De novo design of new chemical entities with AI methods

Maria Korshunova

May 4, 2022
CMU-CB-22-101

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Olexandr Isayev, PhD, Chair
Christopher Langmead, PhD
David Koes, PhD
Alexander Tropsha, PhD

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Copyright © 2022 Maria Korshunova

This research is funded in part by the grants from the National Science Foundation (NSF CHE-1802789 and CHE-2041108); Eshelman Institute for Innovation (EII) award. M.K. acknowledges The Molecular Sciences Software Institute (MolSSI) Software Fellowship and NVIDIA Graduate Fellowship. We also gratefully acknowledge the support and hardware donation from NVIDIA Corporation. Computational experiments are supported by the Pittsburgh Supercomputer Center (PSC), Extreme Science and Engineering Discovery Environment (XSEDE) award CHE200122, funded by the NSF grant number ACI-1053575.

The department specifically disclaims responsibility for any analyses, interpretations, or conclusions. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, donors or the U.S. Government.

Keywords: Drug discovery, *de novo* molecular design, computational chemisty, deep learning, generative neural networks, reinforcement learning, design of kinase inhibitors

I dedicate my dissertation work to the victims of Russian invasion of Ukraine in February of 2022.

Abstract

This thesis describes novel deep generative neural networks and their applications for the *de novo* design of molecules with optimized properties. It discusses the background and motivation for using deep generative models for *de novo* drug design, covering the advantages and limitations of existing techniques such as virtual screening of molecular libraries, genetic algorithms, and combinatorial enumeration of molecules from a set of building blocks. Next, it describes novel deep generative neural network architectures for producing molecules in three commonly used molecular representations – SMILES strings, 2D molecular graphs, and 3D molecular graphs, with details of the design, implementation, and computational experiments. It continues with a slight detour portraying how a generative model can provide an empirical estimate for the number of bioactive compounds in the chemical space and describes the experiment performed to obtain such an estimate. Next, it introduces Reinforcement Learning based strategy to optimize the values of a property of interest for generated molecules. It also proposes several heuristics for more efficient exploration of the chemical space. Finally, it describes how the proposed models and optimization algorithms were used to virtually design and then experimentally confirm novel hits for multiple kinase proteins.

Acknowledgments

I would like to thank my supervisor Dr. Olexandr Isayev who guided me through my PhD training. He was the best mentor I could imagine and my achievements and this work wouldn't be possible without him. I would also like to thank my committee members for making my defense an enjoyable moment and for all the insightful comments and suggestions.

I would also like to give a special thanks to Dr. Alexander Tropsha for hosting me as a visiting research scholar from Russia in 2016. My PhD journey wouldn't be possible without his support and encouragement.

Finally, I would like to thank my family (including my cat and dog) and friends for being with me during these years and patiently listening to me talking about science.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Significance

The analysis of recent trends in drug development and approval presents a bleak picture [3]. The approval of new drugs has been flat over the last two decades. Less than one out of every 10,000 drug candidates become an approved marketed drug. Only three out of every 20 approved drugs bring in enough revenue to cover developmental costs. Moreover, it takes approximately 10-15 years and an average cost of $1-3 billion to develop each new drug. Many promising drug candidates fail in phase II and phase III—later stages of the clinical development process. These high attrition rates incurred at later phases of the pipeline are primarily responsible for the high cost of drug discovery; this high failure rate effectively implies that most molecules making it into later phases should have been filtered out early in the process. On the other hand, the attempts to implement "fail early, fail cheap" concept [4] by employing multiple computational filters developed to be applied to early-stage drug candidates proved to be ineffective in increasing the success rate [5], possibly due to oversimplification of the structural determinants of undesired drug properties [6, 7]. Thus, the pharmaceutical industry is currently challenged to increase the efficiency of drug development.

The combination of big data and artificial intelligence (AI) was referred to by the World Economic Forum as the fourth industrial revolution and can radically transform the practice of scientific discovery [8]. AI is revolutionizing medicine [9] including radiology, pathology, and

other medical specialties [10]. Deep learning (DL) technologies are beginning to find applications in drug discovery [11, 12] including areas of molecular docking [13], transcriptomics [14], reaction mechanism elucidation [15], and molecular energy prediction [16, 17]. Drug discovery is well positioned to be the next frontier for a potential breakthrough. When it comes to optimizing or discovering new molecules, human intuition currently drives the design. The resulting data sets tend to be clustered, sparse, and incomplete, especially in that humans tend to favor inclusion of "successful" data and tend to forget "failed" experiments. The comprehensive incorporation of all data is the strength of machine intelligence. With sufficient data, an AI-driven machine can more effectively choose the next step in experiments or simulations than humans, speeding up the optimization of a given property. A key benefit is achieving a better balance between the exploration of new parameter space, and the exploitation of already promising areas. The methodology presented in this thesis is a paradigm shift for early-stage drug design. It disrupts the whole pipeline of how hit and lead compounds are discovered and optimized.

The crucial step in many new drug discovery projects is the formulation of a well-motivated hypothesis for new lead compound generation (*de novo* design) or compound selection from available or synthetically feasible chemical libraries based on the available structure-activity relationship (SAR) data. The design hypotheses are often biased toward preferred chemistry [18] or driven by model interpretation [19]. Automated approaches for designing compounds with the desired properties *de novo* have become an active field of research in the last 15 years [20, 21, 22]. The diversity of synthetically feasible chemicals that can be considered as potential drug-like molecules was estimated to be between 10^{30} and 10^{60} [23]. However, these estimates are combinatorial and may be overoptimistic. In this thesis we will revisit the estimate of the size of the bioactive chemical space in Section 2.5. Great advances in computational algorithms [24, 25], hardware, and high-throughput screening technologies [26] notwithstanding, the size of this virtual library prohibits its exhaustive sampling and testing by systematic construction and evaluation of each individual compound. Local optimization approaches have been proposed, but they do not ensure the optimal solution, as the design process converges on a local or "practical" optimum by stochastic sampling or restricts the search to a defined section of chemical space that can be screened exhaustively [20, 25, 27].

Notably, a method for exploring chemical space based on continuous encodings of molecules was proposed recently [28]. It allows efficient, directed gradient-based search through chemical space but does not involve biasing libraries toward special physical or biological properties. Another very recent approach for generating focused molecular libraries with the desired bioactivity using recurrent neural networks (RNNs) was proposed as well [29]; however, properties of produced molecules could not be controlled well. An adversarial autoencoder was proposed [30] as a tool for generating new molecules with the desired properties; however, compounds of interest are selected by means of virtual screening of large libraries, not by designing novel molecules. Specifically, points from the latent space of chemical descriptors are projected to the nearest known molecule in the screening database, which are regarded as hit compounds.

This thesis attempts to develop a new computational drug design platform that is based on the innovative applications of AI approaches to the task of the generation of structurally novel compounds with optimized properties. Benefiting both from the availability of Big Data on multiple properties of significance to drug discovery and novel algorithms, our goal is to develop "shotgun" computational models that accomplish, in one sweep, the design of realistic molecular structures that possess desired properties "at birth".

Chapter 2

Deep generative models for molecules

2.1 Background

Deep generative models can be classified by the types of molecular representation generated. The fundamental differences in these approaches lie in types of molecules representation. In this thesis, we define three ways to represent a molecule in a model based on the amount of information this representation provides about the molecule – SMILES strings, molecular graphs and 3D conformer (see Figure 2.1).

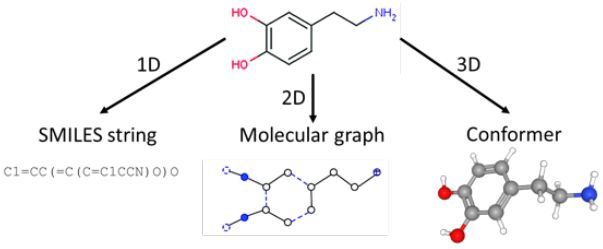


Figure 2.1: Types of molecular representations

Arguably, the most well-studied way to represent a chemical molecule is a simplified molecular-input line-entry system (SMILES) string [31]. A SMILES string consists of symbols corresponding to nodes of the molecular graph in their depth-first order, unambiguously describing the composition and structure of the chemical molecule. SMILES string can be considered a 1D representation of a molecule based on underlying data dimensionality, a vector of character tokens. SMILES string notation in its nature is similar to a natural language. The grammar of SMILES language includes various

bracket types, and a valid SMILES string must contain a balanced parenthesis string. Multiple models for SMILES string generation have been proposed in literature [32, 33, 34]. All these approaches adapted techniques from deep neural networks for dealing with natural language and use a recurrent neural network of some kind to learn a language model of SMILES. Probably the biggest limitation of these methods is imperfect validity (i.e. some of the generated samples are not chemically valid molecules) due to a challenge of learning complex grammatical rules. Another limitation is that SMILES-based approaches cannot be naturally extended to scaffold optimization when a generation process starts from a given core of the molecule and the task is to find a molecule with better properties and pattern of substituents while maintaining the same molecular core. In this thesis, we developed a deep generative model for SMILES, i.e. RNN with augmented memory stack, capable of learning the complex grammar of SMILES language. We talk about this model in section 2.2.1.

Another way to represent a chemical molecule is through its molecular graph. A graph is a natural representation of a chemical molecule. Each node in a graph corresponds to an atom, and each edge corresponds to a bond. Different types of atoms (carbon, oxygen, nitrogen, etc.) are encoded as different kinds of nodes in a graph. The same applies to edges to represent different types of bonds (single bond, double bond, triple bond, and aromatic bond).

Graph-based approaches typically do not suffer from the problem of invalidity of generated molecules. It is also possible to enforce physical constraints on the valency, i.e., how many neighbors each atom can have depending on the atom type. Moreover, these models are more interpretable and more intuitive to chemists. Various algorithms for generating molecular graphs have been developed [2, 35]. Jin et al. [2] proposed a junction tree variational autoencoder. This model first generates a junction tree where every node corresponds to a structural fragment rather than a single atom. Then, the junction tree is converted into a valid molecule with a sampling procedure. This approach produces valid molecules by design; however, there is ambiguity in the process of converting a generated junction tree into a molecule due to sampling. While this is not a problem for the unconstrained generative process, it may cause difficulties with property optimization because molecules with the same junction tree may have a drastic difference in property value. [2] argue that it is beneficial to generate a graph from fragments, however,

atom-by-atom models have already proven as a strong baseline [1, 36, 37].

In [35] the process of graph generation is sequential. Nodes are generated one at a time and then connected to the existing partial graph. With a sequential process, the same graph can be generated with multiple sequences of steps due to the node order permutation. This work does not address the problem of node order permutation. Another limitation of this work is the constraints on the graph size. Only molecular graphs with at most 20 heavy atoms were considered which is not enough for any practical purpose.

In [1], the procedure of molecular graph generation is presented as a Markov Decision Process. The model uses graph convolutional network (GCN) model for goal-directed graph generation with reinforcement learning and adversarial training. This work, similar to [2], only reports top 3 molecules, while top 3 may not represent the model performance as well as the distribution of a properties obtained from a large number of generated samples. Recently, the GraphRNN model [38] was proposed for the generation of undirected graphs. In this thesis we extend GraphRNN to include node and edge type predictions and adapted this approach to molecular graph generation. We talk about our model, MolecularRNN, in section 2.3.

Finally, a molecule can be represented as a conformer in 3D space. Several deep learning models addressed the problem of 3D conformer generation. In one of the first works [39] authors developed a model for 3D linker design. This model is trained as VAE to reconstruct the linker between 2 fragments of molecules in 3D. However, this model only considers the distance and relative angle between two seed fragments, and is not capable to produce molecules from scratch or generate 3D conformations to existing 3D molecules. Another work [40] proposed shape-based variational autoencoder based on SMILES and information about shape of the molecule. In this work Convolutional Neural Network (CNN) was coupled with a shape captioning network consisting of a separate CNN used to condition a recurrent neural network (RNN). In this framework, 3D information was only provided implicitly to seed the RNN, and the method does not allow any further control over generated compounds. As a result, such generative model frequently changed the entire molecule and recovered fewer than 2% of the seed molecules. One more VAE-based model was proposed in [41]. This model is designed to generated 3D molecular structure conditioned on the protein's binding site. The protein information is passed to the model

as a feature vector and VAE uses this vector as condition. The drawback of this approach is that the model produces ligands as ligand density grids and not explicit molecules. Two other works [42, 43] reported in the literature utilize reinforcement learning methods to generated molecules with 3D conformations from scratch. However, both of the models can only generate molecules from a pre-defined set of atoms called ”buckets”. Each configuration of buckets requires training a separate model. The resulting molecules do not contain more than ten heavy atoms, which is not sufficient for real-life applications. One more model we want to mention is TorsionNet [44]. This model also uses reinforcement learning algorithms and message passing neural networks, and only addresses conformer generation for a given molecule and does not tackle generation of novel molecules from scratch. The evaluation of the model was only performed for alkanes and lignins and authors did not show how well TorsionNet can generated conformation for drug-like molecules.

In this thesis, we further extended the MolecularRNN model for 3D graphs generation and described it in section 2.4.

2.2 1D SMILES-based generative models

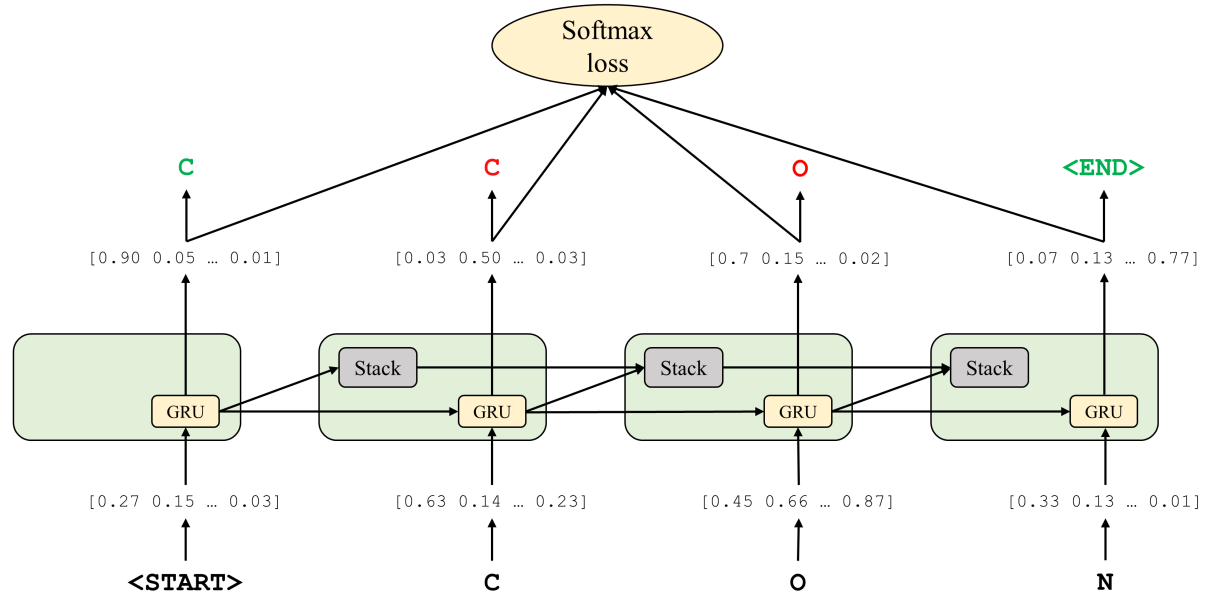
2.2.1 Methods

Generative model for SMILES as a language model A Generative model for SMILES strings is a language model [45]. The generative model has two modes of processing sequences – training and generating. At each time step, in the training mode, the generative network takes a current prefix of the training object and predicts the probability distribution of the next character. Then, the next character is sampled from this predicted probability distribution and is compared to the ground truth. Afterwards, based on this comparison the cross-entropy loss function is calculated and parameters of the model are updated. At each time step, in generating mode, the generative network takes a prefix of already generated sequences and then, like in the training mode, predicts the probability distribution of the next character and samples it from this predicted distribution. In the generative model, we do not update the model parameters.

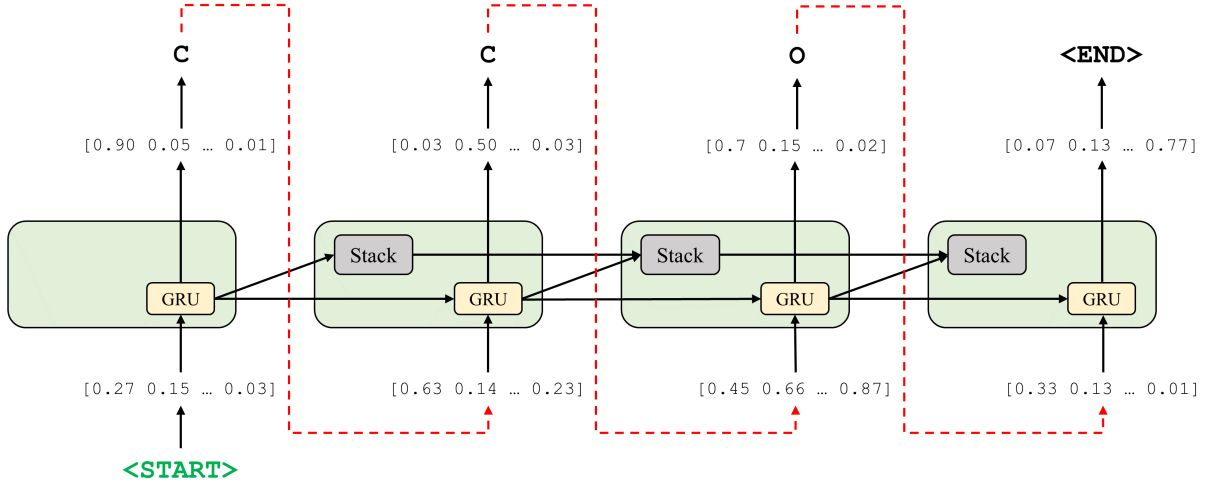
To generate a valid SMILES string, in addition to correct valence for all atoms, one must

count ring opening and closure, as well as bracket sequences with several bracket types. Regular RNNs such as long short-term memory (LSTM) [46] and gated recurrent unit (GRU) [47] are unable to solve the sequence prediction problems because of their inability to count. One of the classical examples of sequences that cannot be properly modeled by regular RNNs are words from the Dyck language, where all open square brackets are matched with the respective closed ones [48]. Formal language theory states that context-free languages, such as the Dyck language, cannot be generated by model without stack memory [49]. As a valid SMILES string should at least be a sequence of all properly matched parentheses with multiple types of brackets, RNNs with an additional memory stack present a theoretically justified choice for modeling SMILES. Another weakness of regular RNNs is their inability to capture long-term dependencies, which leads to difficulties in generalizing to longer sequences [50]. Therefore, memory-augmented neural networks such as Stack-RNN or Neural Turing Machines are the appropriate choice for modeling these sequence dependencies.

The Stack-RNN defines a new neuron or cell structure on top of the standard GRU cell (see Figure 2.5). It has two additional multiplicative gates referred to as the memory stack, which allow the Stack-RNN to learn meaningful long-range interdependencies. Stack memory is a differentiable structure onto and from which continuous vectors are inserted and removed. In stack terminology, the insertion operation is called a *PUSH* operation and the removal operation is called a *POP* operation. These traditionally discrete operations are continuous here, since *PUSH* and *POP* operations are permitted to be real values in the interval $(0, 1)$. Intuitively, we can interpret these values as the degree of certainty with which some controller wishes to PUSH a vector v onto the stack or POP the top of the stack. Such an architecture resembles a pushdown automaton, which is a classic framework from the theory of formal languages, capable of dealing with more complicated languages. Applying this concept to neural networks provides the possibility to build a trainable model of the language of SMILES with correct syntaxes, proper balance of ring opening and closures, and correct valences for all elements.



(a) Training mode



(b) Inference mode

Figure 2.2: SMILES-based generative model.

2.2.1.1 Stack-augmented recurrent neural network

This section describes a generative model with augmented memory stack in more details. We assume that the data is sequential, which means that it comes in the form of discrete tokens, i.e., characters as in the case of SMILES strings. The goal is to build a model that is able to predict the next token conditioning on all previous tokens. A regular recurrent neural network has an input layer and a hidden layer. At time step t the neural network takes the embedding vector of token

number t from the sequence as an input and models the probability distribution of the next token given all previous tokens, so that the next token can be sampled from this distribution. Information of all previously observed tokens is aggregated in the hidden layer. This can be written down as follows:

$$h_t = \sigma(W_i x_t + W_h h_{t-1}),$$

where h_t is a vector of hidden states, h_{t-1} – vector of hidden states from the previous time step, x_t – input vector at time step t, W_i – parameters of the input layers, W_h – parameter of the hidden layer and σ – activation function. The stack memory is used to keep the information and deliver it to the hidden layer at the next time step. A stack is a type of persistent memory, which can be only accessed through its topmost element. There are three operations supported by the stack: POP operation, which deletes an element from the top of the stack, PUSH operation, which puts a new element to the top of the stack; and NO-OP operation, which performs no action. The top element of the stack has value $s_t[0]$ and is stored at position 0:

$$s_t[0] = a_t[PUSH]\sigma(Dh_t) + a_t[POP]s_{t-1}[1] + a_t[NO-OP]s_{t-1}[0],$$

where D is $1 \times m$ matrix and $a_t = [a_t[PUSH], a_t[POP], a_t[NO-OP]]$ is a vector of stack control variables, which define the next operation to be performed. If a_t [POP] is equal to 1, then the value below is used to replace the top element of the stack. If a_t [PUSH] is equal to 1, then a new value will be added to the top and all the rest of the values will be moved down. If a_t [NO-OP] equals 1 then stack keeps the same value on top. Similar rule is applied to the elements of the stack at a depth $i > 0$:

$$s_t[i] = a_t[PUSH]s_{t-1}[i-1] + a_t[POP]s_{t-1}[i+1] + a_t[NO-OP]s_{t-1}[i].$$

Now the hidden layer h_t is updated as:

$$h_t = \sigma(Ux_t + Rh_{t-1} + Ds^k_{t-1}),$$

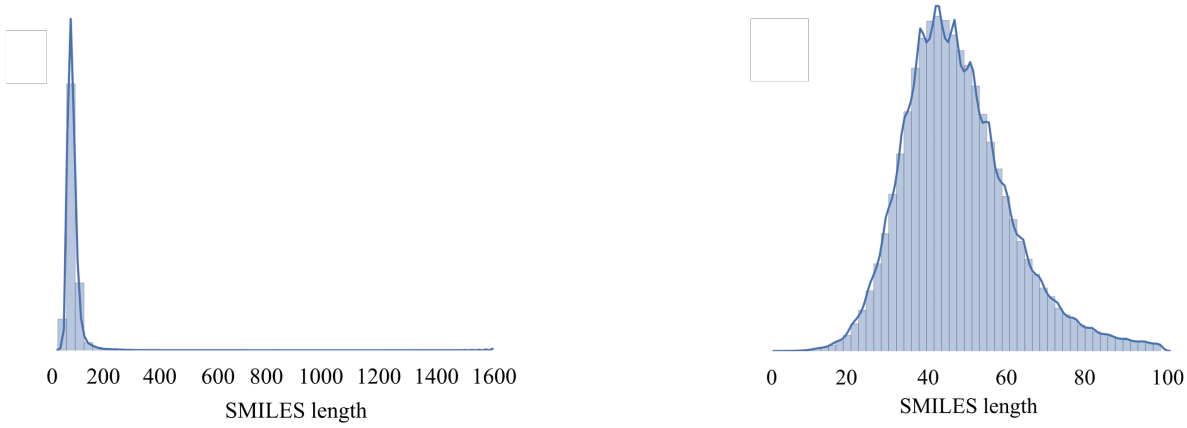
where D is a matrix of size m×k and s^k_{t-1} are the first k elements for the top of the stack at

time step $t-1$.

2.2.2 Results

2.2.2.1 Generation of chemicals with novel structures

The generative network was trained with structures from the ChEMBL21 database [51]. ChEMBL includes approximately 1.5 million SMILES strings; however, we only selected molecules with length of SMILES string of fewer than 100 characters. The length of 100 was chosen because more than 97% of SMILES in training dataset had 100 characters or less. The distribution of SMILES string lengths is shown in Figure 2.3.



(a) Initial (b) Truncated

Figure 2.3: Distributions of SMILES's strings lengths

The objective of training was to learn the rules of organic chemistry that define SMILES string corresponding to realistic chemical structures. To demonstrate the versatility of the baseline (unbiased) Stack-RNN, we generated over one million (1M) of compounds. Random examples of the generated compounds are illustrated in Figure 2.4.

