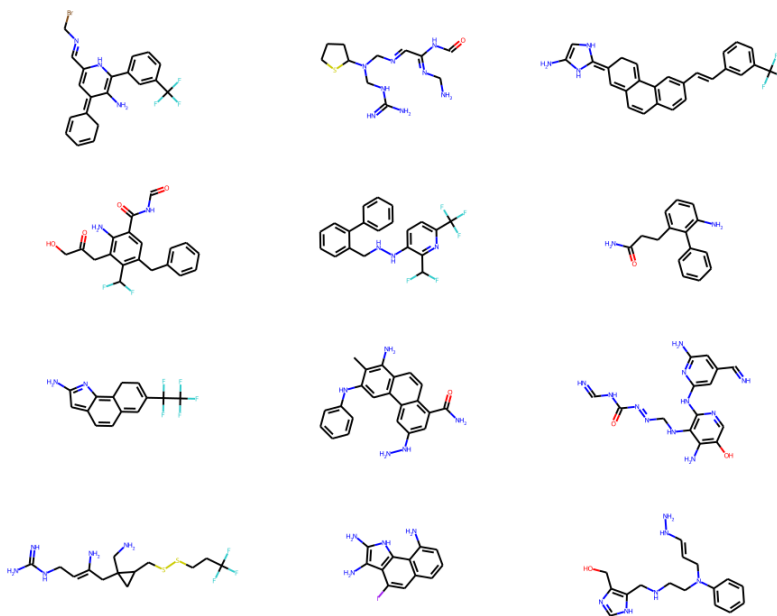


Figure 4.8: *The molecules generated after training with regularization. 66.5% of these are synthesizable and they have more elaborate differences in the substituents, and not just the arrangement of benzene rings.*

thesize. Taken together with the large increase in the CA frequency, it can also imply that aniline and its derivatives are easy to synthesize.

Of course, not all these implications can directly be interpretive of actual synthetic difficulty. There is likely some intrinsic bias in SYBA for some kinds of groups. Using a different model for synthesizability can yield different results. While it is beyond the scope of the current project, given the modular structure of our program, if any synthesizability scoring function is developed in the future, a new model can be trained to improve the quality of these results.

4.2 Real systems

All this effort still leaves us with our original problem - generating ligands *inside* a protein cavity. As a first test system, we chose streptavidin. It is an extremely well-studied system, with a perfectly designed binding domain, which is extremely specific for biotin. There is some study about artificial ligands for streptavidin, but their goal (and ours) is not to find stronger binders than biotin, but rather to find a set of relatively strong alternative binders^[43].

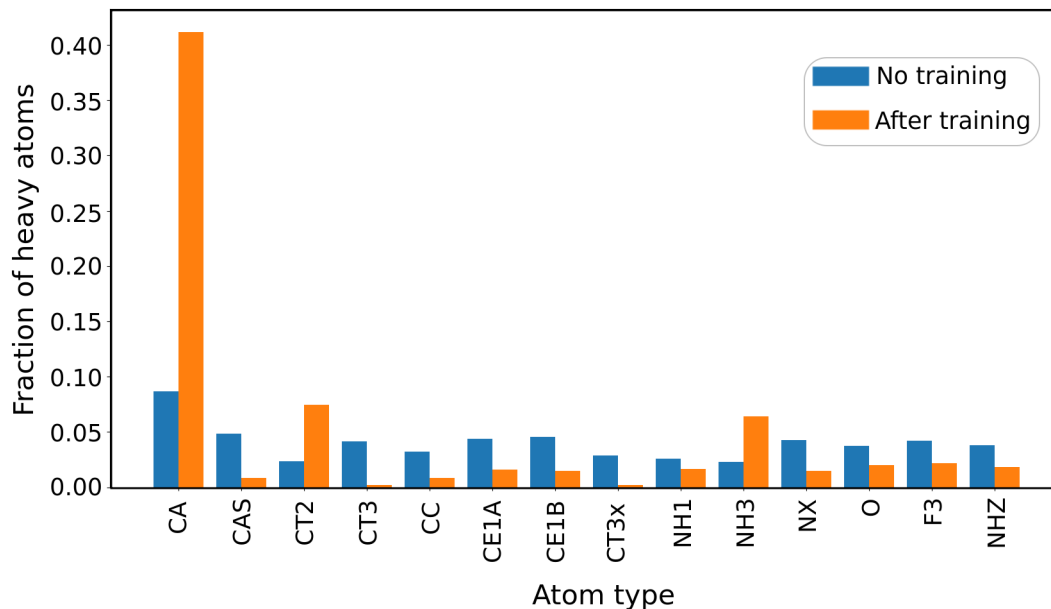


Figure 4.9: *The contribution of specific atom types to the final molecules. The ones with the most changed contributions after training are shown here.*