A known deficiency of approaches for *de novo* molecular design is frequent generation of chemically infeasible molecules [28, 34]. To address this possible issue of concern, we have established that 95% of all generated structures were valid, chemically sensible, molecules. The validity check was performed by the structure checker from ChemAxon (https://chemaxon.com/).

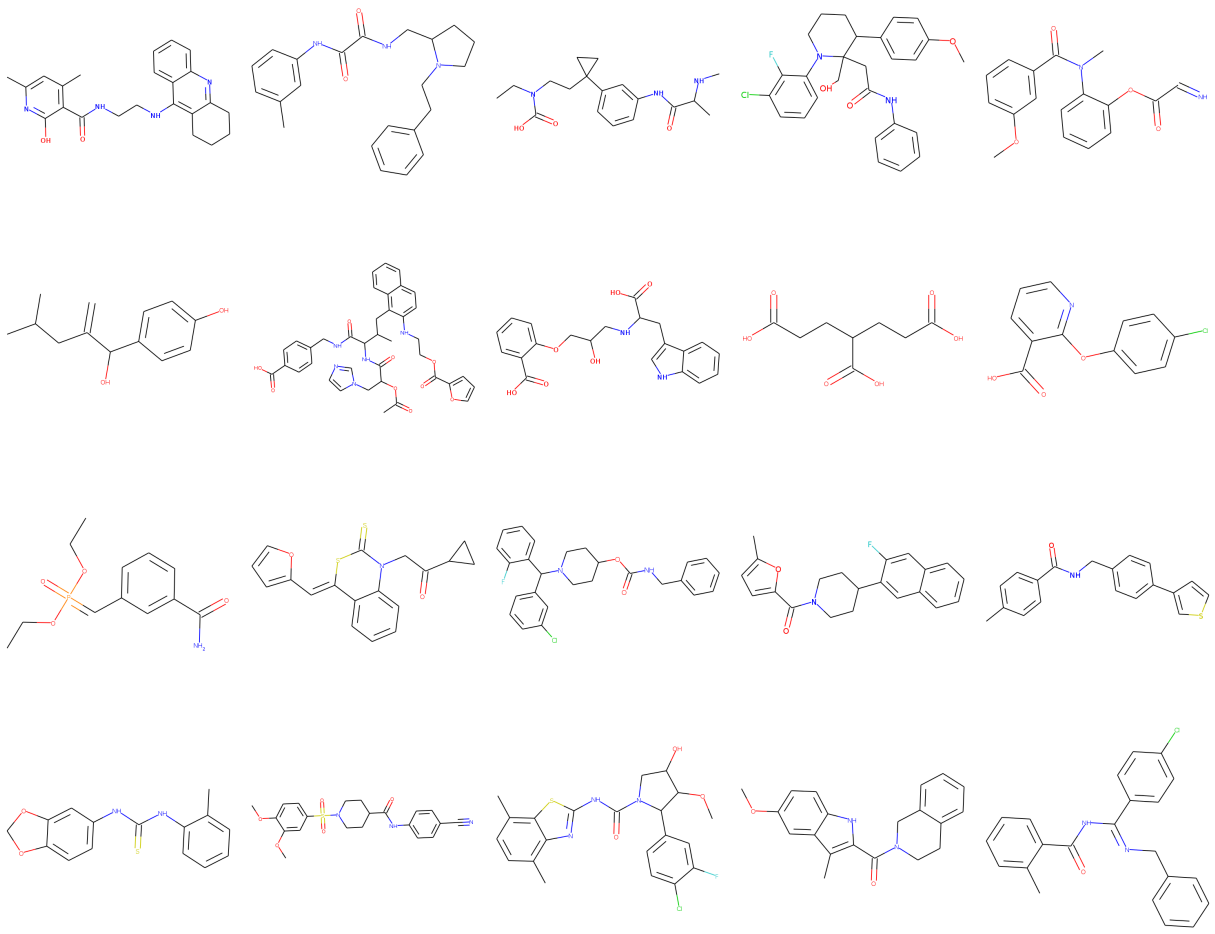


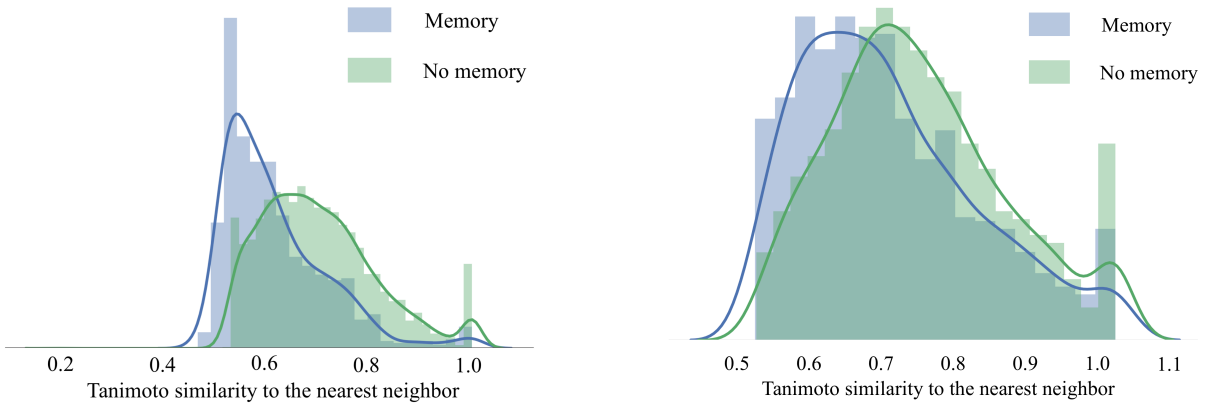
Figure 2.4: A sample of molecules produced by the SMILES-based generative model.

We compared the 1 million *de novo* generated molecules with those used to train the generative model from the ChEMBL database and found that the model produced less than 0.1% of structures from the training dataset. Additional comparison with the ZINC15 database [52] of 320 million synthetically accessible drug-like molecules showed that about 3% (approximately $32,000$ molecules) of *de novo* generated structures could be found in ZINC.

2.2.2.2 Ablation study for augmented memory stack

To assess the importance of using a stack memory augmented network as described in section 2.2.1.1, we compared several characteristics of chemical libraries generated by models developed either with or without stack memory. For this purpose, we trained another generative recurrent

neural network with the same architecture but without using stack memory. Libraries were compared by the percentage of valid generated SMILES, internal diversity, and similarity of the generated molecules to those in the training dataset (ChEMBL). The model without stack memory showed that only 86% of molecules in the respective library were valid (as evaluated by ChemAxon) compared to 95% of valid molecules in the library generated with stack memory network. As expected, in the former library, syntactic errors such as open brackets, unclosed cycles and incorrect bond types in SMILES strings were more frequent. Based on the analysis of respective sets of 10000 molecules generated by each method (See Figure 2.5a), the library obtained without stack memory showed a decrease of internal diversity by 0.2 units of the Tanimoto coefficient and a four-fold increase in the number of duplicates, from just about 1% to 5%. In addition, Figure 2.5b shows that the number of molecules similar to the training dataset ($T_{sim} > 0.85$) for the library obtained without stack memory (28% of all molecules) is twice that obtained with stack memory (14%). These results clearly highlight the advantages of using neural network with memory for generating the highest number of realistic and predominantly novel molecules, which is one of the chief objectives of *de novo* chemical design.



(a) Internal diversity of generated libraries

(b) Similarity of generated libraries to training dataset

Figure 2.5: Performance of stack augmented GRU and "Vanilla" GRU

2.2.2.3 Novelty and synthetic accessibility of the generated molecules

In order to further characterize the structural novelty of the *de novo* generated molecules, we compared the content of the Murcko scaffolds [53] between the ChEMBL training set and the

virtual library generated by our system. Murcko scaffolds provide a hierarchical molecular organization scheme by dividing small molecules into R-groups, linkers, and frameworks, or scaffolds. They define the ring systems of a molecule by removing side chain atoms. We found that less than 10% of scaffolds in our library were present in ChEMBL. Overall, this analysis suggests that the generative Stack-RNN model did not simply memorize the training SMILES sequences but was indeed capable of generating extremely diverse yet realistic molecules as defined by the structure checker from ChemAxon. In addition to passing the structure checker, an important requirement for *de novo* generated molecules is their synthetic feasibility. To this end, we employed the synthetic accessibility score (SAS) method [54], which relies on the knowledge extracted from known synthetic reactions and adds a penalty for high molecular complexity. For ease of interpretation, SAS is scaled to be between 1 and 10. Molecules with high SAS values, typically above 6, are considered difficult to synthesize, whereas molecules with low SAS values are easily synthetically accessible. The distribution of SAS values calculated for 1 million molecules generated by the SMILES-based generative model is shown in Figure 2.6.

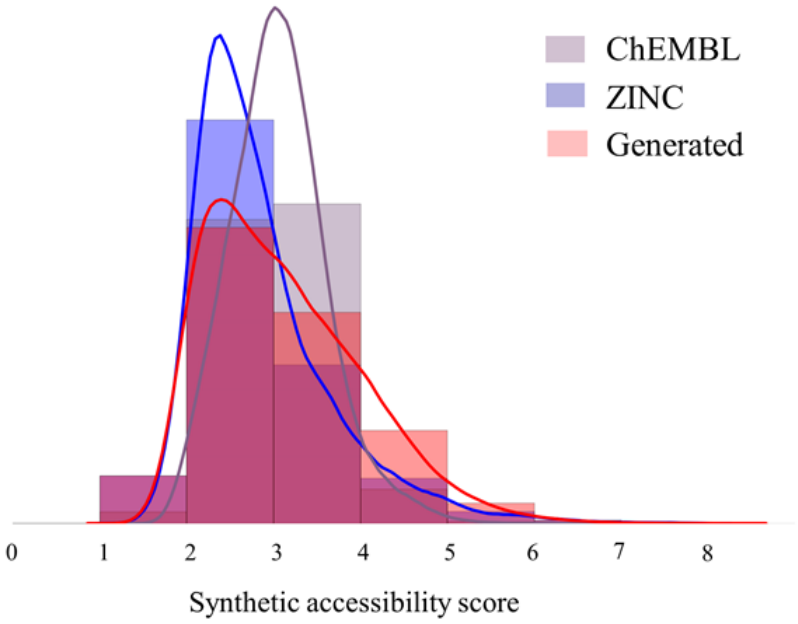


Figure 2.6: Distribution of Synthetic Accessibility Score (SAS) for the full ChEMBL21 database (1.5M molecules), random subsample of 1M molecules from ZINC15 and generated dataset of 1M molecules with baseline generator model G.

To illustrate the robustness of the *de novo* generated chemical library, we compared its SAS

distribution with that of the SAS values both for the full ChEMBL library (1.5M molecules) and for a 1M random sample of molecules in ZINC. Similar to typical commercial vendor libraries, the distribution of SAS for the generated library is skewed towards more easily synthesizable molecules. Median SAS values were 2.9 for ChEMBL and 3.1 for both ZINC and the generated library. Over 99.5% of *de novo* generated molecules had SAS values below 6. Therefore, despite their high novelty, the vast majority of generated compounds can be considered as synthetically accessible.

2.2.2.4 Model analysis

Model interpretation is a highly significant component in any ML study. In this section we demonstrate how Stack-RNN learns and memorizes useful information from the SMILES string as it is being processed. More specifically, we have manually analyzed neuron gate activations of the neural network as it processes the input data.

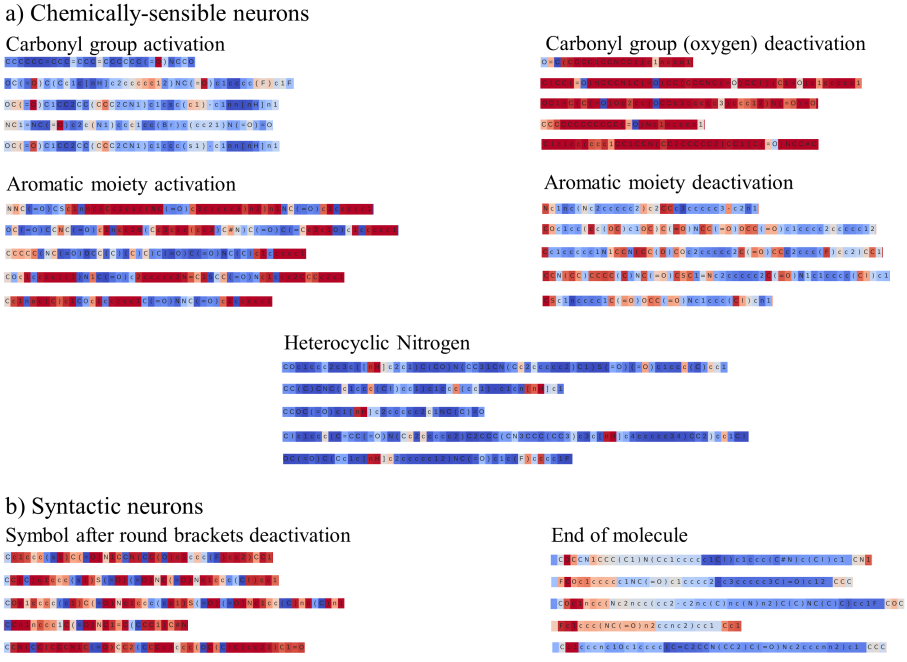


Figure 2.7: Examples of Stack-RNN cells with interpretable gate activations.

Figure 2.7 lists several examples of cells in neural networks with interpretable gate activations.

In this figure, each line corresponds to activations of a specific neuron at different SMILES processing time steps by the pre-trained baseline generative model. Each letter is colored according to the value of $\tanh$ activation in a cool-warm colormap from dark blue to dark red, i.e., from -1 to 1. We found that our RNN has several interpretable cells. These cells can be divided into two groups – chemically sensible groups, which activate in the presence of specific chemical groups or moieties, and syntactic groups, which keep tracks of numbers, bracket opening and closure, and even of SMILES string termination when the new molecule is generated. For instance, we saw cells reflecting the presence of a carbonyl group, aromatic groups or NH moieties in heterocycles. We also observed that in two of these three examples there were counter-cells that deactivate in the presence of the aforementioned chemical groups. Neural network-based models are notoriously uninterpretable [55] and the majority of cells were indeed in that category. On the other hand, the possibility of even partial interpretation offered by this approach could be highly valuable for a medicinal chemist.

2.3 2D graph-based generative models

2.3.1 Methods

The core of our approach is a MolecularRNN model, which extends the GraphRNN [38] model for generating graphs with node and edge types. Section 2.3.1.1 gives background on GraphRNN, and the extension is described in Section 2.3.1.2. We introduce a method of valency-based rejection sampling in Section 2.3.1.3 that yields 100% validity in inference mode.

2.3.1.1 Background: GraphRNN model

GraphRNN [38] was introduced for generation of undirected graphs $G = (V, E)$ with a set of n nodes $V = (v_1, \ldots, v_n)$ and a set of undirected edges $E = (\{v_i, v_j\} | v_i, v_j \in V)$ between those nodes. Under some node ordering π this graph is represented with its adjacency matrix $A^\pi \in \{0, 1\}^{n \times n}$ with $A^\pi_{i,j} = 1$ iff $(\pi(v_i), \pi(v_j)) \in E$. The model generates graphs as sequences of adjacency vectors $S^\pi_i \in \{0, 1\}^{i-1}$ from node $\pi(v_i)$ to previous nodes under π. Thus, $S^\pi_i =$

$\left(A^{\pi}_{1,i}, \dots A^{\pi}_{i-1,i}\right)^T$, and likelihood $p(S^{\pi})$ can be modelled sequentially, being decomposed as

$$p(S^{\pi}) = \prod_{i=1}^{n+1} p(S^{\pi}_i | S^{\pi}_1, \dots, S^{\pi}_{i-1}) = \prod_{i=1}^{n+1} p(S^{\pi}_i | S^{\pi}_{<i}) \tag{2.1}$$

with the special end of sequence token (EOS) as an extra node $n+1$.

State-transition function carries the information from step $i-1$ to step i, generating a node, and output function predicts the parameters for sampling current adjacency vector S^{π}_i of edges. According to GraphRNN, we consider recurrent neural networks for both state-transition function (*NodeRNN*) and output function (*EdgeRNN*). Thus, NodeRNN unrolls across nodes, updating its hidden state, while EdgeRNN unrolls across edges from i to previous nodes, creating parameters $\theta_{i,j}$ with the use of a small MLP head with sigmoid activation, which models S^{π}_i as a dependent Bernoulli sequence:

$$\begin{aligned} h^{node}_i &= \text{NodeRNN}(h^{node}_{i-1}, S^{\pi}_{i-1}), \quad h^{node}_0 = \mathbf{0} \\ h^{edge}_{i,j} &= \text{EdgeRNN}(h^{edge}_{i,j-1}, S^{\pi}_{i,j-1}), \quad h^{edge}_{i,0} = h^{node}_i \\ \theta_{i,j} &= \text{EdgeMLP}(h^{edge}_{i,j}) \end{aligned} \tag{2.2}$$

One of the key insights of the method is to re-order the nodes with breadth-first search (BFS), starting from $\pi(v_1)$, which gradually reduces the space complexity for graph representations. Moreover, BFS order also reduces the number of edge predictions that have to be made, limiting the size of S^{π}_i to M dimensions, which appears to be a small number in practical tasks. Thus, for our modified MolecularRNN (Section 2.3.1.2) we empirically establish $M = 12$.

2.3.1.2 MolecularRNN

In order to represent a molecule with a graph, atoms are mapped to nodes, while bonds are mapped to edges. Now, adjacency vector entries represent categorical bond types $S^{\pi}_{i,j} \in \{0, 1, 2, 3\}$, corresponding to no, single, double, and triple bonds (molecules are modeled in kekulized form as defined in RDKit [56]). Similarly, a categorical type $C^{\pi}_i \in \{1, 2, \dots, K\}$ (oxygen, nitrogen, chlorine, etc.) is assigned to each node. Notice that here a node always has a valid atom class. That is, there is no "terminal node" class, as terminal node notion is already incorporated into S^{π}_i.

Specifically, when a node is generated that has no edges to any of the previous nodes, such a node is terminal. Atom class prediction is ignored for this node in our setting.

The Likelihood in Equation 2.1 is rewritten accordingly for MolecularRNN:

$$p(S^{\pi}, C^{\pi}) = \prod_{i=1}^{n+1} p(C_i^{\pi} | S_{<i}^{\pi}, C_{<i}^{\pi}) p(S_i^{\pi} | C_i^{\pi}, S_{<i}^{\pi}, C_{<i}^{\pi}), \tag{2.3}$$

with $p(C_{n+1}|S_{<n+1}, C_{<n+1}) \equiv 1$ for the terminal node $n+1$.

In our model, once the sub-graph on the first $i-1$ nodes under permutation π is completed, NodeRNN can momentarily decide on the atom type of the following node i. Thus, the process represents a dependent multivariate distribution. Accounting for the sub-graph, as well as the i-th atom type, the model switches to EdgeRNN that links the newly generated node to the set $\{1, \ldots, i-1\}$. That step is in turn modeled with a dependent multivariate distribution, as EdgeRNN is unrolled across nodes that precede i. Overall MolecularRNN structure is shown in Figure 2.8. The model uses embeddings for categorical inputs, and a two-layer MLP with softmax output activation is added on top of hidden states h^{node} and h^{edge} for categorical prediction, so Equation 2.2 is modified:

$$\begin{aligned}
&\text{input}_{i-1} = [emb(S_{i-1}^{\pi}), emb(C_{i-1}^{\pi})] \\
&h_i^{node} = \text{NodeRNN}(h_{i-1}^{node}, \text{input}_{i-1}), \qquad h_0^{node} = \mathbf{0} \\
&\psi_i = \text{NodeMLP}(h_i^{node}) \\
&h_{i,j}^{edge} = \text{EdgeRNN}(h_{i,j-1}^{edge}, emb(S_{i,j-1}^{\pi})), \quad h_{i,0}^{edge} = h_i^{node} \\
&\phi_{i,j} = \text{EdgeMLP}(h_{i,j}^{edge}),
\end{aligned} \tag{2.4}$$

In our BFS ordering the first node is always a Carbon atom, since every organic molecule contains at least one such atom.

2.3.1.3 Valency-based rejection sampling

As we have seen, MolecularRNN samples edge types on each sub-step from a multinomial distribution with parameters coming out of softmax predictions. Even when the model is trained well for producing valid molecules, the softmax layer prediction will always have nonzero values,

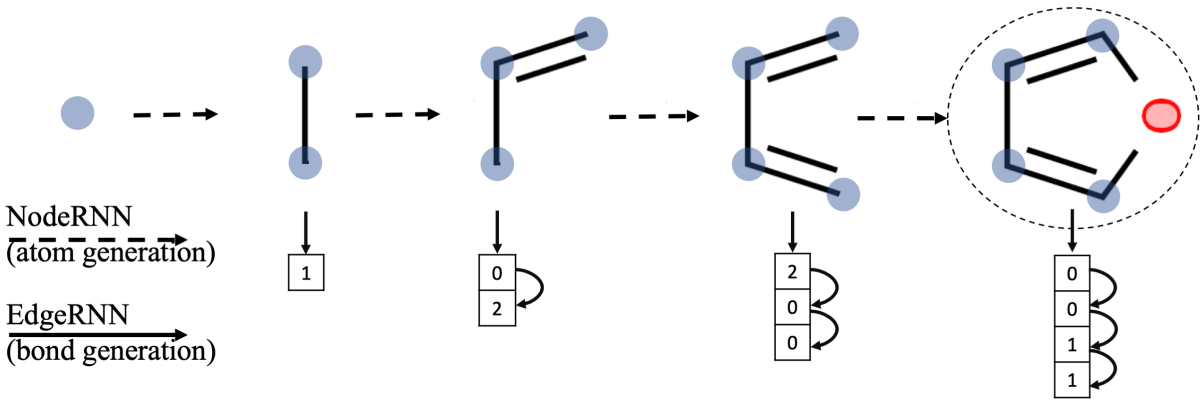


Figure 2.8: MolecularRNN model. The model consists of NodeRNN that unrolls across atoms, predicting the type of the next atom in the molecular graph, and EdgeRNN that for every atom is initialized with NodeRNN hidden state, and unrolls across preceding atoms to predict bond types.

so if sampling is arbitrarily long, any graph can be sampled from the support space. However, real molecules have valency constraints. That is, per-atom valency has to be respected to satisfy chemical constraints. Consequently, in each step, we can ensure that the current sum of all bonds does not exceed the allowed valency. When generating an edge corresponding to a bond of order k between i and j we check the rejection sampling constraint for both atoms:

$$\sum_j A^{\pi}_{i,j} + k \leq \text{valency}_{C^{\pi}_i} \quad \text{and} \quad \sum_i A^{\pi}_{i,j} + k \leq \text{valency}_{C^{\pi}_j} \tag{2.5}$$

For the final molecule, atoms that have not filled up their valencies are complemented with Hydrogens. Notice that valency can be directly enforced only for graphs, unlike SMILES representation, where intermediate sub-strings are not chemically meaningful.

2.3.1.4 Structural penalty

Valency-based rejection sampling can be used in inference, as was already described. However, the invalid intermediate structures that are obtained during training can provide a useful signal to the model. For example, a molecule can be *almost realistic* except for few invalid bonds. We introduce an additional structure penalty for the atoms that disrespect valencies. Thus, instead of providing a penalty for the whole molecule, we target specific atoms, which results in the modification of parameters that respect valency constraints.

2.3.2 Results

To validate the quality of our results and compare those to the state-of-the-art methods we use validity, uniqueness, novelty, internal diversity, synthetic accessibility score (SAS) [54] and drug-likeness score (QED) [57]. Validity is the percentage of chemically valid molecules. While it is important that the model demonstrates a high validity rate, we do not think that the validity of 100% is essential. Moreover, without external constraints (that might be an integrated part of the model) it is not possible to reach perfect validity for a probabilistic generative model due to sampling. Uniqueness is the percentage of unique molecules in the generated pool. Notice that uniqueness is highly dependent on the pool size and may significantly drop for a large generated library. In our experiments, we report uniqueness in up to a million of generated samples. Internal diversity, as proposed in the MOSES benchmark [58], is a quantitative metric of the richness of the generated library and is calculated as the average pairwise distance between all pairs of molecules in the library. However, we do not agree that this is a good measure of dataset diversity. Imagine a synthetic dataset that consists of 1 million molecules, but only 20 of them are unique and each molecule is repeated 50 thousand times. It is obvious that such a dataset is very poor as only negligible part of the whole library is unique and an adequate diversity metrics should reflect this fact. However, the value of this commonly used metric will be around 0.85. The highest possible value for this metrics is 1.0 and the internal diversity of a rich dataset such as ChEMBL is around 0.895. So, this metric might not be a good indicator of the diversity of the generated pool of molecules.