4.2.1 System preparation

The structure of streptavidin was obtained from the PDB entry 1SWE. All extra residues such as biotin and water were removed from the PDB file, and only the protein was saved. The protein was converted to GRO format, using GROMACS’^[44] `pdb2gmx` conversion tool. The charmm27 forcefield was used to name atom types according to the CHARMM forcefield. Hydrogen atoms in the PDB (if any) were ignored and manually added by GROMACS.

4.2.2 DeNovo Input

- Target sizes were between 16 and 26
- Ligand was restrained to being constructed with all atoms within 6\AA of the active site.
- Optimal molecules were collected after every 8 oscillations (after the interaction energy increased by Monte-Carlo chance 8 times, the last batch of molecules had their energy averaged and only molecules with interaction energy below this average were chosen)

4.2.3 Reinforcement in a protein cavity

The reason for choosing streptavidin was the well studied binding domain, and the extremely specific binding site, which could pose a challenge to the generation model. We firstly validated how well the model performs when applied to a protein system. All generations were in the 16-26 heavy atom range.

We started by validating the consistency by verifying self-similarity. There is no significant

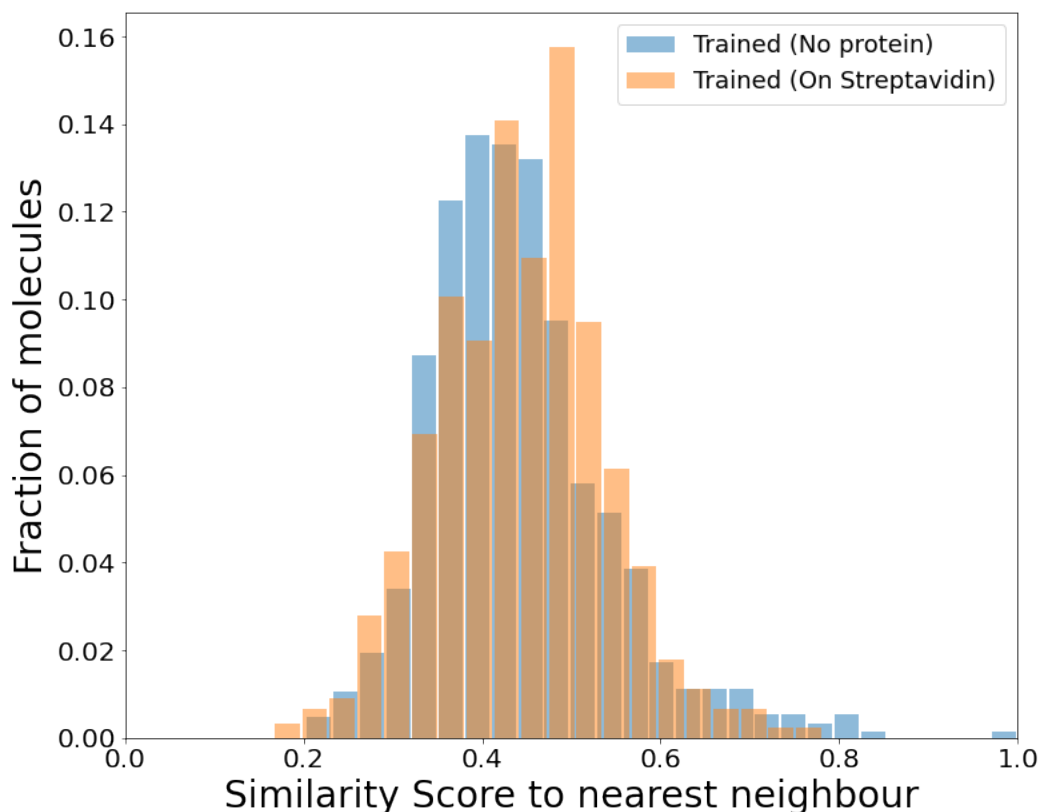


Figure 4.10: *Self-similarity for molecules generated in the streptavidin binding pocket*
The deviation from model-free generation (no training, random selection) is small. Mean
similarity changes from 0.441 to 0.444.

change in the similarity between the trained model in vacuum and streptavidin, with the mean self-similarity deviating only from 0.441 to 0.444

50% of the molecules are synthesizable. This is significant drop from the original 66% for the same size-range. This drop might be due to certain geometries being incapable of binding to the active site, and hence being rejected despite their synthesizability. As the algorithm by default biases for strong interactions, this can be interpreted as a drop in synthesizable fraction to achieve good binding to the domain. However, compared to the original 16%, this

is still a great improvement, and this shows how reinforcement learning makes a difference.

4.2.4 Molecules without reinforcement

Most of the original molecules have chemical groups that are hard to synthesize:

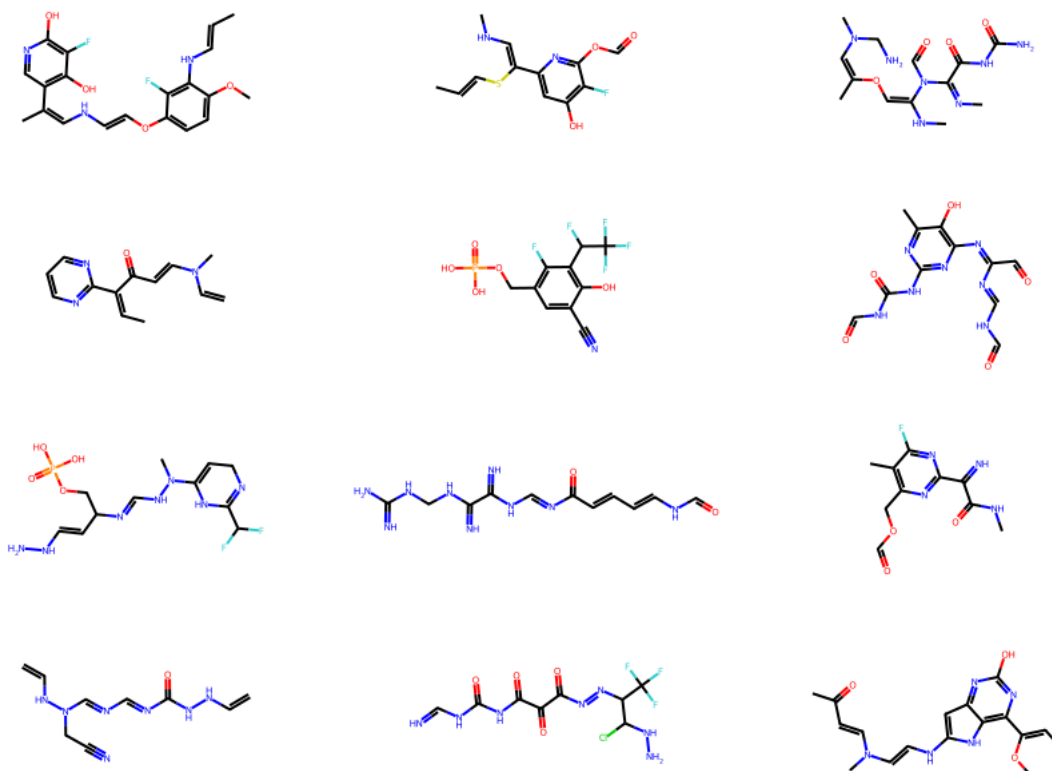


Figure 4.11: *Sample molecules generated for streptavidin without any learning*

4.2.5 Molecules with reinforcement

After reinforcement, the model makes synthesizable molecules: The key differences that we observe are in the number and types of aromatic rings. The trained model avoids long non-cyclic chains and conjugated alkenes in favour of aromatic rings. Figures 4.11 and 4.12 highlight these differences.

The molecules generated fit nicely in streptavidin's binding pocket. A 3D structure of one of the generated ligands bound to the pocket is shown in figure 4.13.