SAS is an estimation of how hard is to synthesize a given molecule, which also reflects its structural complexity. Molecules with a higher score will be more complex and harder to synthesize. However, molecules with very low score might be not complex enough to have the desired property. The values of interest for this metric are in the range between 2 and 4. Finally, QED is a measure of drug-likeness in the range from 0 to 1.

2.3.2.1 Unsupervised likelihood training

We first pretrain MolecularRNN on a large unlabeled dataset of molecules to teach the model to generate diverse realistic samples. Three training datasets are used: ChEMBL [51], random

250k molecules from ZINC [59] and MOSES [58]. These three datasets have different statistics. The statistics are shown in Table 2.1. The ChEMBL dataset contains around 1.5 million of real bioactive molecules (every molecule has at least one experimental bioactivity measurement) and is the most diverse out of all three datasets that we considered. The ZINC 250k random dataset contains 250 thousand molecules randomly selected from a database of commercially available compounds [59]. The MOSES dataset contains almost 2 million molecules that were selected from the ZINC database based on several filters to only include molecules with drug-like properties.

Table 2.1: Statistics for training datasets.

	ChEMBL	**ZINC**	**MOSES**
Number of molecules	1507869	249456	1936962
Mean molecular weight	389 ± 103	331 ± 62	307 ± 28
QED score	0.56 ± 0.21	0.73 ± 0.14	0.81 ± 0.09
SA score	2.88 ± 0.80	3.05 ± 0.83	2.45 ± 0.46

We considered the 9 most common elements ($C, N, O, F, P, S, Cl, Br, I$) and 3 bond types (single, double and triple). The number of atoms in the molecule is restricted to be from 10 to 50, which is chosen based on thef ChEMBL dataset, where 96% of molecules lie in this range. EdgeRNN is unrolled (as discussed in subsection 2.3.1.1) for $M = 12$ steps for each atom. The following architectural parameters are used in all our settings: node embedding of size 128, edge embedding of size 16, NodeRNN with 4 GRU layers of hidden size 256 each, 2 layer NodeMLP with 128 hidden size and ReLU nonlinearity after the first layer, and EdgeRNN with 4 GRU layers of hidden size 128 each. During the unsupervised phase, models are trained with the Adam optimizer for 250 epochs on 4 GPUs with a per-GPU batch size of 512. The starting learning rate is 0.001 with a multiplicative drop of 0.999 every k iterations, and k is chosen based on the dataset so that the learning rate drops to 10^{-5} by the end of the training. MolecularRNN trained with the likelihood maximization on the training datasets achieves validity rate of 65% without valency-based rejection sampling. We further used the structural penalty described in subsection 3.1.2.3 to shift the model towards generating molecules that respect valency constraints. To that end, every atom that violates its valency constraints is assigned a penalty of -10, and then the model is optimized with the policy gradient method. After training with the structural penalty,

our model achieved a validity rate of 90% without valency-based rejection sampling. Enabling valency-based rejection sampling results in 100% valid rate for all models.

Table 2.2: Statistics for 1 million molecules generated by 3 models pretrained on 3 training datasets

Training set	**Valid**	**Unique**	**Novel**	**IntDiv (p=1)**	**IntDiv (p=2)**	**SA score**	**QED**
ChEMBL	100 %	99.2%	99.3 %	0.895	0.890	3.67 ± 1.20	0.56 ± 0.20
ZINC 250k	100 %	99.8 %	100 %	0.892	0.887	3.60 ± 1.01	0.68 ± 0.16
MOSES	100 %	99.4 %	100 %	0.881	0.876	3.24 ± 0.97	0.74 ± 0.14

Table 2.2 summarizes the results of unsupervised likelihood training of MolecularRNN on the three datasets. Statistics are calculated on 1 million generated graphs, which is a much larger scale than previously reported. For comparison, Jin et al. [2] sample 5 thousand graphs, and Li et al. [35] evaluate a 100 thousand set. In all cases, the model produces novel diverse realistic molecules.

We also compare our model with GCPN [1] and JT-VAE [2] in Table 2.3 on 30K molecules generated from each method. MolecularRNN produces comparable results to the baselines in terms of validity, uniqueness, and novelty. GCPN tends to generate overly complex, hard to synthesize molecules (high SA score). Samples from our model are more realistic, and also have higher internal diversity than the ones from JT-VAE.

Table 2.3: Comparison of MolecularRNN, GCPN [1] and JT-VAE [2]. Models are trained on ZINC 250k dataset. Statistics are calculated for 30000 generated molecules.

	Valid	**Unique**	**Novel**	**SA score**	**QED**	**InvDiv**
JT-VAE [2]	99.8%	100 %	100%	3.37	0.76	0.85
GCPN [1]	100%	99.97%	100%	4.62	0.61	0.90
MolecularRNN	100%	99.89%	100%	3.59	0.68	0.89

2.4 3D graph-based generative models

2.4.1 Methods

MolecularRNN 3D model description. MolecularRNN3D, similarly to MolecularRNN described in Section 2.3, is an auto-regressive generative model, meaning that molecules are generated atom-by-atom, connecting the newly generated atom to the previously generated ones and placing atoms in 3D space. Starting from a single carbon atom, at every time step, the model first predicts the next atom type given the previously generated 3D graph. Then, the model decides on the set of adjacent atoms and the bond types. Lastly, it also predicts bond lengths, valence and dihedral angles for the sets of 4 connected atoms forming a dihedral angle. As additional features for predicting atom's 3D coordinates, we use Atomic Environment Vectors (AEVs) [16] for 3 out of 4 atoms in the dihedral angle. In more detail, since the molecule is generated in an auto-regressive manner, i.e., atom-by-atom, at any specific time step, we only have information about part of the molecule that has been generated so far. Thus, to predict the next dihedral angle, we can only calculate AEVs for three prior atoms in the dihedral angle since the 4th atom has not been generated and added to the molecule yet. These AEVs are concatenated with the 2D representation of a molecular graph computed by the neural network and the embedding vector for the 4th atom type. The generative process is over when the model predicts that the next atom is not connected to any previously generated atoms. Figure 2.9 illustrates the generative process of new 3D molecular graphs. Unlike models previously proposed in the literature, MolecularRNN3D does not have any restrictions on molecule size and tackles both problems of novel molecules generation and prediction of 3D conformations for existing molecules.

Notation. Before moving into describing the MolecularRNN3D model, we want to introduce necessary mathematical notation.

Difference between 3D and 2D version of MolecularRNN model is that 3D version contains 3 additional feed-forward layers for predicting bond lengths, valence angles and dihedral angles. To address the problem of non-unique representation of graphs, similarly to MolecularRNN model described in 2.3, this model sorts the nodes in the breadth-first search order, thus introducing an

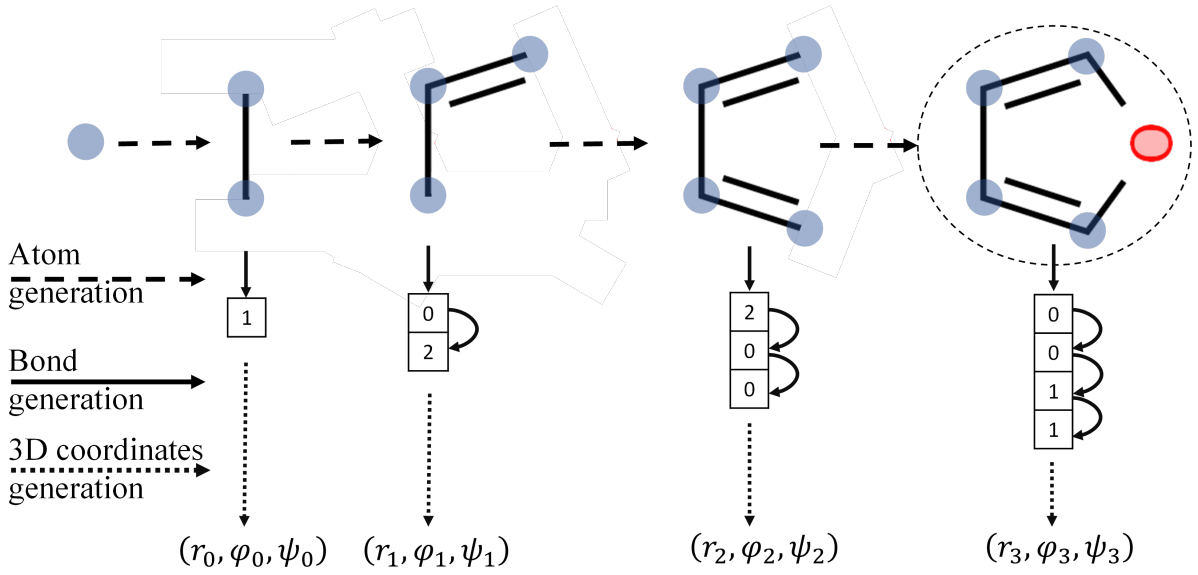


Figure 2.9: Schema of MolecularRNN 3D model.

ordering of the atom. So, given a molecule M with N atoms, the permutation of a molecule's atoms can be denoted as:

$$\pi(M) = \{a_1, a_2, \ldots, a_N\}.$$

The adjacency matrix of molecule M is a matrix A with size $N \times N$, where element A_{ij} defines the existence of a bond between atoms a_i and a_j and the type of this bond if it exists:

$$A_{ij} = \begin{cases} 0, \text{if no bond between atoms } a_i \text{ and } a_j, \\ 1, \text{if single bond between atoms } a_i \text{ and } a_j, \\ 2, \text{if double bond between atoms } a_i \text{ and } a_j, \\ 3, \text{if triple bond between atoms } a_i \text{ and } a_j. \end{cases}$$

Since the ordering of the atoms in the molecule $\pi(M)$ is defined by the breadth-first search algorithm, for each atom a_n starting from $n = 4$, we can find 3 atoms a_i, a_j and a_k preceding a_n in the permutation $\pi(M)$ and forming a dihedral angle $\angle(a_i, a_j, a_k, a_n)$:

$$\forall n \geq 4 \; \exists \, i, j, k : n > i, n > j, n > k, \;\; A_{ij} \neq 0, A_{jk} \neq 0, A_{kn} \neq 0.$$

We will denote the set of atom quadruplets forming dihedral angles in a molecule M as D:

$$D = \{(a_i, a_j, a_k, a_n) : \exists \angle(a_i, a_j, a_k, a_n),\ n > i, j, k,\ n = 4 \dots N\}$$

Notably, it is possible that there will be multiple quadruplets forming a dihedral angle for atom a_n. In this case, we only keep 1 quadruplet per each a_n, denoting this new "de-duplicated" set with exactly $N - 3$ elements as $\widetilde{D}$:

$$|\widetilde{D}| = N - 3$$

The set Ψ represents values of dihedrals angles from $\widetilde{D}$:

$$\Psi = \{\psi_4, \psi_5, \dots, \psi_N\},$$

where $\psi_n = \angle(a_i, a_j, a_k, a_n),\ n = 4, \dots, N$. Numbering of the elements in this set start from 4, since at least 4 atoms in a molecule are necessary to form a dihedral angle.

The set Φ is defines values of valence angle for atom quadruplets from $\widetilde{D}$ as follows:

$$\Phi = \{\phi_3, \dots, \phi_{N-2}\},$$
$$\phi_3 = \angle(a_i, a_j, a_k), a_i, a_j a_k \in (a_i, a_j, a_k, a_4),$$
$$\phi_n = \angle(a_j, a_k, a_n), n = 4, \dots N.$$

Similarly, the set R defines the bonds' lengths:

$$R = \{r_2, \dots, r_{N-1}\},$$
$$r_2 = dist(a_i, a_j),\ r_3 = dist(a_j, a_k),\ a_i, a_j a_k \in (a_i, a_j, a_k, a_4)$$
$$r_n = dist(a_k, a_{n+3}), n = 1, \dots N - 3.$$

In the definitions above, ϕ_3 is a special case of a valence angle, and r_2 and r_3 are special cases of bond lengths formed by atoms in the first dihedral angle from $\widetilde{D}$. The numbering of elements in Φ and R starts from 3 and 2 respectively, because at least 3 atoms are necessary to form a valence angle, and at least 2 atoms are necessary to form a bond.

3D coordinates prediction. After all the necessary notations have been introduced in the previous paragraph, we can move to describing the process of 3D coordinates prediction.

Since the generation process is auto-regressive, let us assume that $n-1$ atoms have already been added to the molecule. During time step n after the next atom's type a_n and its connectivity is generated by the 2D part of MolecularRNN described in section 2.3, MolecularRNN3D predicts triplets (r_n, ϕ_n, ψ_n). For this purpose, the model computes the atomic environment vectors (AEVs) for atoms a_i, a_j and a_k and concatenates them with the hidden vectors h_n and h_N from the 2D part of the model. We will denote the resulting vector as $h_{3D}^{(n)}$:

$$h_{3D}^{(n)} = [AEV(a_i), AEV(a_j), AEV(a_k), h_n, h_N].$$

These new hidden vectors $h_{3D}^{(n)}$ describe both the local 3D environment and the 2D topology around the dihedral angle of interest.

Next, $h_{3D}^{(n)}$ is passed into 3 separate feed-forward layers, which predict a triplet (r_n, ϕ_n, ψ_n). In more detail, bond length prediction is handled as a regression task – the feed forward layers predict normalized value of the bond length between atoms a_k and a_n, which after de-normalization will be used to compute the coordinates of the atom a_n in 3D space. Valence angle prediction and dihedral angle prediction are handled as classification tasks – the angles are discretized with a step of $3°$, which results into 60 classes for valence angles and 120 classes for dihedrals angles. The predicted classes are converted into values of angles in degrees, which together with the predicted bond length are used to compute (x, y, z) coordinates of atom a_n.

Computing coordinates for atoms a_2 and a_3 are special cases, due to the fact that dihedral angle for a_2 and a_3 and valence angle for a_2 are not defined. So, after the atom a_2 and its connectivity is generated, MolecularRNN3D predicts bond length between atoms a_1 and a_2, denoted as r_2. For this purpose, the model computes the atomic environment vector for atom a_1. Since no previous atoms were defined, we assume that the atomic environment vector for 2 previous atoms are zero vectors and the h_{3D}^{1} is defined as:

$$h_{3D}^{(1)} = [AEV(a_1), \overrightarrow{\mathbf{0}}, \overrightarrow{\mathbf{0}}, h_1, h_N].$$

Similarly, during the second time step only bond length and valence angle will be predicted and the $h_{3D}^{(2)}$ is defined as:

$$h_{3D}^{(1)} = [AEV(a_2), AEV(a_1), \overrightarrow{\mathbf{0}}, h_2, h_N].$$

2.4.2 Results

Data collection. Obtaining a training dataset with high-quality low-energy conformation is essential for training a reliable conformer generator model. For this purposes, we used $1.5M$ molecules from ChEMBL dataset [60] which contains from 9 to 56 heavy atoms. Values of 9 and 56 correspond to a 1st and 99th percentiles and ensure that no outliers are included into the training data. First, we generated up to 5 initial 3D conformations per molecule with OEOmega (https://www.eyesopen.com/omega). Next, we performed the optimization of this initial conformations with the AIMNet model [61]. These computations resulted in $9M$ conformations. For the final dataset we only picked one conformation per molecule – the one with the lowest energy.

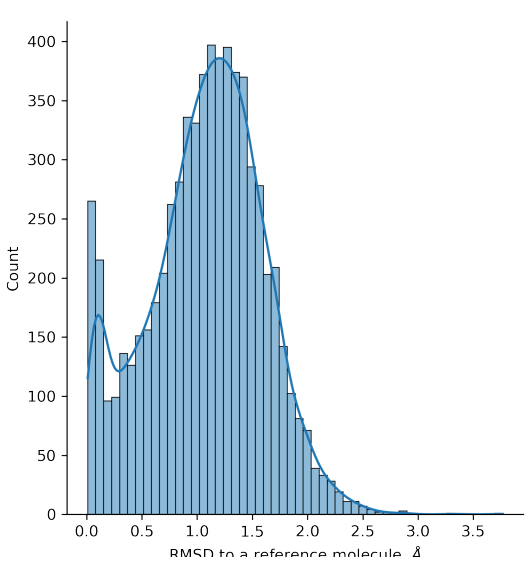
The data was randomly split into train and test – $100K$ molecules went into test and the rest were assigned to train.

Model training. Model training was split into 2 stages. During the first stage we pretrained the 2D part of the MolecularRNN to teach the model to generate realistic molecular graphs. During the second stage, we froze the model weights in the 2D part and only trained 3D part which include 3 layers for predicting bond lengths, valence angles and dihedral angles. The technical details of training the 2D part are described in Section 2.3.

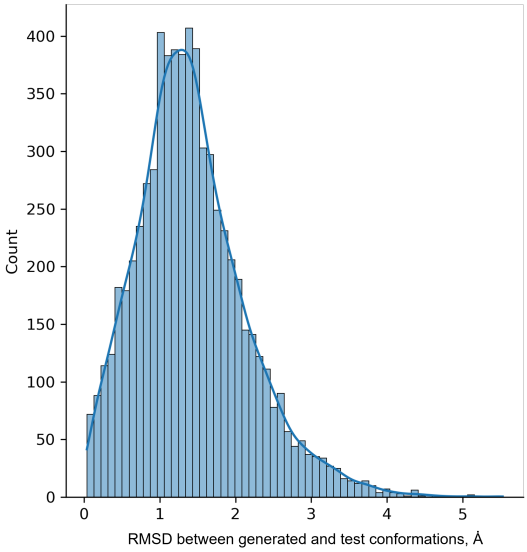
For the 3D coordinates prediction, each layer consisted of 4 feed-forward layers with ReLU activations and 512, 256 and 128 neurons in the first 3 layers and the number of classes in the last layer (1 for bond lenghts, 60 for valence angles and 120 for dihedral angles). The model was trained for 200 epochs with Adam optimizer starting from the learning rate of 0.002 and exponentially decaying it after each training epoch with a decay factor of 0.95.

Assesing quality of the generated conformers. We assessed the quality of the generated conformers in 2 modes. In the first mode, we generated both the 2D molecular graph and the 3D conformation from scratch. The reference conformation was then computed for the 2D molecule using RDKit's force filed algorithm MMFF94. Figure 2.10a shows the distributions of RMSD between the generated conformations and the reference conformation. The median of the RSMD is equal to $1.1\mathring{A}$ and the vast majority of the conformers had $RMSD < 2.0$.

In the second mode, we compared how MolecularRNN3D can generate conformers for existing molecular graphs. For this purposes, we used molecules from our test subset and compared the generated conformation with the well-optimized conformations from the test set. The distribution of RMSD is shown in Figure 2.10b. The median of RMSD is equal to $1.3\mathring{A}$ and most conformers had $RMSD < 2.0$.



(a) RMSD between generated and FF-optimized conformations.



(b) RMSD between generated and test conformations.

Figure 2.10: Distributions of RMSDs.

2.5 Empirical estimation of size of bioactive chemical space with generative models

2.5.1 Introduction

Although multiple theoretical estimates for the size of chemical space (CS) have been provided [23, 62], there is still no consensus in estimating the number of realistic bioactive molecules: the results vary from 10^{23} to 10^{180}. Furthermore, these estimates are primarily combinatorial and do not consider how realistic the molecules are. For example, according to Bohacek et al. [62], the number of compounds consisting of thirty C, N, O, S atoms and having up to 4 cycles and ten branch points is about 10^{60}. On the other hand, Polischuk et al. [23] provide the number 10^{33}, which was obtained by calculating how many molecular graphs satisfy the classical Lipinski's "rule of five" [63].

In this section, we address the question of the size of bioactive chemical space. We propose a procedure for empirically estimating the number of bioactive molecules with a deep generative model. We use a SMILES-based recurrent generative network trained on the ChEMBL database to produce novel molecules with bioactive properties. Using this approach, we generated AI-COSMOS – AI-designed COllection of Small MOleculeS, which consists of 10 billion unique compounds and 4.5 billion unique Bemis-Murcko scaffolds in the form of canonical SMILES strings. We thoroughly assess the properties and synthetic accessibility of the molecules from the AI-COSMOS library. We demonstrate that the statistics of this library match the statistics of ChEMBL, and molecules are novel and diverse. Based on the AI-COSMOS library, we empirically predict lower bounds for the number of unique canonical SMILES strings and Bemis-Murcko scaffolds that can potentially be bioactive. Additionally, we perform an experimental study to demonstrate how the AI-COSMOS library can be used to find potential hits for protein targets.

2.5.2 Results

Library generation. Figure 2.11 illustrates the overall workflow of the AI-COSMOS 10 billion library generation process. First, we trained the deep generative recurrent neural network

architecture introduced in 2.2.1 on SMILES strings from the ChEMBL [60] database. To train the model efficiently, we removed 1% of the shortest and 1% of the longest SMILES strings from the training pool, giving us a range of SMILES strings lengths between 10 and 100 characters. Next, we removed molecules that contained atoms from other than ten most common atom types (H, C, N, O, F, P, S, Cl, Br, I). As a result, the trained model can mainly produce valid SMILES strings, with a validity rate of 90%. Next, we ran this model in inference mode to generate a vast library of molecules. In more detail, the generation process was run on a cluster with NVIDIA Volta V100 16Gb GPUs. We used a batch size of 56, meaning that 56 SMILES strings were generated in parallel at a time. The generation speed was about 1 million SMILES strings per hour. Overall, we generated 17 billion SMILES strings, which took approximately 17000 GPU hours. Postprocessing the initial library included two steps – removal of invalid SMILES strings, SMILES strings standardization, and deduplication. The RDKit [56] sanitization filters defined SMILES strings validity and canonical form. Additionally, we calculated Bemis-Murcko scaffolds for all valid generated molecules. Removal of invalid SMILES resulted in 15 billion valid SMILES strings corresponding to 6 billion Bemis-Murcko scaffolds. After deduplication, we obtained 10 billion unique, valid SMILES strings and 4 billion unique Bemis-Murcko scaffolds.

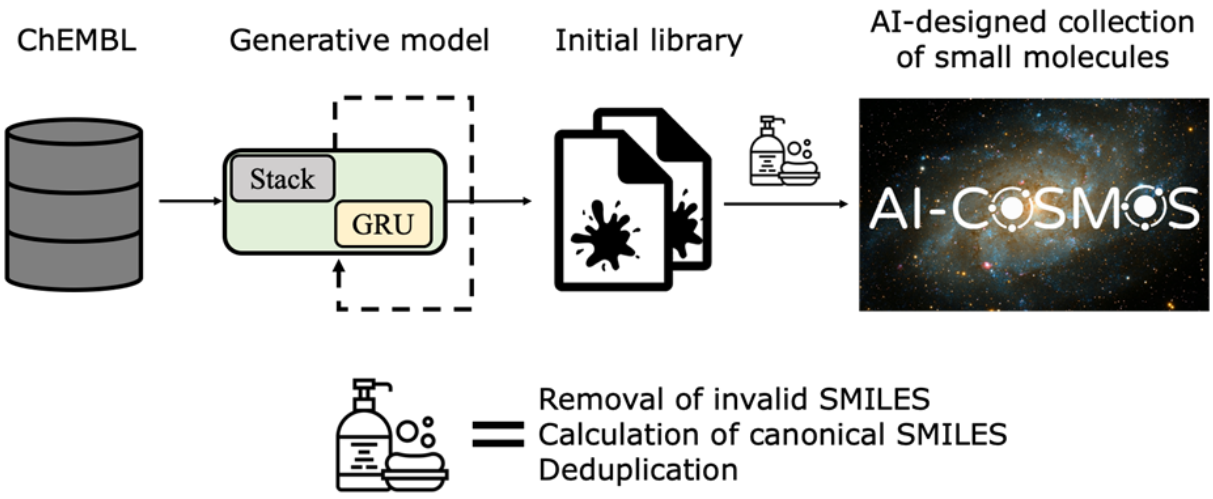


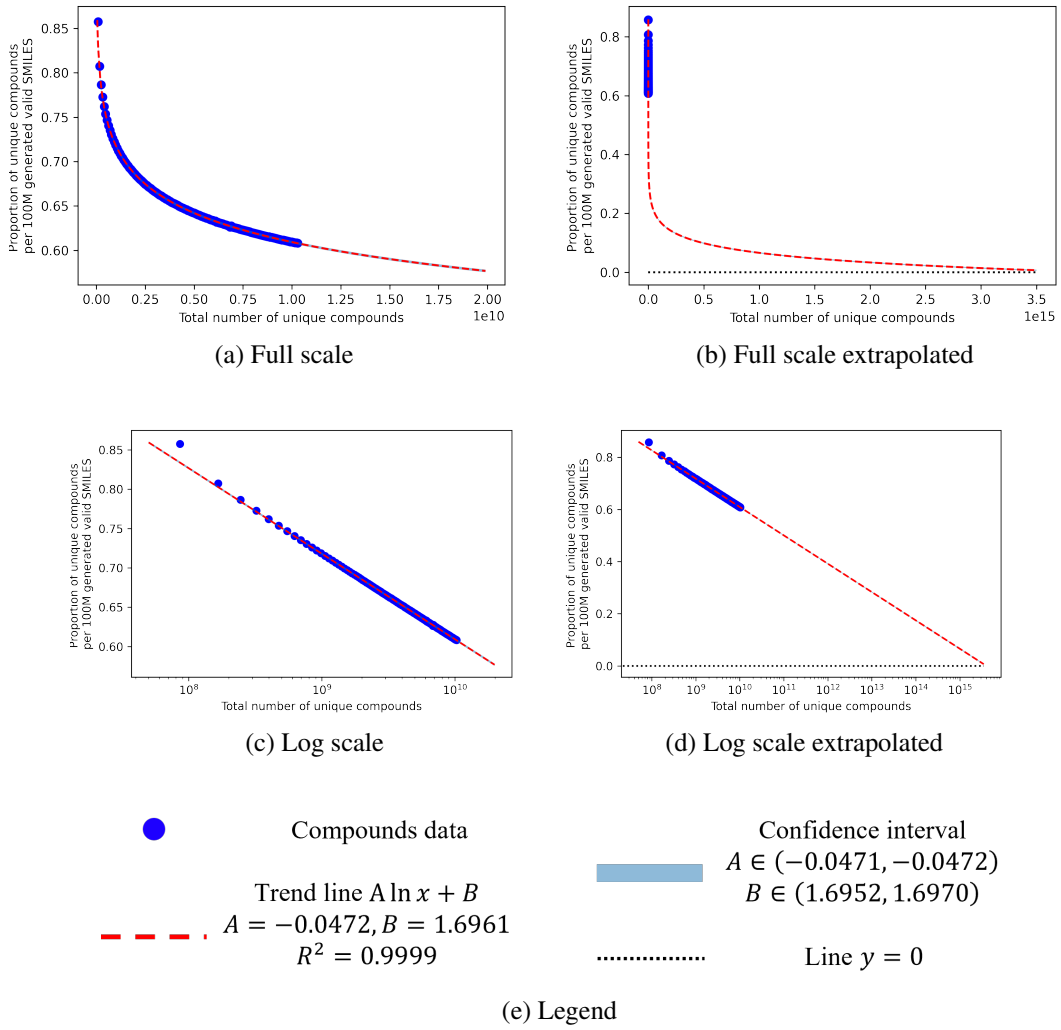
Figure 2.11: The workflow of the library generation process.

Chemical space size estimation. We calculated how the proportions of unique canonical SMILES strings and unique Bemis-Murcko scaffolds change with library size growth during the deduplication step. After removing invalid entries and standardization, we split the library into 100 million canonical SMILES strings blocks. Each block could contain duplicates. Next, we computed the cumulative curves, showing the proportion of unique molecules in each 100 million block given the previously generated sublibrary. Then, we approximated the data with a logarithmic trendline $y = A\ln(x) + B$ using the least-squares method, where x is a number of unique canonical SMILES strings that have been generated so far, and y is the proportion of unique compounds or Bemis-Murcko scaffolds in a newly generated block of 100 million canonical SMILES strings. To justify that the logarithmic function is an adequate approximation functional form for the data, we calculated Pearson correlation coefficients (PCC) for $(y, \ln(x))$ pairs for compounds and Bemis-Murcko scaffolds, and p-values of observing such results under the null hypothesis that y and $\ln(x)$ are not linearly correlated. We obtained $PCC < -0.99$ and $p\text{-}value < 10^{-162}$ for both compounds and Bemis-Murcko scaffolds, which indicates practically perfect linear correlations. These correlations are shown in Figure 2.12a and 2.12c for compounds and Figure 2.13a and 2.13c for scaffolds. Additionally, we computed 5% confidence intervals for the fitted trendlines. Both trendlines for SMILES strings and Bemis-Murcko scaffolds were computed with a coefficient of determination of $R^2 > 0.99$, which again demonstrates that logarithmic functional form is a good choice for this problem. Finally, to estimate the total number of unique canonical SMILES strings that our generative network can produce, we solved the following equation:

$$Aln(x) + B = 0. \tag{2.6}$$

The solution of this equation provides the total number of valid unique canonical SMILES strings that need to be produced before a newly generated block of 100 million SMILES strings would contain 0 unique compounds. We repeated these computations for both canonical SMILES strings and Bemis-Murcko scaffolds and obtained the following estimates:

$$\text{Number of unique canonical SMILES strings} = 4.0 \cdot 10^{15} \pm 0.2 \cdot 10^{15},$$



(a) Full scale

(b) Full scale extrapolated

(c) Log scale

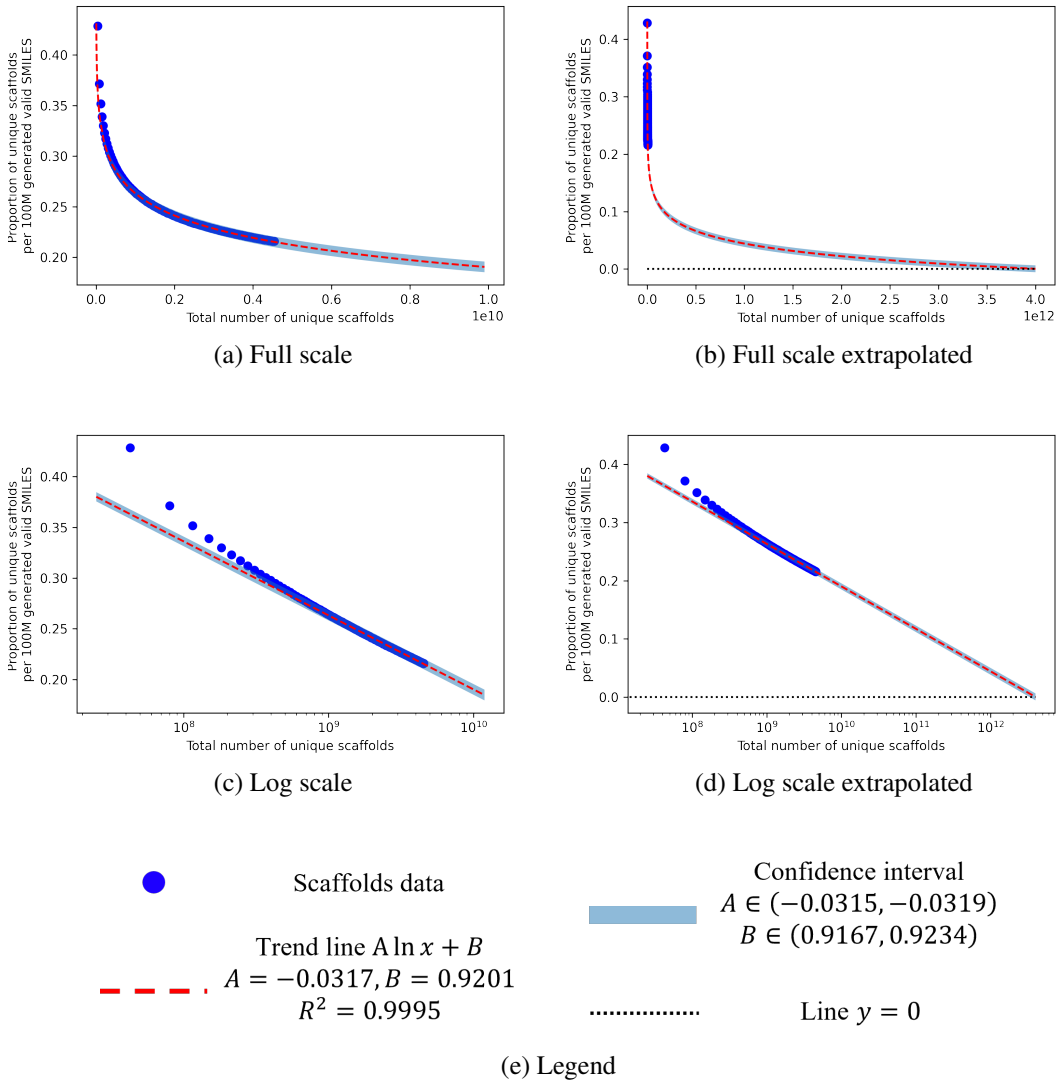
(d) Log scale extrapolated

(e) Legend

Figure 2.12: Extrapolation of the trend for chemical compounds.

$$\text{Number of unique Bemis-Murcko scaffolds} = 4.1 \cdot 10^{12} \pm 0.9 \cdot 10^{12}.$$

Figures 2.12 and 2.13 summarize the steps described above. Moreover, using simple mathematical derivations and L'Hopital's rule, one can demonstrate that the logarithmic function provides a lower estimate than some other adequate monotonic functions that can approximate this data, such as power, exponential, or loglog. Thus, we conclude that the estimate we provide above can serve as a lower bound estimate for the number of biologically active compounds and corresponding Bemis-Murcko scaffolds.



(a) Full scale

(b) Full scale extrapolated

(c) Log scale

(d) Log scale extrapolated

(e) Legend

Figure 2.13: Extrapolation of the trend for chemical compounds.

AI-COSMOS diversity set. Due to the large size of AI-COSMOS library, we constructed the AI-COSMOS diversity set to use in further analysis. This diversity set was constructed during the deduplication process by randomly sampling 10000 canonical SMILES strings from each 100 million block of SMILES and removing duplicates afterwards. The final selection includes 1.3 million unique canonical SMILES strings.

Analysis of the AI-COSMOS library. Next, we took a closer look at the AI-COSMOS library by computing a set of typical properties of interest and comparing distributions of those properties

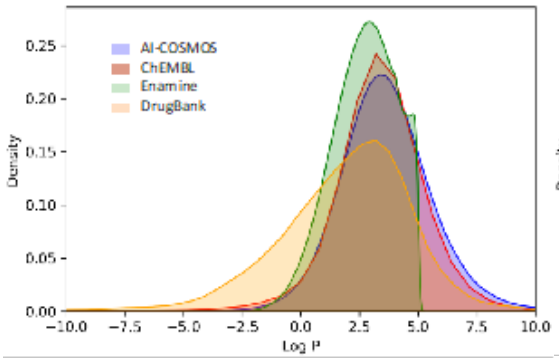
to existing molecular libraries. The comparison included AI-COSMOS diversity set, full ChEMBL library [60], Enamine Diversity Set [64], and Drugbank [65]. For the properties, we chose Quantitative Estimate of Druglikeness (QED) [57], Crippen's Log P [66], and properties from the Lipinski's rule-of-five [63], which constitute $\%$ of molecules with Log P values within $[0, 5]$ region, molecular weight, number of hydrogen bond acceptors and number of hydrogen bond donors. As expected, distributions for the chosen properties look similar for generated molecules and ChEMBL since ChEMBL was used as training data. However, distributions for Enamine Diversity set look slightly different because molecules in the set were manually selected to satisfy Lipinski's rule-of-five ($M_w < 500Da$, $logP < 5$, $N_{HBD} < 5$, $N_{HBA} < 10$). Molecules from DrugBank also have a wider range of values for the chosen properties. These observations are presented in Table 2.4 and Figure 2.14.

Table 2.4: Comparison of property values for AI-COSMOS, ChEMBL, Enamine, and DrugBank libraries.

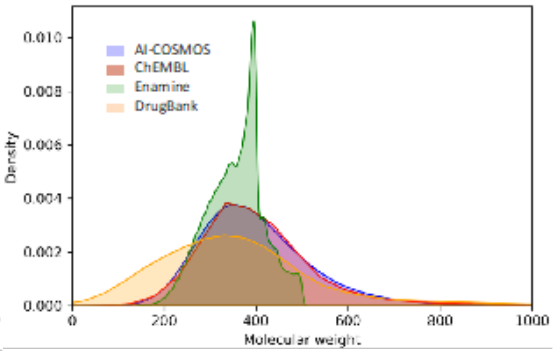
	QED	**Log P**	**Log P within [0, 5]**	**Molecular weight**	**Hydrogen bond acceptors**	**Hydrogen bond donors**
AI-COSMOS	0.55 ± 0.23	3.65 ± 2.02	75%	397 ± 117	4.87 ± 2.10	1.58 ± 1.41
ChEMBL	0.55 ± 0.23	3.32 ± 2.24	78%	422 ± 244	5.54 ± 4.47	1.90 ± 3.37
Enamine	0.74 ± 0.11	2.69 ± 1.36	97%	356 ± 62	4.79 ± 1.66	1.07 ± 0.88
DrugBank	0.50 ± 0.23	1.85 ± 3.32	67%	381 ± 304	5.60 ± 6.30	2.69 ± 0.89

The next question we addressed is how similar (or dissimilar) are the generated molecules to training molecules from ChEMBL. To answer this question, we calculated pairwise similarity between the AI-COSMOS diversity set and the entire ChEMBL library. For each molecule in the AI-COSMOS diversity set, we computed its median Tanimoto similarity to all molecules from ChEMBL and Tanimoto similarity to the nearest neighbor from ChEMBL. These distributions are shown in Figures 2.15a and 2.15b. The median value of Tanimoto similarity (T_{sim}) to the nearest neighbor from ChEMBL is 0.62, indicating that most molecules from the AI-COSMOS library are not exact replicates of molecules from ChEMBL.

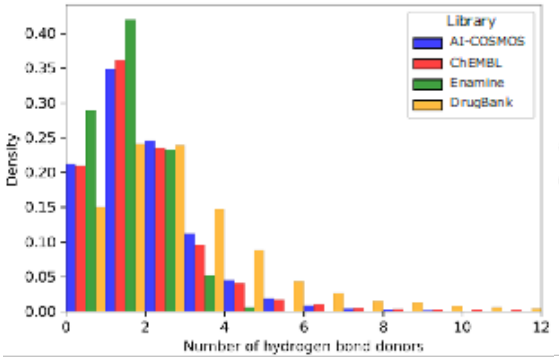
Additionally, we looked at the examples of molecules with low similarities to the nearest neighbor from ChEMBL. We took 1% of molecules with the lowest Tanimoto similarities to



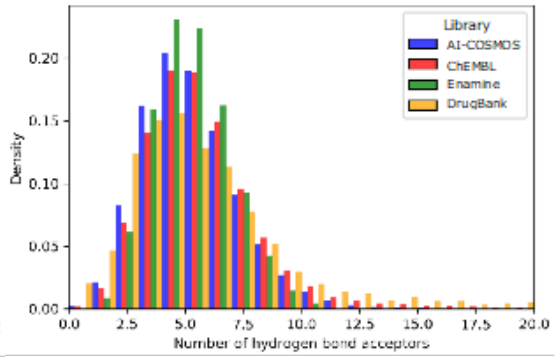
(a) Mol Log P



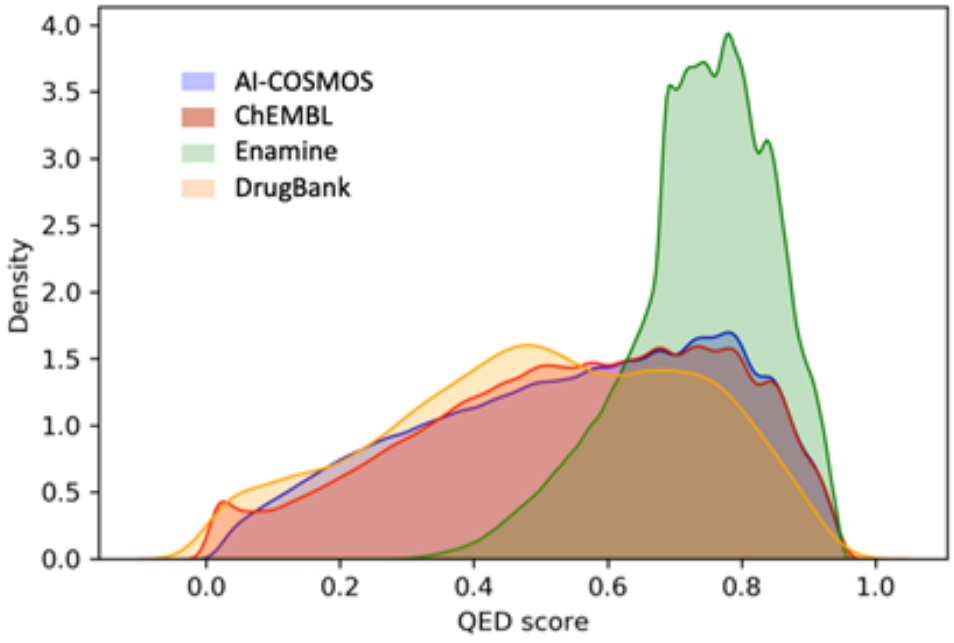
(b) Molecular weight



(c) Hydrogen bond donors



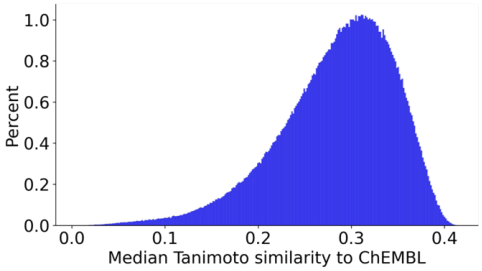
(d) Hydrogen bond acceptors



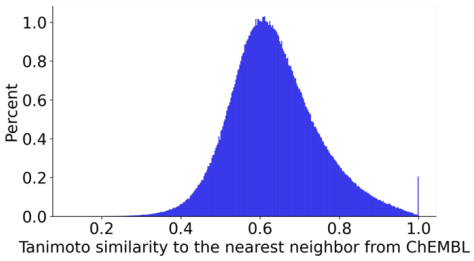
(e) QED score

Figure 2.14: Distribution of properties for AI-COSMOS, ChEMBL, Enamine, and Drugbank.

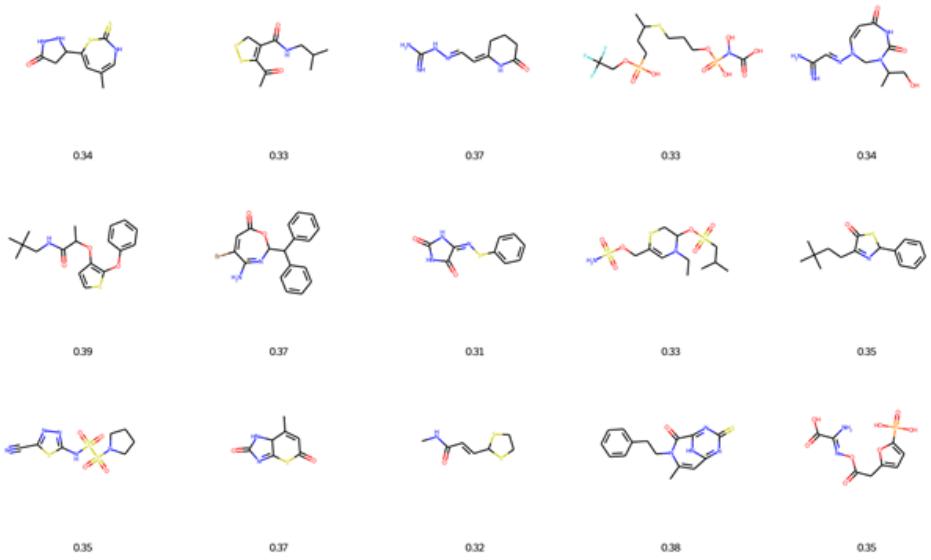
the nearest neighbor and randomly sampled 15 molecules. These molecules and their Tanimoto similarities are shown in Figure 2.15c.



(a) Median T_{sim}.



(b) Maximum T_{sim} .



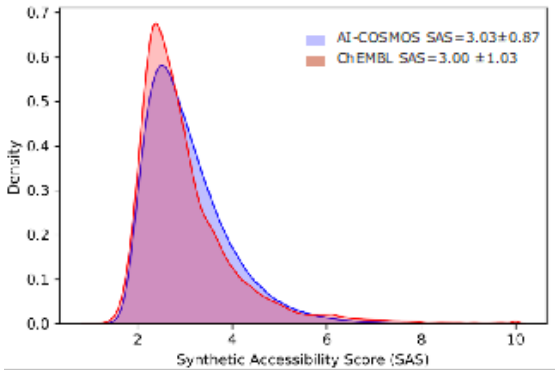
(c) Examples of molecules from bottom 1% of maximum T_{sim} to ChEMBL.

Figure 2.15: The Tanimoto similarity (T_{sim}) between AI-COSMOS diversity set and ChEMBL.

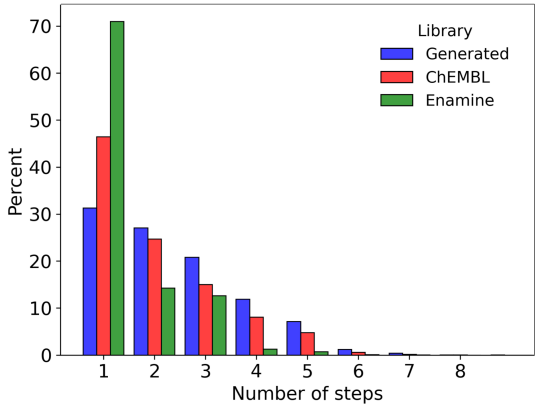
Analysis of synthetic accessibility of the generated molecules. Since the AI-COSMOS library is entirely virtual, it is critical to demonstrate that compounds from this library are synthetically accessible. Although it is impossible to perform extensive experimental synthesis, several computational tools are available to address this question, such as Synthetic Accessibility Score (SAS) [54] and IBM Rxn4Chemistry [67]. SAS is a widely used method to numerically estimate ease of synthesis for druglike molecules based on fragment contributions and complexity penalty. Molecules from catalogs of commercial compounds have a typical range for SAS between

2 and 5 [54]. The advantage of this method is its computation speed. However, it can only provide a qualitative assessment and does not return a synthetic route or a number of reaction steps required to synthesize the input molecule. We used the Molecular Transformer [68] retrosynthetic model from IBM Rxn4Chemistry [67] utility to better address this question. This tool is capable of suggesting possible retrosynthetic routes for a given input molecule. This exercise compares predictions of these computational tools made for the AI-COSMOS library to the ones made for known synthetically accessible druglike compounds such as molecules from ChEMBL [60] or Enamine [69].

We compared SAS results for the AI-COSMOS diversity set to the whole ChEMBL database. Distributions of SAS for both sets are shown in Figure 2.16a. As this figure shows, both distributions match closely with the mean SAS value around 3.0 and standard deviation of 0.87 for generated compounds and 1.03 for ChEMBL. The shape of the distributions is also similar, with heavy tails towards high values of SAS. This leads us to conclude that our generative network produces SMILES string from the same distribution as the training data.



(a) Synthetic Accessibility Score (SAS) distributions



(b) Distribution of the number of steps returned by the retrosynthetic model

Figure 2.16: . Synthetic accessibility assessment results for the AI-COSMOS and ChEMBL libraries

Next, we performed an experiment to assess the retrosynthetic paths for the generated compounds using the IBM Rxn4Chemistry API tool. We randomly selected 11000 from the AI-COSMOS library, ChEMBL and Enamine. To assess the diversity of the selected compounds, we computed the internal similarity for each compound as its Tanimoto distance to the nearest

neighbor. For Tanimoto distance calculation, we chose RDKit topological fingerprints. The vast majority of compounds had Tanimoto distance to its nearest neighbor of less than 0.5, which allows us to conclude that the selected compounds represent diverse subsets. The distributions of internal similarity for all of the chosen subsets are shown in Figure 2.17.