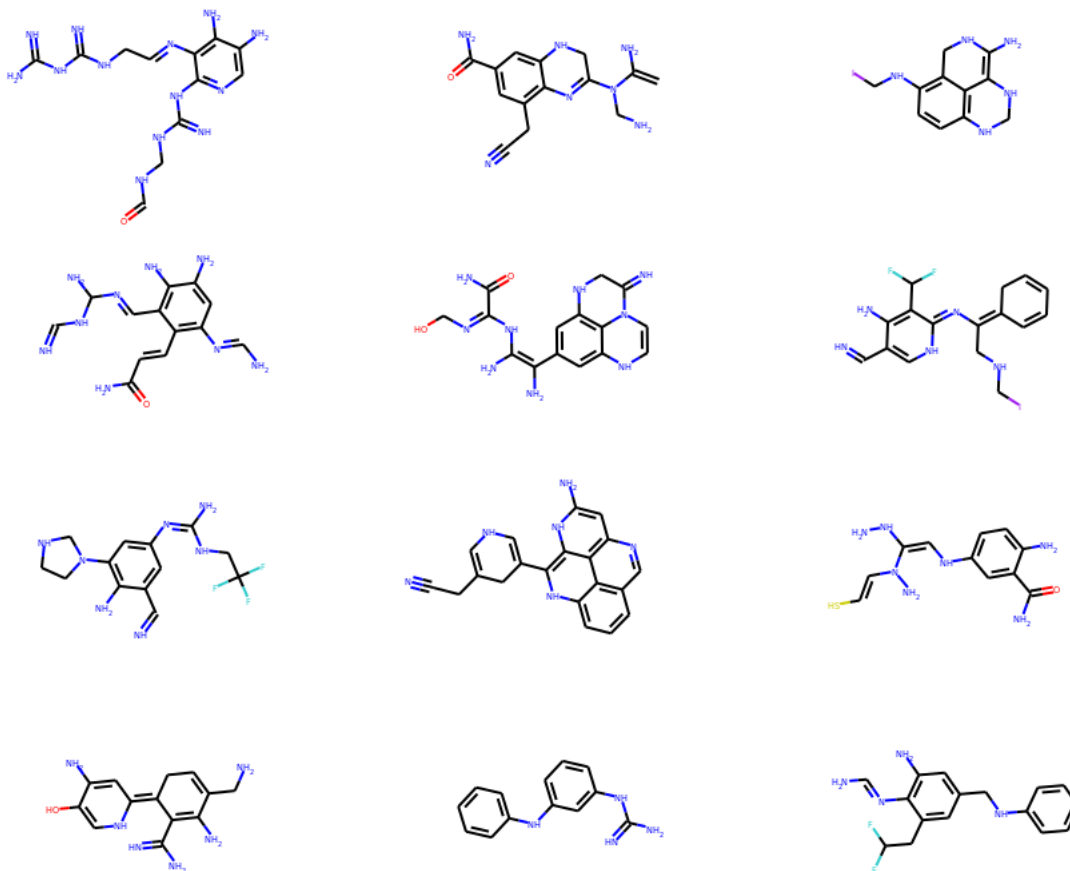


Figure 4.12: *Sample molecules generated for streptavidin after reinforcement*

4.2.6 Comparison to existing molecules

Streptavidin has very few artificially known binders, and even those that exist usually have weaker interactions than biotin^[43]. In order to compare our protocol to a system with known binders, we targetted the ATP binding site of HSP90 (heat-shock protein 90). HSP90 (Heat-shock protein 90) ensures proper folding of other proteins in a cell^[45]. HSP90, with its extremely well-conserved ATP-binding domain is an important therapeutic target, and has been targetted for cancer treatment^[46].

It is important to note that the mean size of the known binders was 27, with one ligand going upto 42 heavy atoms.