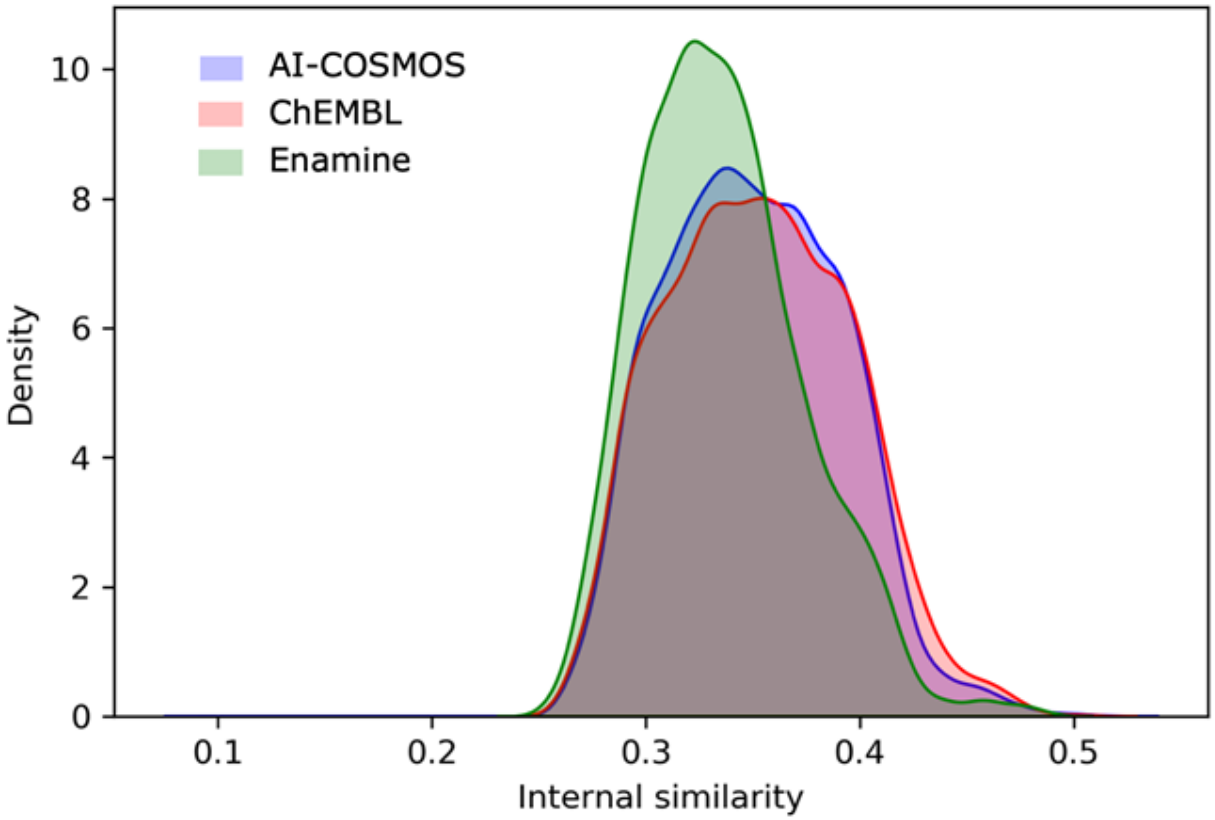


Figure 2.17: The internal similarity of subsets processed with retrosynthetic model.

Next, we computed retrosynthetic paths for the selected compounds using the IBM Rxn4Chemistry tool. This tool takes a query molecule and returns all retrosynthetic paths found by the Molecular Transformer model. Each retrosynthetic path is also assigned a confidence score by the model. A confidence score between 0 and 1 reflects the model's certainty about how feasible the proposed path is. Paths with a score above 0.5 are considered high confidence. IBM Rxn4Chemistry tool uses the eMolecules [70] database of purchasable compounds as building blocks for synthesis. However, sometimes the model couldn't find a retrosynthetic route using commercial building blocks only. In such cases, the retrosynthetic paths also contained non-commercial building blocks. The overall results of the retrosynthetic assessment of the AI-COSMOS library and comparison

to ChEMBL and Enamine are shown in Table 2.5. As Table 2.5 shows, approximately 80% of generated compounds have retrosynthetic paths marked with high confidence by the model, and half of the paths contained only commercial building blocks. Similar results were obtained for molecules from the ChEMBL database. However, we see higher percentages of high confidence routes and routes containing commercial building blocks only for molecules from the Enamine library. This observation can validate the IBM Rxn4Chemistry Molecular Transformer model since the Enamine library is a combinatorial library.

Next, we compared the number of steps in the retrosynthetic routes predicted by the Molecular Transformer model. Since the IBM Rxn4Chemistry tool can return multiple routes for one molecule, we only included the shortest route in this comparison. These distributions for the AI-COSMOS, ChEMBL, and Enamine libraries, are shown in Figure 2.16b. The vast majority of molecules from the Enamine subset are predicted to be synthesized in 2 steps, which is consistent with the Enamine library design and can validate the Molecular Transformer model's predictions. Most of the predicted routes for molecules from the AI-COSMOS and ChEMBL libraries contain up to 5 steps, with a median of 2 steps for ChEMBL and 3 steps for the AI-COSMOS.

Table 2.5: . Results of the retrosynthetic assessment of AI-COSMOS, ChEMBL, and Enamine libraries.

	Total number of compounds	**Compounds with high confidence**	**Compounds with high confidence + commercial building blocks**
AI-COSMOS	11000	8689(79%)	4999(45%)
ChEMBL	11000	9077(83%)	6990(64%)
Enamine	11000	10693(97%)	10397(94%)

2.5.3 Methods

Generative RNN. We used a unidirectional recurrent neural network with GRU [47] layer and augmented memory stack described in Section 2.2.1.1. The size of the hidden layer is 1500. Stack width is 1500, and stack depth is 120. The model was trained with an SGD optimizer, with a learning rate of 0.0002 and batch size of 56 for 100 epochs. The model was implemented with the PyTorch [71] framework. For both training and inference, we used NVIDIA Volta V100 16Gbs

GPUs.

Retrosynthetic model. We used IBM Rxn4Chemistry Python wrapper [72] to obtain retrosynthetic routes for molecules from AI-COSMOS, ChEMBL, and Enamine libraries. We set the time limit of 2 minutes for the API to make predictions, which means that if the server didn't return results within the time limit for a molecule, this molecule was discarded from consideration. As a result, 63% of molecules were processed within the set timeframe: to obtain 11000 synthetic routes per library, we had to submit roughly 17500 queries to the IBM Rxn4Chemistry server through the Python wrapper per library.

Chapter 3

Optimizing properties of generated molecules

3.1 Reinforcement learning for property optimization

3.1.1 Introduction

Deep and reinforcement learning in drug discovery. The development and application of deep generative models for de novo design of molecules with desired properties have emerged as an important modern research direction in Computer-Assisted Drug Discovery (CADD) [73, 74, 75, 76]. Deep generative models can be categorized by the types of molecular representation employed in model development. The most commonly used representations are SMILES strings [31] and molecular graphs. Multiple models for generating SMILES strings [28, 29, 32, 34] and molecular graphs [2, 77, 78, 79, 80] corresponding to synthetically feasible novel molecules have been proposed, including models proposed in this thesis in sections 2.2.1 and 2.3.1.2. Initially, these models are typically trained on a diverse dataset of molecules so that they can generate a broad distribution of molecules. We shall denote a naïve unbiased generative model as a model that has been trained on a generic dataset prior to any specific property optimization.

Reinforcement learning (RL) [32, 81, 82] has been a popular strategy for optimizing properties of the generated molecules. For example, Olivecrona et al.[34] and Blaschke et al.[83]

proposed the REINVENT algorithm and memory-assisted reinforcement learning, respectively, and demonstrated how these approaches could maximize the predicted activity of generated molecules against the 5-hydroxytryptamine receptor type 1A (HTR1A) and the dopamine type 2 receptor (DRD2). Another recent example is the RationaleRL algorithm proposed by Jin et al.[84] The authors used RationaleRL to maximize the predicted activity of inhibitors against glycogen synthase kinase-3 beta (GSK3β) and c-Jun N-terminal kinase-3 (JNK3). Born et al.[85] proposed performing optimization with RL on a merged protein/ligand latent spaced constructed by the VAE. Unfortunately, the aforementioned studies included no experimental validation of the proposed computational hits. Notably, Zhavoronkov et al.[86] not only proposed a novel generative tensorial reinforcement learning algorithm, but also used their method to design potent DDR1 kinase inhibitors, and performed experimental validation of virtual hits.

Most theoretical studies on de novo molecular design employ optimization tasks for properties LogP [2] and Quantitative Estimate of Druglikeness (QED) [57], or the benchmark collection proposed in GuacaMol [87]. Such tasks employ objective metrics obtained directly from a molecule's SMILES [31] or underlying molecular graph through a scoring function. These scoring functions return continuous values that can be used to assign a reward to generated molecules. For example, the Quantitative Estimate of Druglikeness score (QED) has values between 0 and 1.0, with 0 being least drug-like and 1.0 being most drug-like. In such a case, every generated molecule would receive a continuous score: the bigger score values will correspond to bigger reward values, and vice versa. Moreover, a naïve generative model pre-trained on a dataset of drug-like compounds such as ChEMBL [60] would produce molecules with relatively high QED values (see Figure 3.1). In this case, optimization of the generative model via reinforcement learning will proceed efficiently as every generated molecule would get a score. Indeed, the efficient optimization of the QED score has been demonstrated many times in the literature [1, 2, 32]. These benchmarks are unable to simulate tasks with sparse rewards, such as designing molecules with high activity against a specific protein target. In such a case, only a small fraction of generated molecules possess the target property, which leads to reward sparsity during model training.

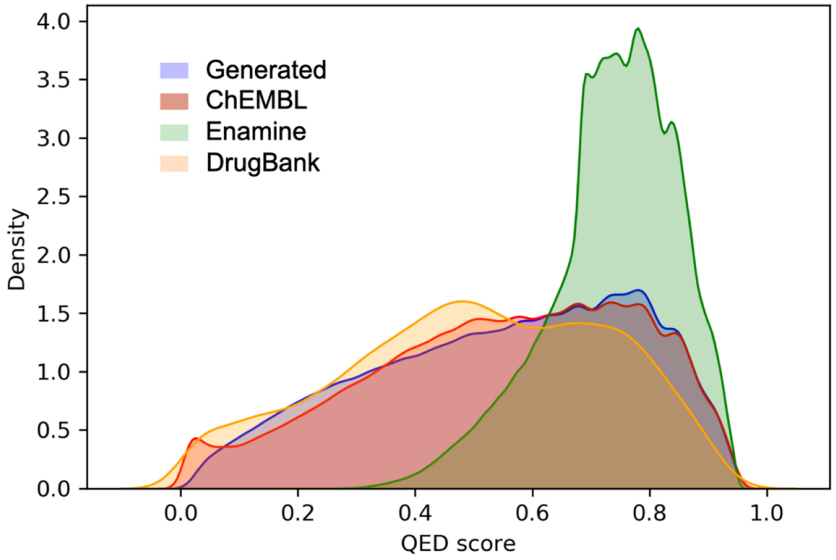


Figure 3.1: Distributions of QED score for molecules from various sources .

3.1.2 Methods

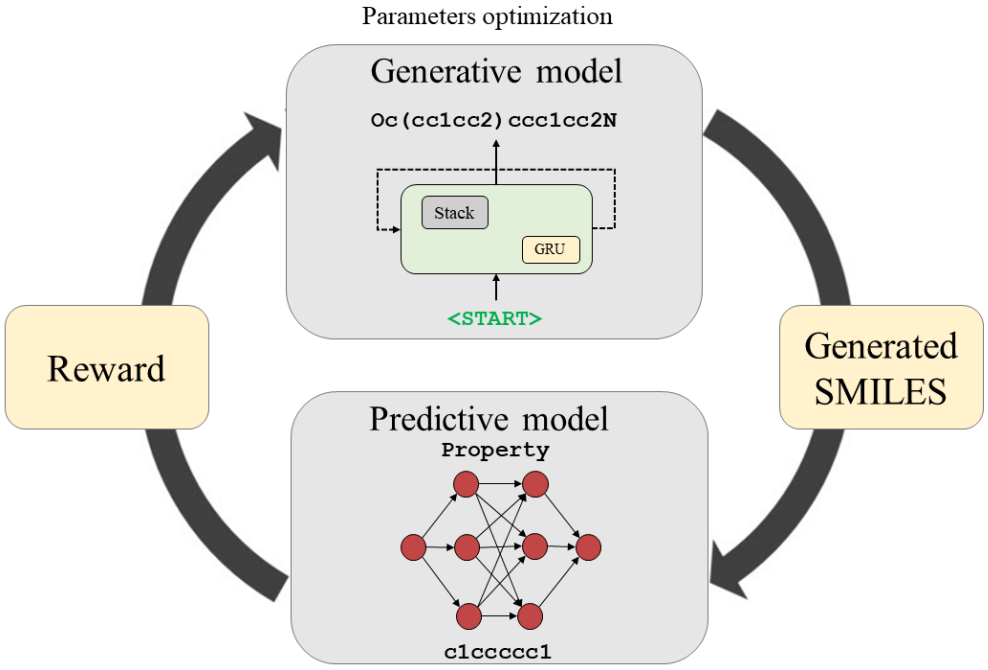


Figure 3.2: General pipeline of a reinforcement learning system for novel compounds generation.

3.1.2.1 Reinforcement learning formulation for SMILES strings

The idea is to combine both generative G and predictive model D into one reinforcement learning system. The set of actions A is defined as an alphabet of SMILES notation. The set of states S is defined as all possible strings in the alphabet with lengths from 0 to some T. The state s_0 with length 0 is unique and considered to be an initial state. The state s_T of length T is called terminal state and it causes episode of molecule generation to end. The subset of terminal states $S^* = \{s_T \in S\}$ of S which contains all the states s_T with length T is called the terminal states set. Reward $r(s_T)$ is calculated in the end of an episode, when the terminal state is reached. Intermediate rewards $r(s_t), t < T$ are equal to 0. In these terms the generator network G can be treated as a policy approximation model. At each time step $t,\ 0 < t < T$, G takes previous state s_{t-1} as an input and estimates probability distribution $p(a_t|s_{t-1})$ of the next action. Afterwards, the next action a_t is sampled from this estimated probability. Reward $r(s_T)$ is a function of the predicted property of s_T by the predictive model D:

$$r(s_T) = f(D(s_T)),$$

where f is chosen expertly depending on the task. Some examples of the functions f are provided further in the computational experiment section. Given these notations and assumptions, the problem of generating chemical compounds with desired properties can be formulated as a task of finding a vector of parameters θ of policy network G which maximizes the expected reward:

$$J(\theta) = \mathbb{E}[r(s_T)|s_0, \theta] = \sum_{s_T \in S^*} G(s_T) r(s_T) \rightarrow \max.$$

This sum iterates over the set S^* of terminal states. In our case this set is exponential and the sum can not be exactly computed. The trick is to approximate this sum as a mathematical expectation by sampling terminal sequences from the model distribution:

$$J(\theta) = \mathbb{E}[r(s_T)|s_0, \theta] = \mathbb{E}_{a_1 \sim p_\theta(a_1|s_0)} \mathbb{E}_{a_2 \sim p_\theta(a_2|s_1)} \dots \mathbb{E}_{a_T \sim p_\theta(a_T|s_{T-1})} r(s_T).$$

So, the procedure for $J(\theta)$ estimation is following: sequentially sample a_t from the model G for t

from 0 to T. The unbiased estimation for $J(\theta)$ is the sum of all rewards in every time step which in our case equals to the reward for the terminal state as we assume that intermediate rewards are equal to 0. As this quantity needed to be maximizes, we need to compute its gradient. This can be done with a REINFORCE algorithm [31] which uses approximation of mathematical expectation as a sum, which we provided above, and the following trick:

$$\partial_\theta f(\theta) = f(\theta)\frac{\partial_\theta f(\theta)}{\partial\theta} = f(\theta)\partial_\theta[\log f(\theta)].$$

So, the gradient of $J(\theta)$ can be written down as:

$$\partial_\theta J(\theta) = \sum_{s_T \in S^*} [\partial_\theta p_\theta(s_T)] r(s_T) =$$

$$= \sum_{s_T \in S^*} p_\theta(s_T)[\partial_\theta \log p_\theta(s_T)] r(s_T) = \sum_{s_T \in S^*} p_\theta(s_T) \left[\sum_{t=1^T} \partial_\theta \log p_\theta(a_t|s_{t-1})\right] r(s_T) =$$

$$= \mathbb{E}_{a_1 \sim p_\theta(a_1|s_0)} \mathbb{E}_{a_2 \sim p_\theta(a_2|s_1)} \dots \mathbb{E}_{a_T \sim p_\theta(a_T|s_{T-1})} \left[\sum_{t=1^T} \partial_\theta \log p_\theta(a_t|s_{t-1})\right] r(s_T),$$

which gives as an algorithm for $\partial_\theta J(\theta)$ estimation.

3.1.2.2 Reinforcement learning formulation for molecular graphs

While generating realistic molecules is an appealing goal, our ultimate aim is to shift the distribution of the generated samples for some desired property. To optimize the chosen property, we use the policy gradient algorithm. In this formulation, MolecularRNN acts as a policy network that outputs the probability of the next action given the current state. The set of actions is defined as the set of atom labels times the set of combinations of possible generated atom connection to the existing graph. The set of states is defined as all possible sub-graphs of graphs with up to a fixed number of N nodes. Consistently with the BFS ordering in MolecularRNN, initial state s_0 is a graph of a single carbon atom. The set of final states is defined as the set of all graphs that correspond to a valid molecule with up to N heavy atoms. The reward $r(s_N)$ for a final state s_N (without loss of generality s_N is used even if $n < N$ in the generated graph) is calculated

with a critic. We distributed the final reward to all intermediate steps, with the discounting factor, which proves to show more stable convergence in our experiments. Thus, intermediate rewards $r(s_i),\ 0 < i < N$ are obtained by discounting the final reward with a fixed factor γ.

The transition probabilities $p(s_i|s_{i-1};\theta)$ are the elements of the product in Equation 2.3. Given those, we can write down the loss function for the policy gradient optimization algorithm by Williams [88], which is designed to maximize the expected reward:

$$L(\theta) = -\sum_{i=1}^{N} r(s_N) \cdot \gamma^i \cdot \log p\left(s_i|s_{i-1};\theta\right). \tag{3.1}$$

3.1.2.3 Structural penalty for MolecularRNN

Valency-based rejection sampling can be used in inference, as was already described. However, the invalid intermediate structures that are obtained during training can provide a useful signal to the model. For example, a molecule can be *almost realistic* except for few invalid bonds. We introduce an additional structure penalty for the atoms that disrespect valencies. Thus, instead of providing a penalty for the whole molecule, we target specific atoms, which results in the modification of parameters that respect valency constraints.

3.1.3 Results

3.1.3.1 Generation of property value biased libraries with the RL system and SMILES-based generative model

To explore the utility of the RL algorithm in a drug design setting, we have conducted case studies to design libraries with three controlled target properties:

- physical properties considered important for drug-like molecule;
- specific biological activity;
- chemical complexity.

For physical properties, we selected melting temperature (T_{melt}) and n-octanol/ water partition coefficient (LogP). For bioactivity prediction, we designed putative inhibitors of Janus protein kinase 2 (JAK2) with novel chemotypes. Finally, the number of benzene rings and the number

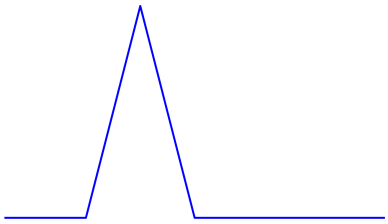
of substituents (like $-OH$, $-NH_2$, $-CH_3$, $-CN$, etc.) was used as a structural reward to design novel chemically complex compounds. Figure 4 shows the distribution of predicted properties of interest in the training test molecules and in the libraries designed by our system. In all cases, we sampled $10,000$ molecules by the baseline (no RL) generator and RL-optimized generative models, and then calculated their properties with a corresponding predictive model. Values of the substructural features were calculated directly from the 2D structure. Table 3.1 summarizes the analysis of generated molecules and the respective statistics.

Table 3.1: Comparison of statistics for generated molecular datasets.

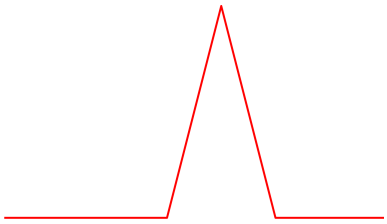
Property		**Valid**	**Mean SA score**	**Mean molar mass**	**Mean property value**	**ZINC15 matches**	**ChEMBL matches**
T_{melt}	baseline	95 %	3.1	435.4	181	4.7%	1.5%
	min	31%	3.1	279.6	137	4.6%	1.6%
	max	53%	3.4	413.2	200	2.4%	0.9%
JAK2	baseline	95%	3.1	435.4	5.70	4.7%	1.5%
	min	60%	3.85	481.8	4.89	2.5%	1.0%
	max	45%	3.7	275.4	7.85	4.5%	1.8%
logP	baseline	95%	3.1	435.4	3.63	4.7%	1.5%
	optimized	70%	3.2	369.7	2.58	5.8%	1.8%
$N_{benzene}$	baseline	95%	3.1	435.4	0.59	4.7%	1.5%
	max	83%	3.15	496.0	2.41	5.5%	1.6%
N_{subs}	baseline	95%	3.1	435.4	3.8	4.7%	1.5%
	max	80%	3.5	471.7	7.93	3.1%	0.7%

Melting temperature (T_{melt}). In this experiment, we set two goals, i.e., either to minimize or to maximize the target property. Upon minimization, the mean of the distribution in the de novo generated library was shifted by $44°C$ as compared to the training set distribution (Figure 3.3c). The library of virtually synthesized chemicals included simple hydrocarbons like butane, as well as poly-halogenated compounds like CF_2Cl_2 and $C_6H_4F_2$. The molecule with the lowest $T_{melt} = -184°C$ in the produced dataset was CF_4. Clearly, this property minimization strategy was extremely effective, as it allowed for the discovery of molecules in the regions of the chemical space far beyond those of the training set of drug-like compounds. In the maximization

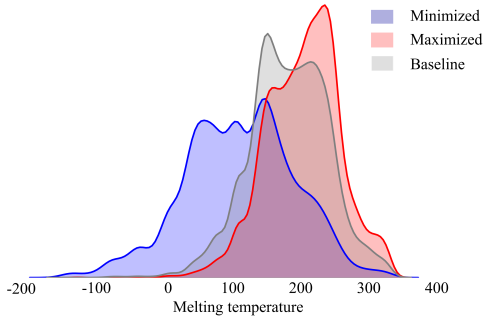
regime, the mean of the melting temperature was increased by $20°C$ to $200°C$. As expected, the generated library indeed included substantially more complex molecules with the abundance of sulphur-containing heterocycles, phosphates, and conjugated double bond moieties.



(a) T_{melt} minimization reward



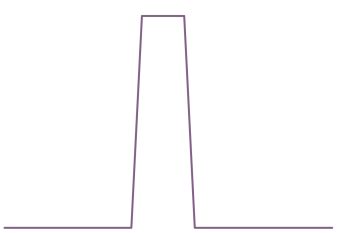
(b) T_{melt} maximization reward



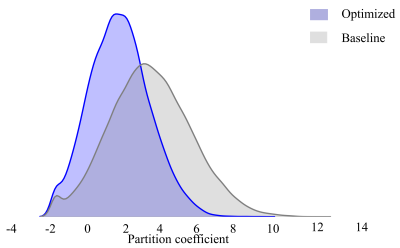
(c) T_{melt} distribution before and after optimization

Figure 3.3: Melting temperature optimization RL experiment.

Designing a chemical library biased toward a range of lipophilicity (LogP). Compound hydrophobicity is an important consideration in drug design. One of the components of the famous Lipinski's rule of five is that orally bioavailable compounds should have the octanol-water partition coefficient LogP less than 5 [89]. Thus, we endeavored to design a library that would contain compounds with LogP values within a favorable drug-like range. The reward function in this case was defined as a piecewise linear function of LogP with a constant region from 1.0 to 4.0 (see Figure 3.4a). In other words, we set the goal to generate molecules according to a typical Lipinski's constraint. As is shown in Figure 3.4b, we have succeeded in generating a library with 88% of the molecules falling within the drug-like region of LogP values.



(a) LogP optimization reward

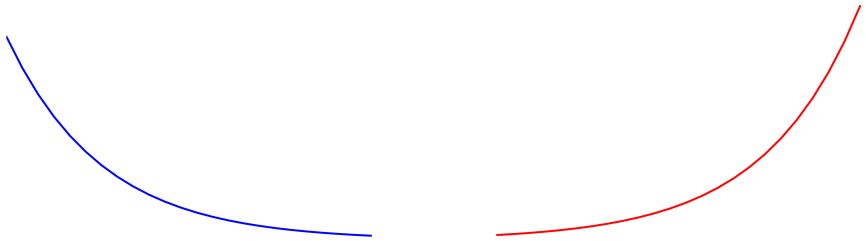


(b) LogP distribution before and after optimization

Figure 3.4: LogP optimization RL experiment.

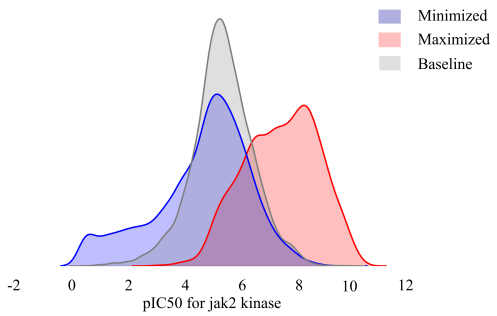
Inhibition of JAK2. In the third experiment, which serves as an example of the most common application of computational modeling in drug discovery, we have employed our system to design molecules with the specific biological function, i.e., JAK2 activity modulation. Specifically, we designed libraries with the goal of minimizing or maximizing pIC50 values for JAK2. While most of drug discovery studies are oriented toward finding molecules with heightened activity, bioactivity minimization is also pursued in drug discovery to mitigate off-target effects. Therefore, we were interested in exploring the ability of our system to bias the design of novel molecular structures toward any desired range of the target properties. JAK2 is a non-receptor tyrosine kinase involved in various processes such as cell growth, development, differentiation or histone modifications. It mediates essential signaling events in both innate and adaptive immunity. In the cytoplasm it also plays an important role in signal transduction. Mutations in JAK2 have been implicated in multiple conditions like thrombocythemia, myelofibrosis or myeloproliferative disorders [90]. The reward functions in both cases (min and max) were defined as exponential functions of pIC_{50} (see Figures 3.5a, 3.5b). The results of library optimization are shown in Figure 3.5c. With minimization, the mean of the predicted pIC_{50} distribution was shifted by about one pIC_{50} unit and the distribution was heavily biased toward the lower ranges of bioactivity with 24% of molecules predicted to have practically no activity ($\text{pIC}_{50} \leq 4$). In the activity maximization exercise, properties of generated molecules were more tightly distributed across the predicted activity range. In each case, our system virtually synthesized both known and novel compounds, with the majority of *de novo* designed molecules being novel compounds. The generation of known compounds (i.e.

not included in the training set) can be regarded as model validation. Indeed, the system retrospectively discovered 793 commercially available compounds deposited in the ZINC database, which constituted about 5% of the total generated library. Importantly, as many as 15 of them (exemplified by ZINC263823677 -http://zinc15.docking.org/substances/ZINC000263823677/ and ZINC271402431 - http://zinc15.docking.org/substances/ZINC000271402431/) were actually annotated as possible tyrosine kinase inhibitors.



(a) JAK2 minimization reward

(b) JAK2 maximization reward



(c) Distribution of pIC_{50} of JAK2 before and after optimization

Figure 3.5: JAK2 optimization RL experiment.

Substructure bias. Finally, we also performed two simple experiments mimicking the strategy of biased chemical library design where the designed library is enriched with certain user-defined substructures. We defined the reward function as the exponent of

- the number of benzene rings (-Ph);
- total number of small group substituents.

Among all case studies described, structure bias was found to be the easiest to optimize. The results of the library optimization study are shown in Figures 3.7b and 3.7d. Furthermore, Figure 3.6 illustrates the evolution of generated structures as the structural reward increases. Indeed, we see that the model progresses toward generating increasingly more complex, yet realistic molecules with greater numbers of rings and/or substituents.

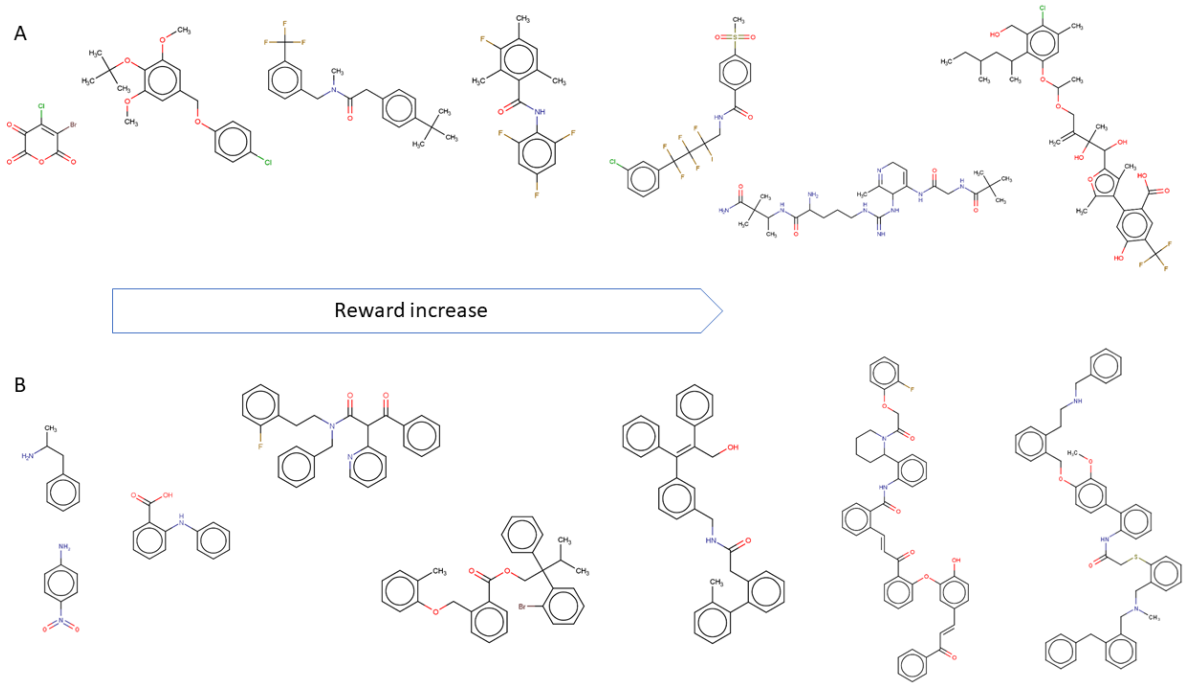
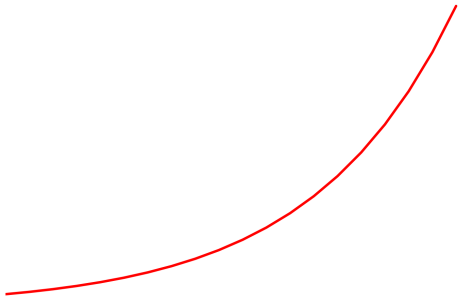


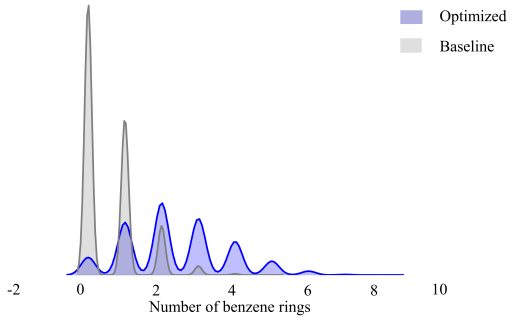
Figure 3.6: Evolution of generated structures as chemical substrcture reward increases for number of substitutents (top) and benzene rings (bottom).

We expect that designing structurally biased libraries may be a highly desirable application of the ReLeaSE approach as researchers often wish to generate libraries enriched for certain privileged scaffold(-s) and lead compounds optimization [91]. Conversely, the system also allows to avoid particular chemical groups or substructures (like bromine or carboxyl group) that may lead to undesired compound properties such as toxicity. Finally, one could implement certain substructure, or pharmacophore similarity, reward to explore additional chemical space. Table 3.1 shows a decrease in the proportion of the valid molecules after the optimization. We may explain this phenomenon by the weaknesses of predictive models P and the integration of predictive and generative models into a single design system. We presume that the generative model G

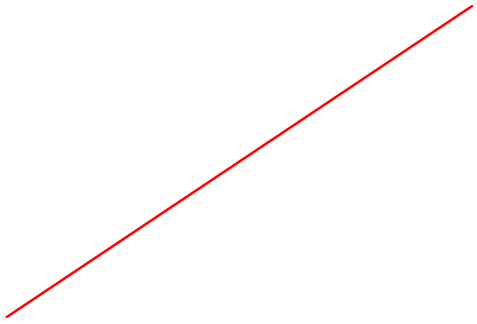
tends to find some local optima of the reward function that correspond to invalid molecules, but predictive model P assigns high rewards to these molecules. This explanation is also supported by the results of structure bias optimization experiments, as we did not use any predictive models in these experiments and the decrease in the proportion of valid molecules was insignificant. We also noticed, that among all experiments with predictive models, those with LogP optimization showed the highest proportion of valid molecules and, at the same time, the predictive model for LogP estimation had the highest accuracy $R^2 = 0.91$. Probably it is harder for RL system to exploit high quality predictive model and produce fictitious SMILES strings with predicted properties in the desired region.



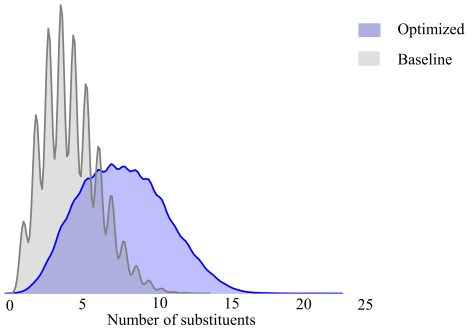
(a) Number of benzene rings maximization reward



(b) Distribution of number of benzene rings before and after optimization

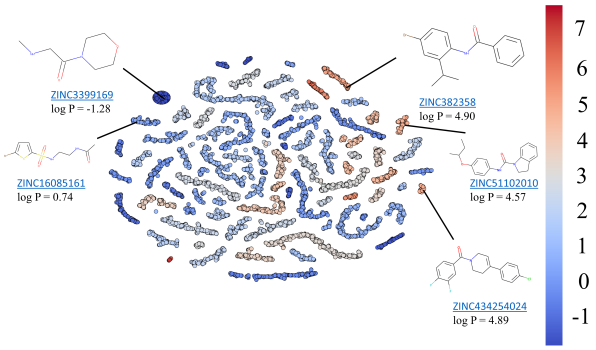


(c) Number of substituents maximization reward

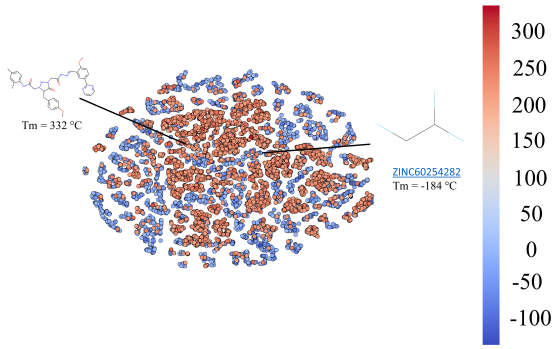


(d) Distribution of number of substituents before and after optimization

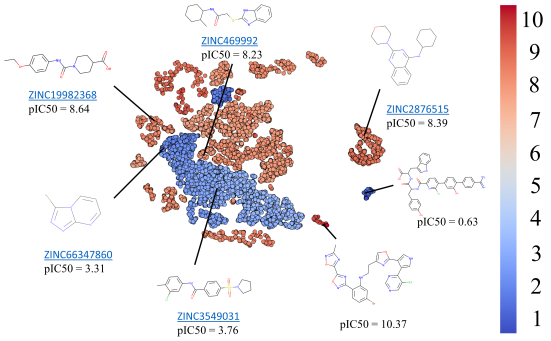
Figure 3.7: Property distributions for RL-optimized versus baseline generator model.



(a) Partition coefficient, logP



(b) Melting temperature, T_melt,



(c) Predicted activity, pIC_{50}

Figure 3.8: Clustering of generated molecules by t-distributed stochastic neighbor embedding (t-SNE).

3.1.3.2 Visualization of new chemical libraries

In order to understand how the generative models populate chemical space with new molecules, we used t-Distributed Stochastic Neighbor Embedding (t-SNE) for dimensionality reduction [92]. We selected datasets for three endpoints used in our case studies (Tm, LogP, JAK2) that were produced with corresponding optimized generative models G. For every molecule, we calculated a latent vector of representation from the feed-forward layer with ReLU activation function in the predictive model P for the respective property and constructed 2D projection using t-SNE. These projections are illustrated in Figure 3.8. Every point corresponds to a molecule and is colored according to its property value.

For libraries generated to achieve a certain partition coefficient distribution (Figure 3.8a), we can observe well-defined clustering of molecules with similar LogP values. In contrast, for melting temperature (Figure 3.8b) there are no such clusters. High and low T_{melt} molecules are intermixed

together. This observation can be explained by the fact that melting temperature depends not only on the chemical structure of the molecule, but also on intermolecular forces as well as packing in the crystal lattice. Therefore, plotting molecules in this neural net representation could not achieve good separation of high vs. low T_{melt}. In the case of the JAK2 model (Figure 3.8c), we could observe two large non-overlapping areas roughly corresponding to inactive ($pIC_{50} < 6$) and active($pIC_{50} \leq 6$) compounds. Inside these areas, molecules are typically clustered around multiple privileged scaffolds. Specifically for JAK2 we see abundance of compounds with 1,3,5-triazine, 1,2,4-triazine, 5-Methyl-1H-1,2,4-triazole, 7H-pyrrolo[2,3-d]pyrimidine, 1H-pyrazolo[3,4-d]pyrimidine, thieno-triazolo-pyrimidine and other substructures. Overall, this approach offers a rapid way to visualize compound distribution in chemical space in terms of both chemical diversity and variability in the values of the specific prediction endpoint. Furthermore, joint embedding of both molecules in the training set and those generated de novo allows one to explore differences in the chemical space coverage by both sets and establish whether structurally novel compounds also have the desired predicted property of interest.

3.1.3.3 Property optimization with reinforcement learning and MolecularRNN

We performed experiments on the properties optimization of generated molecules starting with our strong pretrained model with the policy gradient algorithm (section 3.1.2.2). We choose maximization of penalized logP as defined in [2] and QED [57] starting from MolecularRNN that is likelihood-pretrained on ZINC 250k dataset. We also performed an additional experiment with maximization of melting temperature. This is an appealing exercise because it requires training an additional model for melting temperature prediction, while logP and QED can be computed directly from the molecular graph structure. This experiment mimics realistic drug discovery scenario, where toxicity or bioactivity is optimized. It paves the way for further research in this important direction.

Penalized logP and QED maximization. As in [1, 2], we independently maximize two properties – penalized logP and QED score. MolecularRNN is tuned for 300 iterations with a generated batch size of 512 and Adam optimizer with a constant learning rate of 10^{-5}. The objective

function in Equation 3.1 maximizes the following rewards:

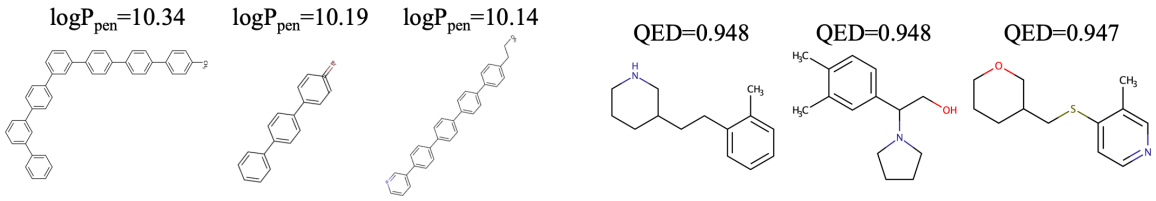
$$r(mol) = 5 \cdot \log P_{pen}(mol)$$

$$r(mol) = 10 \cdot QED(mol).$$

We use discount factor $\gamma = 0.97$. The best 3 molecules after optimization for both properties are shown in Table 3.2, and demonstrates the distribution shift. In this experiment, our model outperforms all baselines in both tasks. The top 3 molecules are shown in Figure 3.9. Samples with high logP values are very realistic, as the model learned to grow a chain of aromatic rings that would very strongly bind to a lipid membrane (high lipophilicity). This is an indicator that the model learned some underlying physics about relationship between molecular structure and properties.

Table 3.2: Comparison of the top 3 scores for penalized logP and QED.

	Penalized logP				QED score			
Methods	**1st**	**2nd**	**3rd**	**Valid**	**1st**	**2nd**	**3rd**	**Valid**
ZINC	4.52	4.30	4.23	100%	0.948	0.948	0.948	100%
[81]	3.63	3.49	3.44	0.4%	0.896	0.824	0.820	2.2%
JT-VAE [2]	5.30	4.93	4.49	100%	0.925	0.911	0.910	100%
GCPN [1]	7.98	7.85	7.80	100%	0.948	0.947	0.946	100%
MolecularRNN	**10.34**	**10.19**	**10.14**	**100%**	**0.948**	**0.948**	**0.947**	**100%**



(a) penalized logP (b) QED

Figure 3.9: Top 3 molecules for MolecularRNN optimized with policy gradient

We took a step further and not only looked at molecules with top 3 scores but also considered the full distribution of the maximized QED for libraries gen-

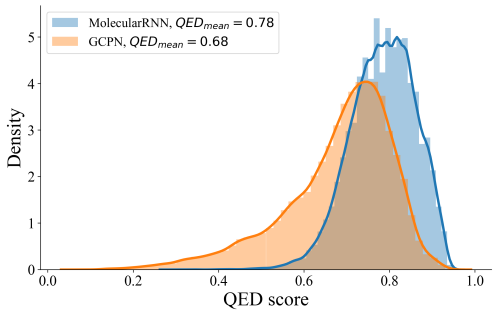
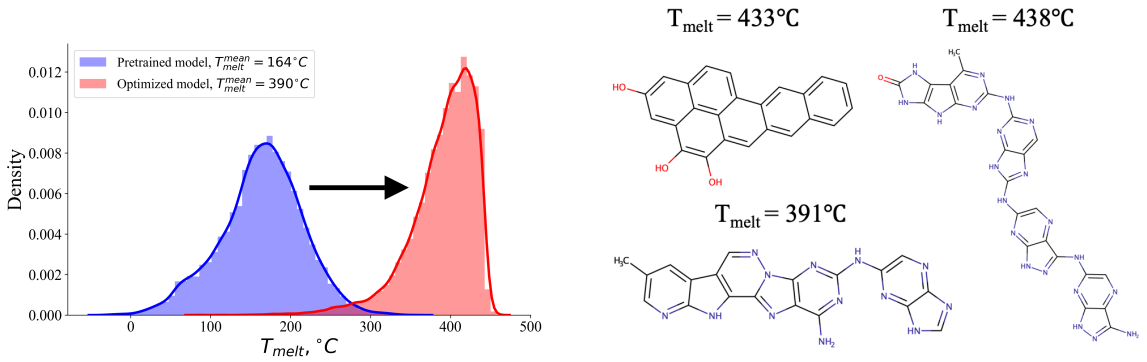


Figure 3.10: Distribution of maximized

erated with our MolecularRNN and GCPN [1] as the best baseline. We argue that reporting only the top 3 scores is not the most informative benchmarking metric, since top 3 may not reflect the real performance of the model. Instead, we encourage reporting the statistics of the optimized distribution. Figure 3.10 shows that MolecularRNN shifts the distribution father to the maximum values of QED compared to GCPN.

Melting temperature maximization. We train a graph convolution regression model introduced in [93] for predicting the melting point of a molecule. Training and test datasets were 37940 and 9458 objects correspondingly; with T_{melt} ranging from $-196°C$ to $517°C$. The model has 4 layers with hidden sizes of 128. We use Adam optimizer, starting with a learning rate of 0.001 and exponential decay with $\gamma = 0.8$ after every epoch. The model is trained with a batch size of 32 for 30 epochs. The model converges to RMS error of $39.5°C$, that is comparable to the state-of-the-art for the same dataset [94]. This model is then used to assign a reward function $r(mol) = \exp(t_{pred}(mol) + 1)$, where $t_{pred}(mol)$ is the normalized predicted melting temperature for a molecule.

For this experiment, we used a model pretrained on ChEMBL dataset and optimized it with the same settings as in the previous experiments – 300 iterations with a batch size of 512 and Adam optimizer with a constant learning rate of 10^{-5}. Figure 3.11a shows the relative distribution shift of predicted property for the molecules sampled from the pretrained model and for the molecules sampled from the optimized model. Example of generated molecules with predicted values of T_{melt} are shown in Figure 3.11b. Interestingly, in this experiment, MolecularRNN rediscovered two known chemical phenomena. First, fusing multiple aromatic rings significantly increases the T_{melt}. Second, the presence of C=O, OH, NH_2 and heterocyclic nitrogens make molecules more polar. This usually enhances dipole-dipole interactions and subsequently increases T_{melt} as well.



(a) Distribution of predicted melting temperature for base and optimized models

(b) Examples of generated molecules with highest melting temperature

Figure 3.11: Melting temperature maximization

3.2 Heuristics for improving property optimization

3.2.1 Introduction

The problem of sparse rewards in reinforcement learning. In contrast to physical properties such as LogP that can be calculated directly from molecular structure, the biological activity of a novel compound designed to bind the desired protein target cannot be predicted from its chemical structure alone. A common way to predict the binding affinity of novel, untested ligands is by using Quantitative Structure-Activity Relationship (QSAR) models [95, 96] trained on historical experimental data for a protein target of interest using machine learning techniques. These models have either continuous outputs (pKd, pIC_50, etc.) for regression problems or categorical outputs (active/inactive class label in a binary case) for classification problems. QSAR models could, in principle, be used to construct a reward function for reinforcement learning to optimize the binding affinity of generated molecules, as was shown, for instance, in section 3.1.2.2 of this thesis. However, unlike physical molecular properties like LogP that every molecule possesses, specific bioactivity is a target property that exists for only a small fraction of molecules, which leads to reward sparseness in the generative models. This sparse rewards problem represents a serious obstacle for the effective use of reinforcement learning for designing molecules with high activity. Indeed, the low success probability often leads to the overwhelming majority of training trajectories resulting in a zero reward, which implies that the reinforcement learning agent or

policy network struggles to explore the environment and learn the optimal strategy for maximizing the expected reward [97, 98, 99]. Thus, a promising molecule with high bioactivity for a protein of interest is unlikely to be observed if molecules are randomly sampled from a naïve generative model.

Training the generative network to optimize the potency of generated molecules against a desired protein target is an excellent example of a reinforcement learning problem with sparse rewards. There is a very low chance of observing a molecule with high potency when sampled randomly from the distribution of unoptimized generative model. During training procedure with RL, training examples are produced by the generative model. The model trained just on negative examples (molecules with low potency values) is unlikely to discover positive examples (molecules with high potency values). In this section, we demonstrate that the naïve generative model produces molecules predicted to be inactive in most cases. Under such a scenario, the naïve generative model rarely observes good examples and fails to maximize the active class probability for generated ligands. We further address this problem by proposing a set of heuristic approaches (a "bag of tricks") combined with reinforcement learning in the sparse rewards situation to increase the efficiency of optimizing the structures of generated molecules to have higher predicted active class probability. Using the epidermal growth factor receptor (EGFR) ligands as a case study, we show that by combining a reinforcement learning pipeline for generative model optimization with proposed heuristics, we could overcome sparse reward issues and successfully rediscover known active scaffolds for EGFR using the feedback from the classification QSAR model only. In addition to methodological advances, we also performed experimental bioassay validation of the novel generated hit molecules, which confirmed the experimental activity of virtual hits which will be covered in chapter 4 of this thesis in more detail.

Major findings. We performed a series of experiments that resulted in the following chief observations:

1. The generative model trained with only the policy gradient algorithm could not discover any molecules with high active class probability for EGFR due to sparse rewards.
2. The combination of policy gradient algorithm with proposed fine-tuning by

- transfer learning;
- experience replay;
- real-time reward shaping.

3. Experimental testing of selected computational hits that could be obtained from a commercial source validated the efficiency of our proposed approach for discovering novel bioactive molecules.

Below, we discuss how we arrived at the above observations. Overall, results of this study consist of two main parts. In the first part covered in this chapter, we describe our computational analysis concerning the first two observations. In the second part covered in chapter 4, we discuss the generation, selection, and experimental bioactivity testing of computational hit compounds for an important cancer biological target, epidermal growth factor receptor (EGFR). The most active compound featured a privileged EGFR scaffold found in the known active molecules. Notably, the training set was not enriched for this scaffold as compared to other scaffolds and this scaffold was not used selectively as part of the reinforcement learning procedure.