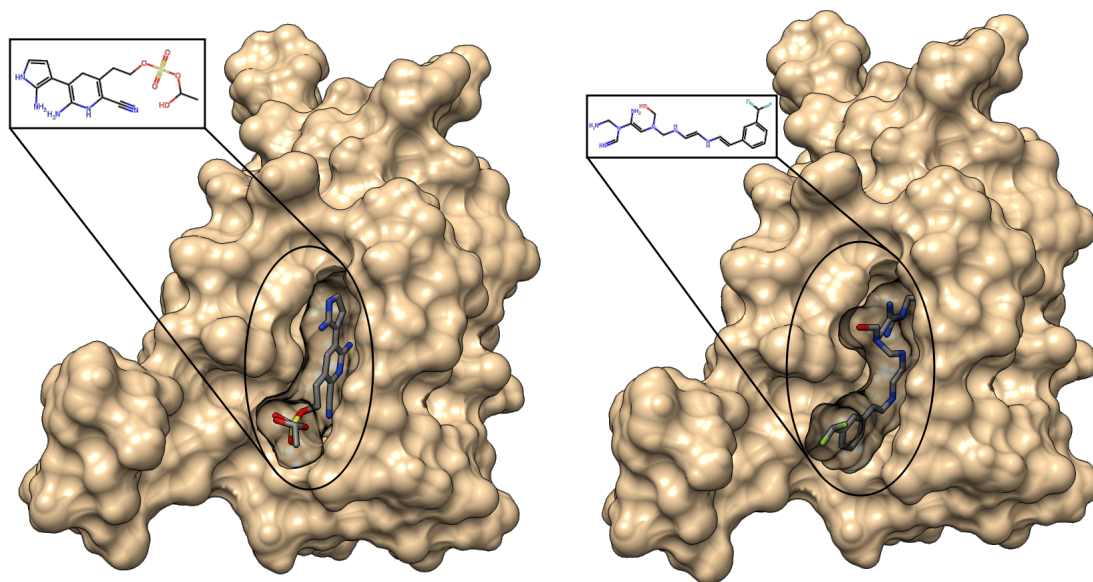


Figure 4.13: *Ligands made by our algorithm sit tightly in the binding site*

4.2.7 Modifying known molecules - A fragmentation approach

This algorithm also has the ability to modify a given lead molecule through small structural changes. This is achieved by randomly erasing part of the ligand and regrowing till the same size is reached. The user decides how much variation in the original molecule is allowed by specifying a retention fraction (between 0 and 1). This fraction of heavy atoms are retained in every molecule generated. For example, in a ligand with 20 heavy atoms, a retention fraction of 0.5 would enforce all generated ligands to have a substructure of size at-least 10 that matches the original ligand. Figure 4.14 shows how similarity to the original molecule is retained by this method by showing the Tanimoto similarity to the starting template.

We picked drug_30, which had the highest docking score of -10.1 kcal (structure shown in figure 4.15a). The sections b-d of figure 4.15 show the most similar molecules after regrowing. We can notice that in most case, it is able to preserve the core moieties of the molecule as part of the generation. The regrowing process is useful to get predictions for good modifications to an existing lead molecule. Also, it can be used to further fine-tune the DeNovo generation results, by picking molecules with many preferred properties.

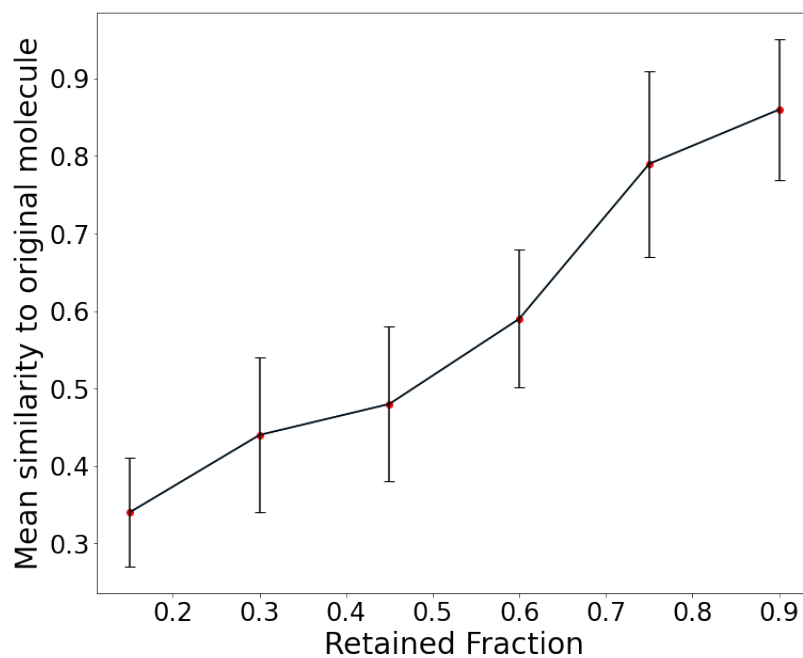


Figure 4.14: Comparing the Tanimoto similarity to the original template as a function of the retention fraction.

We find a steady increase in similarity as expected. This use of the program allows users to tune the amount of deviation from the template.