3.2.2 Methods

In this section, we describe enhancements of deep learning and reinforcement learning approaches used to generate virtual molecules with desired properties. Briefly, we employ the reinforcement learning pipeline introduced in section 3.1.2.2 with several novel improvements to overcome the problem of sparse rewards. Below we will talk about each part of the pipeline, introduce our novel tricks and heuristics in more detail, and discuss an EGFR case study.

Generative model. For the generative model, we used a deep recurrent neural network with an augmented memory stack described in sections 2.2.1 and 2.2.1.1. This network is trained to produce novel molecules in the form of SMILES strings [31]. The network has two modes – training mode and inference mode. In the training mode, the model receives a SMILES string from the training set and tries to reconstruct it, starting from the given prefix. The model is essentially trained as a multiclass classifier, where classes are represented as symbols in the

SMILES string alphabet. In the inference mode, instead of receiving prefix from the training set, the model iteratively takes its output as new inputs to generate the next symbol based on the previously generated ones. The generation stops when the network produces a unique stop token interpreted as a command to end generation. The model is implemented as a part of OpenChem [100] (https://github.com/Mariewelt/OpenChem) – an open-source deep-learning toolkit for computational chemistry and drug design, which is discussed in detail in Chapter 5.

Reinforcement learning. For the method for shifting the distribution of predicted target active class probability for generated molecules, we used the policy gradient algorithm [88]. We adapted the problem to a reinforcement learning setting by treating the generative model as the policy network. In this formulation, the generative model predicts the probability of the next action, i.e., adding a new character to the SMILES string prefix. The set of actions is then limited to the SMILES alphabet. The set of states is then limited to all strings in the SMILES alphabet with lengths up to a specific limit N, where N is a hyperparameter defined by the maximum length of SMILES strings from the training dataset. According to the policy gradient algorithm, the objective function to be maximized is defined as the expected reward:

$$L(\Theta) = -\sum_{i=1}^{N} r(s_N) \cdot \gamma^i \cdot \log\left[p(s_i|s_{i-1};\Theta)\right], \tag{3.2}$$

,

where s_N is the generated SMILES string, $s_i, i = 1, \ldots, N$ is the prefix of s_N of length $0 < i < N$, γ is the discount factor, $p(s_i|s_{i-1};\Theta)$ is the transition probability obtained from the generative model, and $r(s_N)$ is the value of the reward function for the generated SMILES string based on the output of the predictive model of active class probability for EGFR activity.

3.2.2.1 Exploration and exploitation trade-off.

An encounter of a molecule active against a specific target (e.g., EGFR) is a rare event, so the generative model may very infrequently observe promising molecules. Such a scenario will result in over-exploration – a situation when the model mostly experiences low rewards for inactive

molecules and receives insufficient signal to shift the distribution of the generated samples. At the same time, the model should not over-exploit the historical data since our ultimate goal is to generate novel active molecules. Thus, we do not want to over-exploit information about known active molecules from the historical data, so that it can generate novel active molecules. We address this problem by complementing the classic policy gradient algorithm with novel heuristics detailed below to balance exploitation and exploration while training the model to maximize the predicted activity active class probability of the generated molecules.

Fine-tuning by transfer learning on high-reward examples. The first algorithmic advance we have explored was to fine-tune the model by transfer learning using generated molecules with high rewards as training samples. Fine-tuning means training the model by minimizing cross-entropy loss in the same manner as during the pretraining stage. A similar idea has already been introduced in the literature [98]. Our approach differs from previous approaches through our selection process for fine-tuning training samples. Whereas the previous work uses historical data with high experimental activities, we used generated molecules as training samples, whereas the previous work uses historical data with high experimental activities. Overall, fine-tuning by transfer-learning results in high exploitation and low exploration. With sufficient rounds of fine-tuning, the generative model produces molecules highly similar to those used for fine-tuning. Thus, training on historical data results in the exploitation of already known chemical scaffolds instead of discovering novel scaffolds. Such an approach could be suitable for the lead optimization process when the goal is to optimize molecules with a prespecified scaffold. In contrast, fine-tuning on generated molecules with high rewards results in the exploitation of scaffolds produced by the generative network and highly scored by the predictive model. Generated scaffolds could be novel, thus increasing their potential in drug discovery applications.

Experience replay on high-reward molecules. Another technique that we proposed addresses the problem of sparse rewards while maintaining balancing the exploration-exploitation trade-off. To perform experience replay, we save high-reward trajectories (molecules) to the replay buffer. We randomly draw experience samples from the replay buffer during training and let the generative network follow the experience trajectory through teacher forcing [101]. We then calculate the

expected reward maximization loss function and apply policy gradient updates to the generative network parameters. The concept of using experience replay for reinforcement learning is not new and has previously proven to be an effective training method in the reinforcement learning domain [102, 103, 104]. We propose using this approach to deal with rare high-reward molecules while avoiding over-exploitation. Like the fine-tuning scenario, we utilize generated molecules with high rewards as training examples (or experiences) in the experience replay. Unlike the fine-tuning scenario, experience replay does not directly enforce specific characters in the generated SMILES string. Instead, it provides feedback in the form of a high reward at the end of the replay episode, resulting in less exploitation.

Real-time reward shaping. Real-time reward shaping is one more of our proposed advancements to train the neural network more efficiently in a situation when molecules with high rewards are observed rarely. The idea behind this technique is to change the reward function over training dynamically. We shall explain this concept using a threshold reward function and a predictive model returning the active class probability as an illustrative example. A molecule is considered active in these settings if the returned probability exceeds some threshold, such as 0.5. At the beginning of the training process, very few generated molecules will have such a high probability; instead, there often is a cohort of molecules with probabilities slightly higher than zero. The real-time reward shaping technique helps the model exploit molecules with non-zero predicted active class probabilities in the absence of good examples. We introduce the probability threshold p_0 to differentiate between good and bad examples in our threshold reward function:

$$R(s) = \begin{cases} r_{pos}, \text{if } p(s) > p_0, \\ r_{neg}, \text{ otherwise}, \end{cases} \tag{3.3}$$

where s is the generated molecule, $p(s)$ is the probability of active class returned by the predictive model, p_0 is the probability threshold, r_{pos} is the reward value for good examples, and r_{neg} is the reward value for bad examples. The probability threshold p_0 is initialized to a small value and dynamically increased during training. After several iterations of training, we generate a large enough batch of molecules with the current model and predict active class probabilities with

the predictive model. The threshold p_0 is increased if the big enough portion of molecules has predicted active class probabilities bigger than the threshold's current value. In our experiments, we started with $p_0 = 0.05$ and increased it by 0.05 when at least 15% out of 3000 generated molecules have predicted probabilities of active class greater than p_0.

3.2.3 Results

Model pipeline. Neural network optimization is a nontrivial task as a network's hyperparameter values define a training protocol. Due to the high number of hyperparameters, the training space is vast. To complicate things further, neural network training is a computationally expensive task that can last hours to days. The choice of training hyperparameters thus has a significant influence on model quality. We sought to run a benchmark experiment to investigate how different training techniques interact and how they affect model quality. As a case study, we performed the optimization of the generative model with reinforcement learning to maximize the predicted probability of active class for EGFR protein. The experimental training pipeline is shown in Figure 3.12.

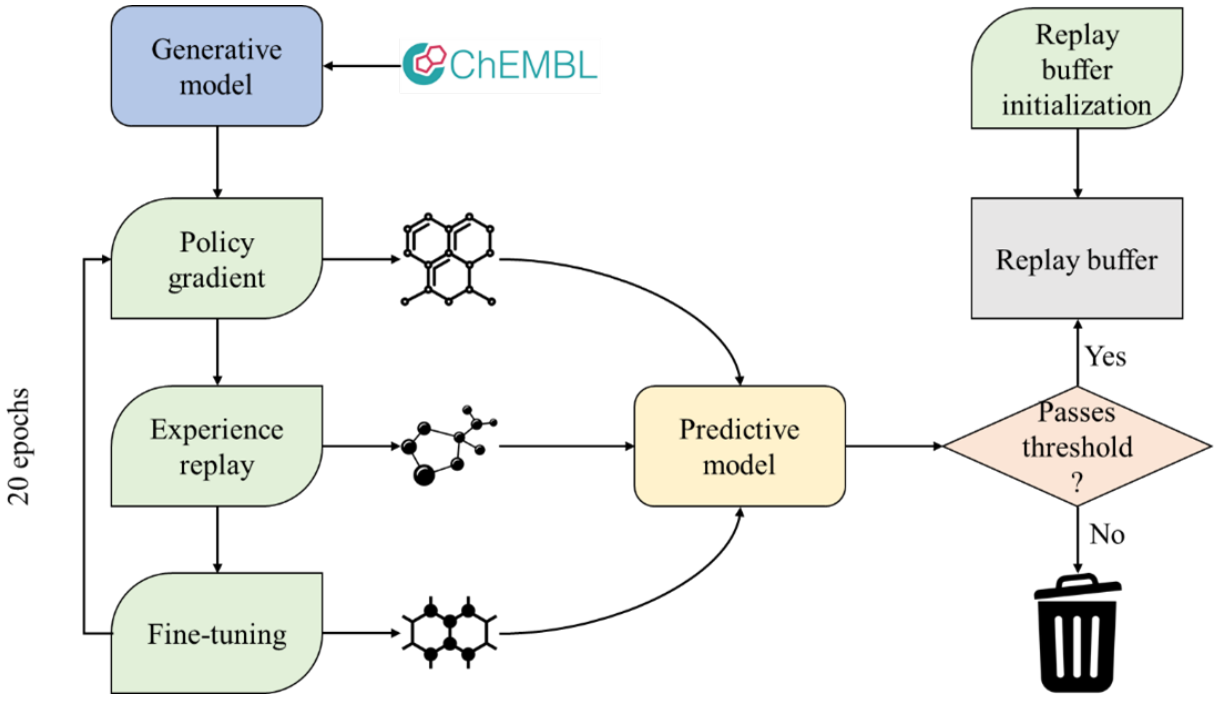


Figure 3.12: EGFR model training pipeline.

Model training consists of 2 stages – pre-training the generator from scratch on a vast dataset such as ChEMBL [60] in a supervised manner to produce mostly valid SMILES strings without any property optimization at this point. The second stage is training the model with RL to optimize the property values of the generated molecules. We used the pre-trained ChEMBL model and populated the experience replay buffer with generated predicted active molecules to initialize training. The model was trained using different combinations of policy gradient, experience replay, and fine-tuning. At the end of each substep, 3200 molecules were generated for intermediate evaluation. If experience replay and/or fine-tuning were used, molecules with predicted active class probability exceeding the probability threshold were admitted into the experience replay buffer. In turn, the replay buffer influences training at the policy replay and fine-tuning steps in the next epoch if used. At the end of the training, the model generated 16000 molecules for evaluation. We first trained the model for a variable number of epochs and verified that the model learns significantly after 20 epochs. Models were trained for 20 epochs for nine different combinations of fine-tuning and experience replay with the following options: no fine-tuning, 20 iterations of fine-tuning, or 100 iterations of fine-tuning; and no experience replay, 10 iterations of experience replay, and 20 iterations of experience replay. The number of policy gradient steps was adjusted so that each training epoch had 25 iterations of replay and policy gradient (e.g. 25 policy steps for 0 replay steps and 15 policy steps for 10 replay steps) (Figure 3.13).

We used Random Forest ensemble model as a predictor in this pipeline. The ensemble model consists of 5 individual Random Forest models trained in a 5-fold cross-validation manner, and the final prediction is the mean of predictions from each model in the ensemble.

Effect of fine-tuning vs. reinforcement learning. The bar chart shown in Figure 3.14 summarizes the findings for four representative conditions:

- policy gradient only;
- policy gradient and fine-tuning;
- policy gradient and experience replay;
- policy gradient, experience replay, and fine-tuning.

We assessed the extent of overfitting by recording the fraction of the generated trajectories

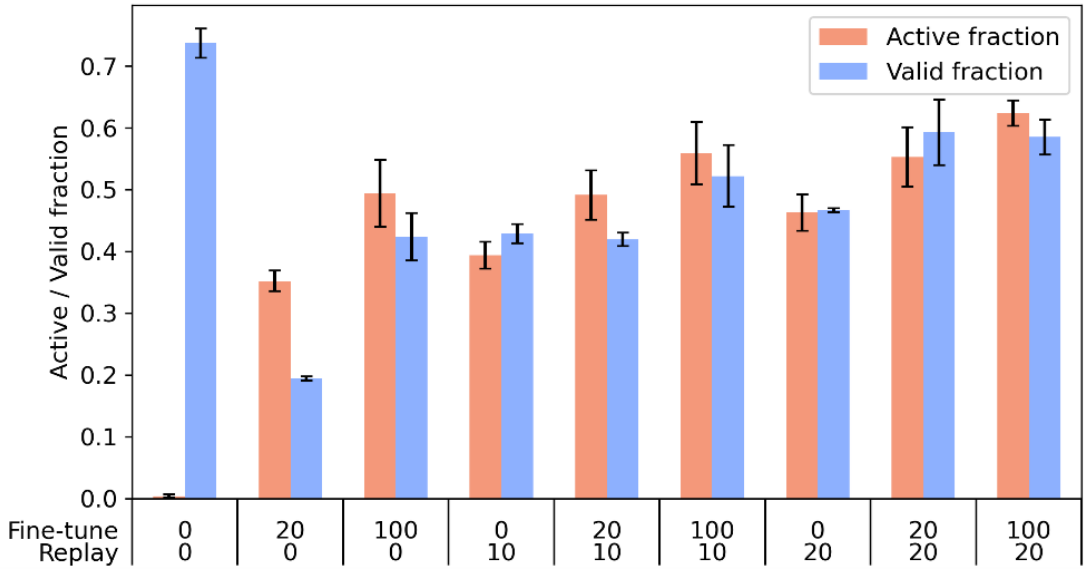


Figure 3.13: Combined effects of fine-tuning and reinforcement learning.

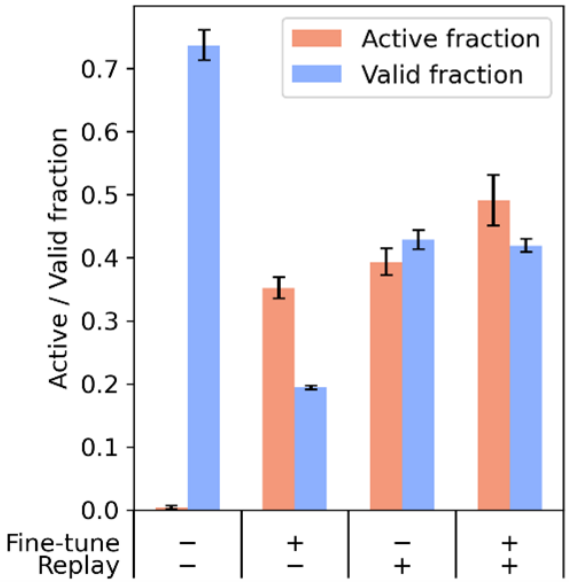


Figure 3.14: Combined effects of fine-tuning and reinforcement learning.

that generate valid SMILES strings, which is defined as the ratio of valid and unique SMILES strings over the total number of the generated trajectories. In more detail, the model can overfit with respect to the property predictor. For example, if the QSAR model assigns high active class

probabilities to molecules with a specific chemical group, the generative model can discover and exploit it by stacking multiple aforementioned chemical groups into a single molecule. Such scenario often leads to decrease in validity. We use the ratio of valid and unique SMILES strings over the total number of generated trajectories. This metric would detect mode collapse, since we are discarding repeated molecules. We assessed the extent of model learning by recording the fraction of the generated trajectories resulting in active chemical structures, which is defined as the ratio of valid SMILES strings with predicted EGFR activity (with the arbitrary probability threshold of 0.75) over the number of valid and unique SMILES strings generated. Training without replay tricks has a near-zero "active" fraction and the highest "valid" fraction. This observation is consistent with the sparse rewards hypothesis. In the absence of rewards from active molecules, this model effectively trains on the classifier objective. Instead of learning to generate active molecules, the model optimizes valid fraction. Training with a single trick (fine-tuning or experience replay) teaches the model to generate active molecules, albeit at the expense of a lower valid fraction. Training with only fine-tuning results in a lower fraction of valid molecules. Training with both experience replay and fine-tuning yields the best results, with both high active fraction and high valid fraction.

Mode collapse effect. Next, we analyzed the effect of fine-tuning steps on mode collapse [105]. Mode collapse poses a significant challenge in generative models. Reinforcement learning teaches generative models to produce output with high reward; however, it does not consider the distribution of generated output. Thus, the model can discover a pathological local minimum in the objective function by converging to generate a few instances with high reward; in such cases, the model undergoes mode collapse. Such overfitted models explore limited regions of chemical space and are undesirable for library generation.

Our experiments used the active fraction as a proxy for training progress and the valid and unique fraction as a proxy for mode collapse. Two scenarios can decrease the valid fraction:

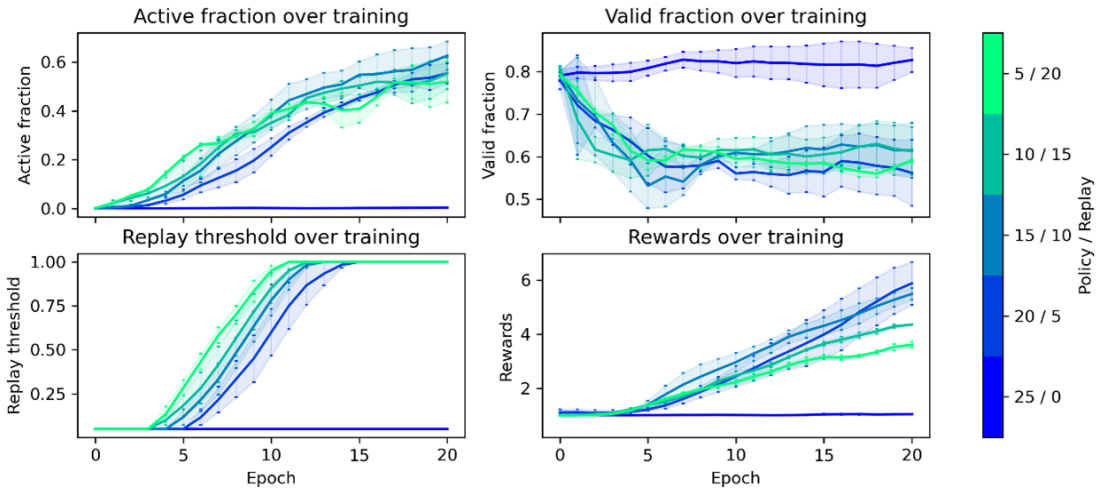
1. the model generates a larger fraction of invalid SMILES strings (fewer valid SMILES strings);
2. the model suffers from mode collapse and generates many repeats of the same SMILES

string (fewer unique SMILES strings)

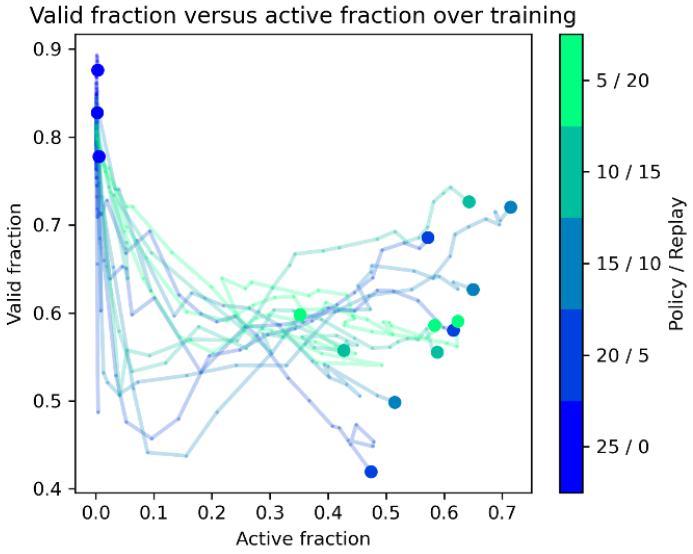
The first factor is caused by the restricted chemical space of higher activity molecules and is specific to the reward function. The second factor is caused by the nature of training and can be controlled.

To investigate how learning affects mode collapse, we ran several experiments where the generative model was trained with 25 iterations of policy gradient and one of 0, 20, 50, 100, 200, 500, or 1000 iterations of fine-tuning per epoch. We recorded valid fraction and active fraction after each epoch. The resulting trajectories are illustrated in Figure 3.15.

Figure 3.15a shows how active fraction, valid fraction, replay threshold, and average reward change with training for a different number of fine-tuning steps used in training. Figure 3.15b shows the joint trajectories of an active fraction and valid fraction change with training for the different number of fine-tuning steps. Figure 3.15 shows that when the model uses no fine-tuning, it fails to produce active molecules and maintains a high valid fraction. When the model uses fine-tuning, it learns to generate active molecules at the expense of a lower valid fraction. All runs with fine-tuning experienced a significant drop in a valid fraction in the first epoch of training. This drop may represent a transient phase when the model cannot generate active molecules and partially overfits to the initial molecules in the replay buffer. The decrease in the valid fraction is more pronounced in models that use more fine-tuning iterations, consistent with this proposal. Models with the fewest fine-tuning iterations have the lowest active fraction and the lowest valid fraction. Over model training, the active fraction is negatively correlated with the valid fraction, suggesting that the model suffers mode collapse as it learns to generate active molecules. Models with higher fine-tuning iterations have progressively higher active fractions and valid fractions. The model appears to increase valid fraction for the highest numbers tested (500 and 1000 iterations) as it learns. Although models with higher fine-tuning iterations initially experience a more considerable drop in valid fractions, they eventually have higher valid fractions than models with lower fine-tuning iterations. Similarly, we analyzed the effect of the different number of experience replay steps. Similar to the fine-tuning benchmark, the model with no experience replay fails to generate active molecules and maintains a high valid fraction. Inclusion of experience replay results in successful learning with a simultaneous decrease in valid fraction.



(a) Trajectory of training for different fine-tuning values.



(b) QED

Figure 3.15: Evolution of active and valid fractions over training.

Unlike the fine-tuning benchmark, however, the number of experience replay steps does not clearly affect model quality. In these experiments, model quality is largely determined by the presence or absence of experience replay steps.

Experience replay buffer effect. Finally, we investigated different initializations of the experience replay buffer. The experience replay library is typically filled with predicted molecules

generated by the model pretrained on the ChEMBL database, but our procedure enables us to use an arbitrary replay library alternatively. Due to sparse rewards, model learning is initially dictated by the replay library. We generated a second replay library with molecules from the Enamine kinase library, which consists of 65000 small molecules with predicted activity against kinases [106]. This library was chosen based on the expectation that general-purpose kinase inhibitors should contain scaffolds suitable for EGFR kinases.

We first selected molecules with non-zero active class probabilities for EGFR, as predicted by the random forest ensemble. We then filtered the active molecules to remove molecules with Bemis-Murcko scaffolds [53] present in the historical EGFR data. This step ensured that the replay buffer molecules were dissimilar from known molecules. The final Enamine replay library had 219 molecules.

This experiment tested three different replay libraries: an empty replay library (Empty buffer), the replay library from the model (Generated actives), and the Enamine library selected as above (Enamine). Figure 3.16 shows the 12 most common Bemis-Murcko scaffolds [53] in the generated libraries produced by each of the models. All scaffold calculations were done using the RDKit [56] package.

In the generated library produced with replay buffer initialized with compounds from the Enamine kinase library, the main quinazoline scaffold is notably absent. The Enamine-trained library suffers from lower diversity, likely because the initial replay buffer selected from the Enamine kinase library predominantly contains thiophene-fused rings. Such bias was introduced by the predictive model used to select the initial replay buffer. The predictive model favored compounds with thiophene-fused rings. This observation confirms that the initial selection of molecules in the replay library greatly influences the regions of chemical space that the model explores.