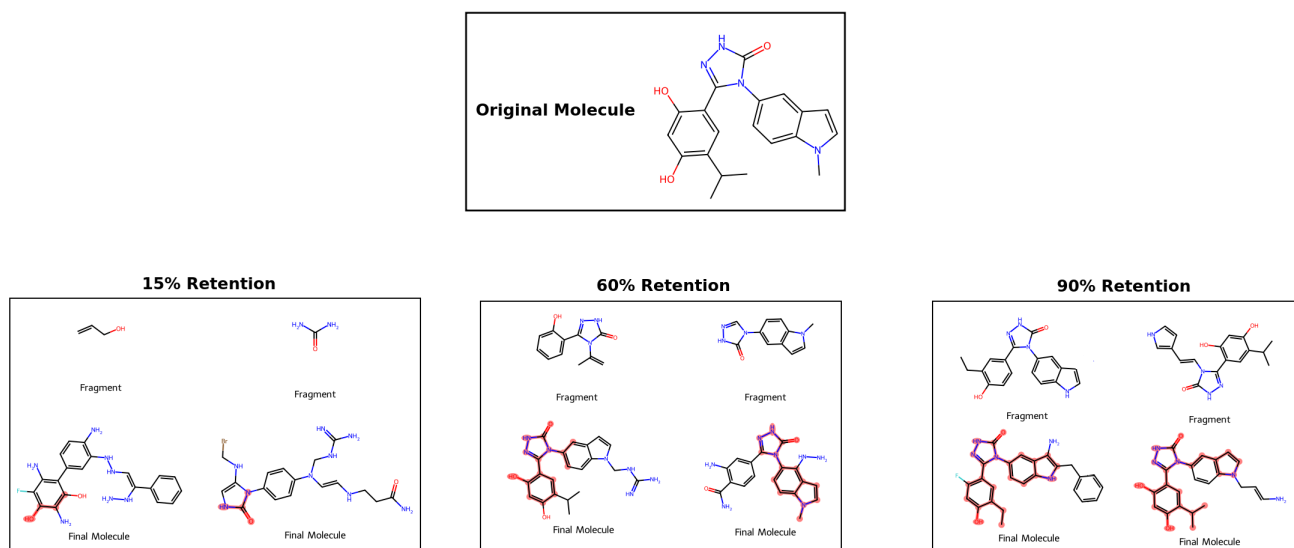


Figure 4.15: Molecules are reconstructed by breaking the molecule and regrowing it. The closeness to the original molecule can be tuned using the retention fraction.

Chapter 5

Conclusion and Future Directions

We have developed an algorithm capable of *de-novo* drug design given only the protein structure and a target region. It uses forcefield parameters from the CHARMM27^[27] forcefield, which have been extended to suit the specific requirements of a generation algorithm. There is still some scope to extend the parameters further to explore more regions of chemical space.

Given the vast regions of chemical space available to the system, it cannot consistently reproduce known binders for a given protein target. However, due to many constraints not captured in synthesizability alone, the currently available ligands are likely preferred over stronger but potentially harmful/biologically unsatisfactory alternatives that the program might be predicting.

The modular nature of the program encourages such tuning as well. Reinforcement with such filters also pretrained can allow for more drug-like molecules in general, and this is a potential extension of this program that would be reasonable to pursue.

Additionally, the generation algorithm does not consider the dynamics of the protein during the generation. It also does not consider potential interactions of the ligand with water. One way to approach this would be by estimating the solvation energy based on the ligand's atom type composition and accounting for it as part of the energy calculation.

Finally, the long-term goal is to generalize the algorithm to consider multiple protein conformations during generation by integrating molecular dynamics into the generation algorithm.

Bibliography

- [1] Gregory Sliwoski, Sandeepkumar Kothiwale, Jens Meiler, and Edward W. Lowe Jr. Computational methods in drug discovery. *Pharmacol Rev*, 66:334–395, 2014.
- [2] Amy C. Anderson. The process of structure-based drug design. *Chemistry & Biology*, 10:787–797, 2003.
- [3] H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, and P.E. Bourne. The protein data bank. *Nucleic Acids Research*, 28:235–242, 2000.
- [4] Shaher Bano Mirza, Ramin Ekhteiari Salmas, M. Qaiser Fatmi, and Serdar Durdagi. Virtual screening of eighteen million compounds against dengue virus: Combined molecular docking and molecular dynamics simulations study. *Journal of Molecular Graphics and Modelling*, 66:99–107, 2016.
- [5] Niamh Coleman and Jordi Rodon. Taking aim at the undruggable. *American Society of Clinical Oncology Educational Book*, (41):e145–e152, 2021.
- [6] Mitchell L. Cohen. Epidemiology of drug resistance: Implications for a post—antimicrobial era. *Science*, 257:1050–1055, 1992.
- [7] Marshall N. Rosenbluth and Arianna W. Rosenbluth. Monte carlo calculation of the average extension of molecular chains. *J. Chem. Phys.*, 23, 1955.
- [8] B. Jayaram, Tanya Singh, Goutam Mukherjee, Abhinav Mathur, Shashank Shekhar, and Vandana Shekhar. Sanjeevini: a freely accessible web-server for target directed lead molecule discovery. *BMC Bioinformatics*, 13, 2012.
- [9] Mariya Popova, Olexandr Isayev, and Alexander Tropsha. Deep reinforcement learning for de novo drug design. *Science Advances*, 4-7, 2018.

- [10] Jacob O. Spiegel and Jacob D. Durrant. Autogrow4: an open-source genetic algorithm for de novo drug design and lead optimization. *J. Chem Inf.*, 12, 2020.
- [11] Renxiao Wang, Ying Gao, and Luhua Lai. Ligbuilder: A multi-purpose program for structure-based drug design. *J. Mol. Model.*, 6:498–516, 2000.
- [12] Florent Chevillard, Helena Rimmer, Cecilia Betti, Els Pardon, Steven Ballet, Niek van Hilten, Jan Steyaert, Wibke E. Diederich, and Peter Kolb. Binding-site compatible fragment growing applied to the design of β 2-adrenergic receptor ligands. *J. Med. Chem.*, 61:1118–1129, 2018.
- [13] Hans-Joachim Böhm. The computer program ludi: A new method for the de novo design of enzyme inhibitors. *J-CAMD*, 1992.
- [14] Varnavas D. Mouchlis, Antreas Afantitis, Angela Serra and Michele Fratello, Anastasios G. Papadiamantis, Vassilis Aidinis, Iseult Lynch, Dario Greco, and Georgia Melagraki. Advances in de novo drug design: From conventional to machine learning methods. *Int. J. Mol. Sci.*, 2021.
- [15] Peng Zhou, Xing-Lou Yang, Xian-Guang Wang, Ben Hu, Lei Zhang, Wei Zhang, Hao-Rui Si, Yan Zhu, Bei Li, Chao-Lin Huang, Hui-Dong Chen, Jing Chen, Yun Luo, Hua Guo, Ren-Di Jiang, Mei-Qin Liu, Ying Chen, Xu-Rui Shen, Xi Wang, Xiao-Shuang Zheng, Kai Zhao, Quan-Jiao Chen, Fei Deng, Lin-Lin Liu, Bing Yan, Fa-Xian Zhan, Yan-Yi Wang, Geng-Fu Xiao, and Zheng-Li Shi. A pneumonia outbreak associated with a new coronavirus of probable bat origin. *Nature*, 579:270–273, 2020.
- [16] Luca Monticelli and D. Peter Tieleman. Ch. 8: Force fields for classical molecular dynamics; biomolecular simulations. 924:197–213, 2013.
- [17] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21, 1953.
- [18] Jörn Ilja Siepmann and Daan Frenkel. Configurational bias monte carlo: a new sampling scheme for flexible chains. *J. Mol. Phys.*, 1992.
- [19] D Frenkel, G C A M Mooij, and B Smit. Novel scheme to study structural and thermal properties of continuously deformable molecules. *J. Phys.: Condens. Matter*, 4:3053, 1992.

- [20] Laurianne David, Amol Thakkar, Rocío Mercado, and Ola Engkvist. Molecular representations in ai-driven drug discovery: a review and practical guide. *J. Cheminform*, 12, 2020.
- [21] David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *J. Chem. Inf. Comput. Sci.*, 28, 1988.
- [22] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach Learn*, 1992.
- [23] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.
- [24] David Van Der Spoel, Erik Lindahl, Berk Hess, Gerrit Groenhof, Alan E. Mark, and Herman J. C. Berendsen. Gromacs: Fast, flexible, and free. *J. Comp. Chem*, 26:1701–1718, 2005.
- [25] Zhong and Haizhen A. *ADMET Properties: Overview and Current Topics*, pages 113–133. Springer Singapore, 2017.
- [26] Milan Voršilák, Michal Kolář, Ivan Čmelo, and Daniel Svozil. Syba: Bayesian estimation of synthetic accessibility of organic compounds. *Journal of Cheminformatics*, 12, 2020.
- [27] Alexander D Mackerell Jr, Michael Feig, and Charles L Brooks 3rd. Extending the treatment of backbone energetics in protein force fields: limitations of gas-phase quantum mechanics in reproducing protein conformational distributions in molecular dynamics simulations. *J. Comput. Chem.*, 25(11):1400–1415, 2004.
- [28] M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, G. A. Petersson, H. Nakatsuji, X. Li, M. Caricato, A. Marenich, J. Bloino, B. G. Janesko, R. Gomperts, B. Mennucci, H. P. Hratchian, J. V. Ortiz, A. F. Izmaylov, J. L. Sonnenberg, D. Williams-Young, F. Ding, F. Lipparini, F. Egidi, J. Gogings, B. Peng, A. Petrone, T. Henderson, D. Ranasinghe, V. G. Zakrzewski, J. Gao, N. Rega, G. Zheng, W. Liang, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, K. Throssell, J. A. Montgomery, Jr., J. E. Peralta, F. Ogliaro, M. Bearpark, J. J. Heyd, E. Brothers, K. N. Kudin, V. N. Staroverov, T. Keith, R. Kobayashi, J. Normand, K. Raghavachari, A. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, J. M. Millam, M. Klene, C. Adamo,

- R. Cammi, J. W. Ochterski, R. L. Martin, K. Morokuma, O. Farkas, J. B. Foresman, and D. J. Fox. Gaussian 09, revision d.01.
- [29] Johann Gasteiger and Mario Marsili. A new model for calculating atomic charges in molecules. *Tetrahedron Letters*, 19(34):3181–3184, 1978.
- [30] Zhenqin Li and Harold A. Scheraga. Monte carlo-minimization approach to the multiple-minima problem in protein folding. *Proc. Natl. Acad. Sci. USA*, 84:6611–6615, 1987.
- [31] Rituparno Chowdhury, Venkata Sai Sreyas Adury, Amal Vijay, Reman K. Singh, and Arnab Mukherjee. Atomistic de-novo inhibitor generation-guided drug repurposing for sars-cov-2 spike protein with free-energy validation by well-tempered metadynamics. *Chemistry – An Asian Journal*, 16(12):1634–1642, 2021.
- [32] G. Richard Bickerton, Gaia V. Paolini, Jeremy Besnard, Sorel Muresan, and Andrew L. Hopkins. Quantifying the chemical beauty of drugs. *Nature Chem*, 4:90–98, 2012.
- [33] Jiaxuan You, Bowen Liu, Rex Ying, Vijay Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. *arXiv*, 2018.
- [34] Prateek Joshi. Python machine learning cookbook. chapter Chapter 4. Packt Publishing Ltd, 2016.
- [35] Bharath Ramsundar and Reza Bosagh Zadeh. *TensorFlow for Deep Learning*. O’Reilly Media, Inc., 2018.
- [36] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv*, 2014.
- [37] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *arXiv*, 2019.
- [38] P. Jaccard. *Bull. Soc. Vaudoise Sci. Nat.*, 37:547– 589, 1901.
- [39] Greg Landrum. Rdkit: Open-source cheminformatics software. 2016.
- [40] Anna Gaulton, Louisa J. Bellis, A. Patricia Bento, Jon Chambers, Mark Davies, Anne Hersey, Yvonne Light, Shaun McGlinchey, David Michalovich, Bissan Al-Lazikani, and

- John P. Overington. ChEMBL: a large-scale bioactivity database for drug discovery. *Nucleic Acids Research*, 40:D1100–D1107, 2011.
- [41] Bhagyashree, Vandana Kushwaha, and G. C. Nandi. Study of prevention of mode collapse in generative adversarial network (gan). In *2020 IEEE 4th Conference on Information Communication Technology (CICT)*, 2020.
- [42] P. G. Polishchuk, T. I. Madzhidov, and A. Varnek. Estimation of the size of drug-like chemical space based on gdb-17 data. *J Comput Aided Mol Des*, 27:675–679, 2013.
- [43] Takuya Terai, Moe Kohno, Gaelle Boncompain, Shigeru Sugiyama, Nae Saito, Ryo Fujikake, Tasuku Ueno, Toru Komatsu, Kenjiro Hanaoka, Takayoshi Okabe, Yasuteru Urano, Franck Perez, and Tetsuo Nagano. Artificial ligands of streptavidin (alis): Discovery, characterization, and application for reversible control of intracellular protein transport. *Journal of the American Chemical Society*, 137(33):10464–10467, 2015.
- [44] Lindahl, Abraham, Hess, and van der Spoel. Gromacs 2021.2 source code, May 2021.
- [45] Sophie E. Jackson. *Hsp90: Structure and Function*, pages 155–240. Springer Berlin Heidelberg, 2013.
- [46] M. P. Goetz, D. O. Toft, M. M. Ames, and C. Erlichman. The hsp90 chaperone complex as a novel target for cancer therapy. *Annals of Oncology*, 14(8):1169–1176, 2003.

Image References

[Img1] Mcy jerry. Diagram showing competitive inhibition. https://en.wikipedia.org/wiki/Competitive_inhibition, 2009. Wikipedia on Competitive Inhibition.