The library generated by the Empty buffer-trained model shows clear signs of overfitting, as 3 of the 12 most common scaffolds appear to be duplications of the quinazoline scaffold. The first active molecules greatly influence the model admitted into the replay library. When the replay library is initially empty, the model heavily exploits the first active molecules generated. As a result, the empty buffer-trained model explores a very limited region in chemical space (See

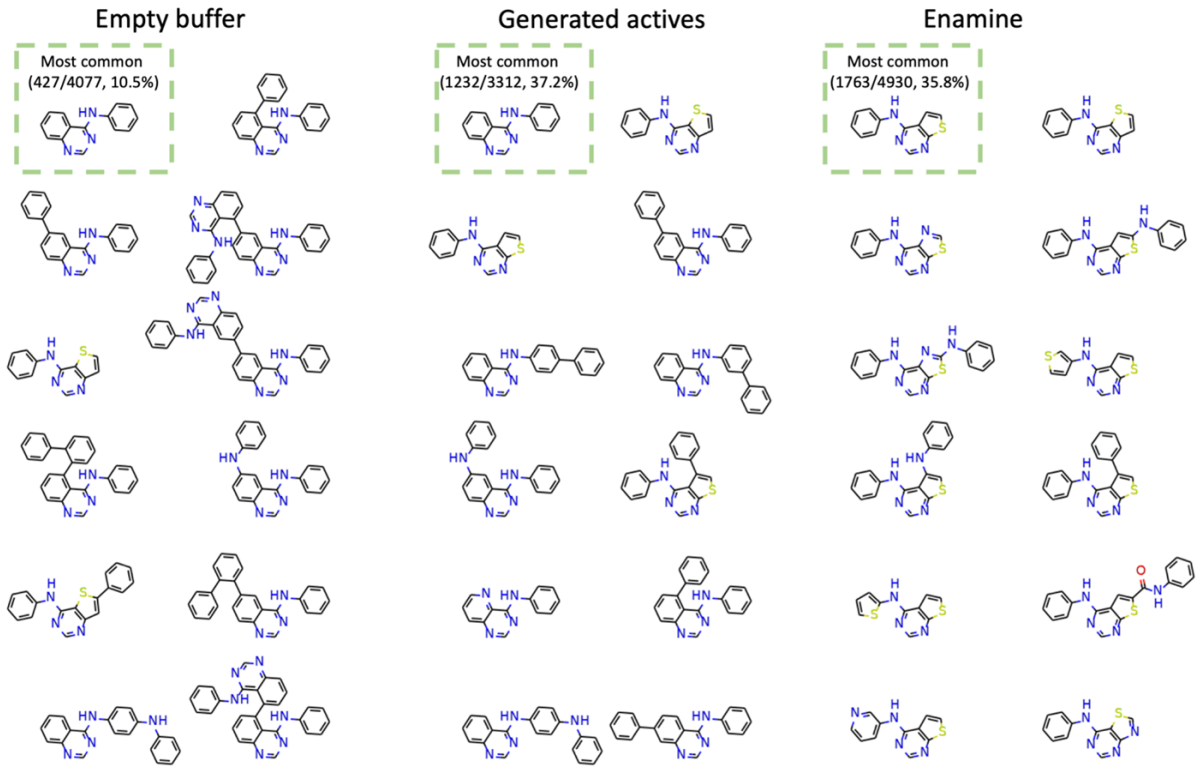


Figure 3.16: The 12 most common Bemis-Murcko scaffolds for models trained from different libraries.

Figure 3.17).

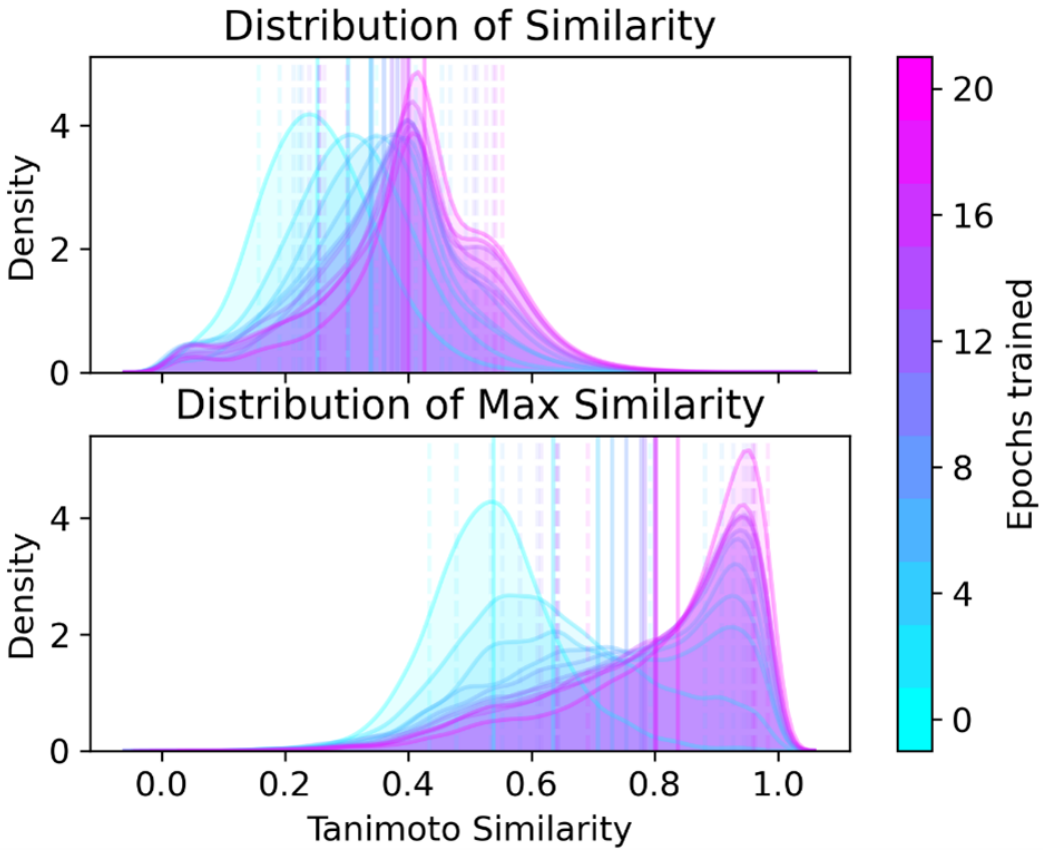


Figure 3.17: Distributions of Tanimoto similarities for libraries generated after different points in training.

Chapter 4

Applications of deep generative models for design of novel kinase inhibitors

4.1 Background

Discovery of chemical probes is a key step for unraveling the specific biological function of protein kinases. Such probes are small molecules that potently inhibit the catalytic activity of the target kinase, while minimally affecting the activity of the rest of the kinome [107, 108]. Kinase active sites are typically very similar as every kinase has an ATP binding site as part of the catalytic site; thus, small molecule inhibitors tend to bind non-specifically throughout the kinome. Balancing on-target potency with off-target selectivity when designing and optimizing kinase inhibitors is a considerable barrier to achieving complete coverage of the kinome with useful chemical probes. This makes designing chemical probes for all, but especially, for understudied kinases where chemical genomics data is absent, especially challenging. Solving this challenge is highly important as kinases play a critical role in health and disease [109].

Kinases represent a large group of enzymes that are implicated in many human diseases such as developmental and metabolic disorders and cancer. Characterization of kinase functions and kinase-oriented drug discovery is facilitated by chemical probes defined as “potent, selective and cell-permeable inhibitors of protein function” [110]. In terms of available chemical genomics data, kinases range from well-studied to understudied, or “dark”, kinases. The publicly accessible

[111] ChEMBL database [60], for instance, includes more than 1M bioactivity datapoints for more than $60K$ targets. Yet, there are nearly 150 kinases recognized as understudied meaning that there are no highly active compounds (and sometimes, no active compounds at all) known. Importantly, even with the wealth of chemogenomics data for well-studied kinases, so far chemical probes have been developed for 9 kinases [110] out of more than 500 functionally characterized kinase enzymes. Computational approaches have been sought after in order to accelerate the identification of kinase chemical probes. However, the use of these approaches is reliant on the existence of sufficiently large experimental data collections. Traditional molecular modeling approaches such as docking or QSAR require target-specific data, such as the knowledge of the three-dimensional structure of the target kinase or chemical bioactivity data for a sufficiently large set of molecules tested against a specific kinase. However, the historically relatively slow pace of kinase probe discovery suggests that traditional approaches have not been very successful in accelerating probe discovery even when wealth of data for well-characterized kinases is available. Naturally, when there is a dearth of experimental chemogenomics data, the efficacy and utility of computational approaches are significantly reduced. This dilemma of data is particularly pronounced in the case of understudied kinases, where, as the very name suggests, there is limited or no preexisting chemogenomics screening data or, with rare exception of orphan proteins, there are no available crystal structures for these targets. The problem of modeling of both relatively well-studied and understudied kinases thus demands innovative and revolutionary methodologies that break free from the current paradigm.

In this section, we address the challenging problem of identifying new kinase chemical probes by using novel computational methodologies that employ the artificial chemical intelligence methods described in Chapters 2 and 3. Specifically, we have chosen 4 kinase proteins as case studies – EGFR, CSNK2A2, DYRK1b and MKNK2. While EGFR is a relatively well-studied kinase with multiple FDA approved drugs and drug candidates in clinical trials, CSNK2A2, DYRK1b and MKNK2 are understudied kinases.

For the EGFR case study our main goal is to demonstrate that we can re-discover known active scaffolds using reliable QSAR models as a part of our reinforcement learning pipeline described in Chapter 3.

For the CSNK2A2, DYRK1b and MKNK2 case studies our goal is to discover novel hit molecules with previously unseen chemical structures that can serve as a starting points for developing chemical probes for these understudied kinases.

4.2 EGFR case-study

This section describes how models developed in section 3.2 wer used to generate molecules with maximized bioactivity for EGFR. I also covers selection and experimental validation of hit candidates.

4.2.1 Methods

Generative model. The generative model was pretrained using the ChEMBL dataset [60], which consists of approximately 2 million bioactive molecules. Notably, every molecule from ChEMBL has reported experimental bioactivity for at least one protein target. The pretraining step teaches the generative model to fit the distribution of molecules from the training data. Once pretrained, the generative network is used to sample new molecules from this distribution. Thus, we can assume that pretraining on a dataset of bioactive molecules such as ChEMBL ensures that the generative model will be capable of sampling bioactive-like molecules. This feature is essential to us since our ultimate goal is to produce active molecules to inhibit EGFR.

Activity data and predictive model. The predictive model was trained on historical experimental data of activities for EGFR extracted from ChEMBL. The EGFR training dataset includes bioactivities extracted from ChEMBL[60] (Target ID CHEMBL203). We considered only pChEMBL activities with a confidence score of 8 or greater for 'binding' or 'functional' human EGFR assays. Replicate compounds with bioactivity differences larger than one unit on a log scale were excluded. For similar replicate measurements, a single representative assay value was selected for inclusion in the training dataset. Activity values were binarized according to the $1\mu M$ cutoff. Chemical data were processed using the OpenEye chemistry toolkit. Standardizer was used for structure canonicalization, JChem 18.2, 2018, ChemAxon (http://www.chemaxon.com).

The dataset was curated according to a well-known protocol [112].

For the predictive model, we used an ensemble of five random forest (RF) classifiers. For features, we used 2,048-bit ECFP fingerprints as implemented in RDKit (https://www.rdkit.org/). We trained five random forest models on a cross-validated dataset to solve a binary classification problem. Each model in the ensemble returns the probability of class "active" for an input molecule. The resulting ensemble prediction is obtained by averaging predictions of all models in the ensemble. An interesting observation about this dataset is the presence of a privileged scaffold. Around 50% of active molecules that fall into the active class after binarization contain the quinazoline chemotype [113], a known hinge binder in kinase inhibitors [114]. From the crystal structures of know EGFR inhibitors, it is known that hydrophobic residues surround the quinazoline ring. The aniline group substituted at the 4 position of quinazoline ring and itself quinazoline ring of drugs like gefitinib and erlotinib are bounded by the hydrophobic pocket [115, 116]. With such a 4-anilinoquinazoline prevalence, we expect to see a bias in the predictive model's predictions towards this specific chemotype.

4.2.2 Results

4.2.3 Generation and selection of hit compounds.

With the information obtained through computational analysis described in section 3.2, we fixed the model training protocol. We trained the ChEMBL-pretrained model for 20 epochs, with 15 steps of policy gradient, 10 steps of experience replay, and 20 steps of fine-tuning by transfer learning per epoch. Every 2 epochs, we produced snapshot libraries of 16000 molecules. Each snapshot library included the distribution of active class probability for the generated molecules. Figure 4.1 illustrates the time-lapse of this distribution. The prominent peaks at 0 and 1 suggest that the model learns by increasing the fraction of highly active molecules, as opposed to generating molecules with progressively higher activities. This observation is likely because the random forest classifiers in the ensemble predictor were trained on the same dataset.

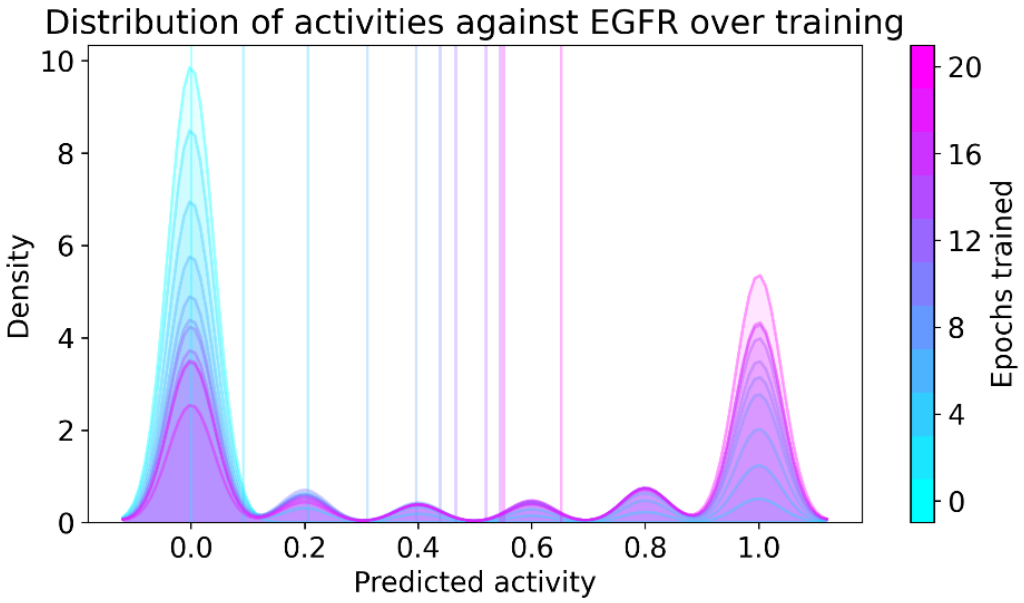


Figure 4.1: Time-lapse distribution of active class probability values during training.

4.2.3.1 Experimental Validation.

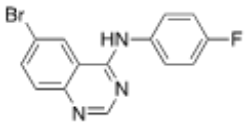
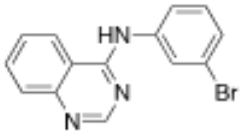
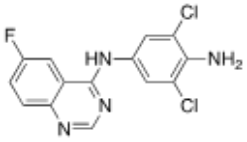
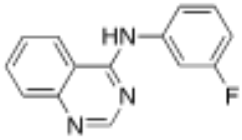
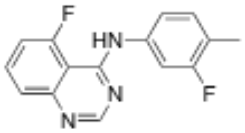
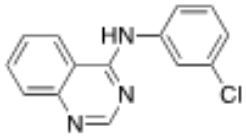
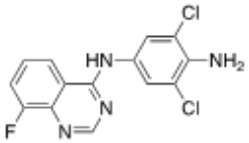
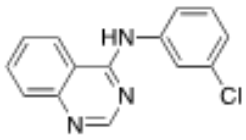
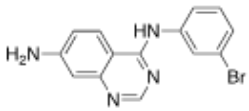
With few notable exceptions [86, 117], most of the current *de novo* design publications are purely computational. However, it is important to know how many computationally predicted candidates are experimentally validated by *in vitro* (at least) assays. For this test, we established the following screening protocol.

The model described in section 3.2 was used to generate a large library of novel computational hits with high active class probabilities. To enable rapid testing of the computational models all hit molecules were parsed through the Enamine REAL database (Release 2020q1-2, https://enamine.net/library-synthesis/real-compounds/real-database) of 1.36B on-demand commercially available molecules. The Enamine REAL (readily accessible) database is based on the synthesis of ultra-large chemical libraries using two- or three-step three-component reaction sequences and available starting materials with pre-validated (at least 80% synthesis success rate) chemical reactivity [118].

Seventeen computational hit molecules were matched with Enamine REAL. All of the predicted active compounds were derivatives of 4-anilinoquinazoline, a chemotype that was well represented in Enamine REAL. The predicted active compounds contained a few small substituents

on the quinazoline ring (positions 5–8: F, Cl, Br, OCH_3) but a wide range of substituents on the 4-anilino group. As a negative control, we selected five molecules predicted to be inactive but containing the same 4-anilinoquinazoline scaffold. The twenty two 4-anilinoquinazoline analogs were dissolved in DMSO and sent to Reaction Biology (https://www.reactionbiology.com/) for EGFR enzymatic assay screening. Two compounds in the predicted active series were insoluble in DMSO; therefore, biological tests were not performed. The 4-anilinoquinazoline analogs were initially tested in single-dose duplicate mode at a concentration of $1\mu M$ and percent inhibition relative to DMSO control was determined. Staurosporine was used as a reference EGFR tyrosine kinase inhibitor .

Table 4.1: Data for EGFR kinase inhibiton of compounds 1-4.

Catalog ID	pIC50	Structure	**Nearest neighbor (NN) from training**	**pChEMBL for NN)**
Z1192045732	7.5			7.6
Z1576525970	7.4			7.6
Z1182636554	6.7			7.3
Z1823625743	5.9			7.3
Most active from ChEMBL				10.0

Four 4-anilinoquinazolines from the predicted hit set showed $> 40\%$ inhibition of EGFR enzyme activity in the $1\mu M$ single dose assay (Table 4.1), while all five of the negative control analogs were inactive. Notably, the four active compounds contained only small substituents (Br, NH_2, CH_3) at the 4' position of the 4-anilino group (Table 4.1) paired with halogen

substitution on the 5, 6, or 8 positions of quinazoline core. Surprisingly, however, the 4'-fluroanilino-6-fluroquinazoline analog was not active. Notably, all of the analogs with large linear or branched substituents at the 4' position were inactive in the enzyme assay. The four active compounds from the single-dose assay were further tested in 10-dose IC50 mode with 3-fold serial dilution starting at $10\mu M$ to determine their EGFR inhibition potency. The 4-anilinoquinazolines 1 and 2 (Table 4.1) were potent EGFR inhibitors with $IC50 < 100nM$, comparable to the potency of staurosporine (Table 4.1). The 4-anilinoquinazolines 3 was slightly less potent with an $IC50 = 210nM$. Analog 4 was the least potent with $IC50 = 1.4\mu M$.

Each of the active compounds 1–4 had a 3'-halogen substituted 4-anilinoquinazoline as a close neighbor in the training set that was reported to have a similar EGFR inhibition potency (Table 1). The most potent EGFR inhibitor from ChEMBL was N-(3-bromophenyl)quinazoline-4,7-diamine (CHEMBL420624), which had activity at sub-nanomolar concentrations. Although all five out of the negative control compounds were inactive in the EGFR enzyme assay, it should be noted that they each contain large linear or branched substituents at the 4'-position of the aniline. Analogs with the same or similar substitution on the aniline that were selected to be active in the computational model were also shown to be inactive in the EGFR assay.

4.3 CSNK2A2, DYRK1B, MKNK2 case study

4.3.1 Virtual screening of AI-COSMOS library with ML models

Selection of hit candidates. In this section, we explored how the AI-COSMOS library can be used for practical purposes of finding new hit molecules for protein targets. We chose three kinase proteins of interest, namely CSNK2A2, MKNK2, and DYRK1b. We developed regression QSAR models for each protein to predict the potency of new molecules based on historical data from ChEMBL [60]. We used a concatenation of RDKit topological descriptors with maximum path sizes of 5, 7, 9, and 11 as features for the regression models. Each regression model is an ensemble of 5 XGBoost [119] models trained with a 5-fold cross-validation technique. Hyperparameters of the models were chosen with the Bayesian hyperparameter optimization technique [120]. The resulting models yielded R^2 of 0.76, 0.81, and 0.72 for CSNK2A2, MKNK2 and DYRK1b

respectively. Next, we used the trained models to virtually screen the AI-COSMOS library to find potential hits for the proteins targets of interest. We only screened 250 million molecules out of the 10 billion library for practical purposes. After that, we selected the top 1000 molecules with the highest predictions made by the QSAR models. To address the synthetic accessibility of the molecules and reduce the cost of this experiment, we ran a similarity search between our 1000 selected candidate molecules and Enamine REAL Space21 of 21 billion make-on-demand molecules using the Feature Trees algorithm as a part of FTrees software package from BioSolveIT [121]. Features Trees are special descriptors representing molecules as graphs with each ring and functional group reduced to a single node. Each node has a property profile that stores the physicochemical properties of that node. To calculate the similarity between two molecules, the algorithm overlays the corresponding Feature Trees to preserve the topology. The similarity between the two Feature Trees is a normalized sum of local Tanimoto distances between the corresponding property profiles in two molecules. We set up a similarity cutoff of 0.9 to identify molecules from Enamine REAL Space similar to the hits from the AI-COSMOS library. Next, we screened the molecules returned by the similarity search with QSAR models developed for virtual screening and filtered out molecules with predicted $pKd < 6.0$. Additionally, we trained a QSPR model for LogS. We used this model to filter out potentially insoluble molecules removing molecules with predicted $LogS > -5.0$. We also removed molecules that have Tanimoto similarity $T_sim > 0.75$ to any known hit from the dataset we used for training QSAR models. Finally, our goal was to select 50 molecules per kinase protein target for experimental validation. To identify a diverse subset of molecules, we used a spherical clustering algorithm to subdivide the pool of molecules into 50 clusters and then picked a molecule with the highest predicted pKd from each cluster.

Although with this approach, we chose not to synthesize molecules directly from the AI-COSMOS library due to high cost and time restrictions, the overall results would be hard to achieve without the AI-COSMOS library. Enamine REAL Space contains 21 billion molecules, which makes full virtual screening of this library an extensive computational task. Also, this experiment should serve as a proof-of-concept to justify custom synthesis of molecules from AI-COSMOS library in future studies.

Experimental validation of hit candidates. Selected molecules (50 molecules for each target) were purchased from Enamine REAL space which is available on-demand. However, not all molecules were successfully synthesized. Specifically, $48/50$, $26/50$ and $42/50$ molecules were successfully synthesized for CSNK2A2, DYRK1b and MKNK2 targets correspondingly.

Next, the remaining compounds were sent to Eurofins Scientific (https://www.eurofins.com/) for enzymatic assay screening for 3 selected kinases. The screening was performed in two steps. First, we measured enzymatic activity at a single protein concentration of $10\mu M$ for all available compounds. Second, for the hits that showed kinase residual enzymatic of $< 50\%$ in the single-dose assay we further performed a 10-dose IC50 experiment with 3-fold serial dilution starting at $10\mu M$ to determine their inhibition potency for the corresponding kinase.

Overall, $7/48$ compounds for CSNK2A2, $7/26$ compounds for DYKR1b and $9/42$ compounds for MKNK2 showed activity in a single-dose experiment. The corresponding IC50s together with the compounds' structures, structures of the nearest neighbors (NN) from the training set and $\%$ of residual activity in a single dose experiment are shown in Tables 4.2, 4.3 and 4.4.

Table 4.2: Data for CSNK2A2 kinase inhibition.

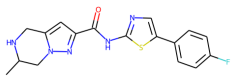
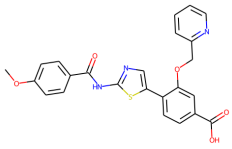
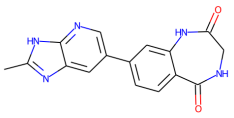
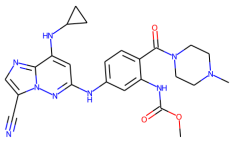
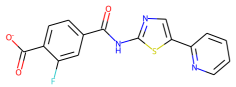
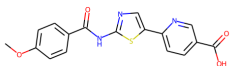
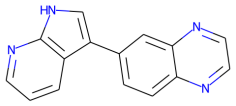
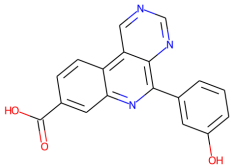
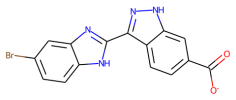
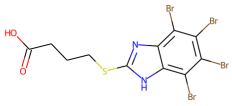
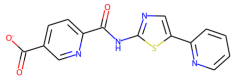
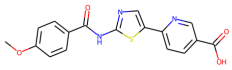
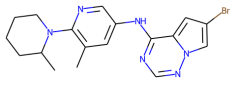
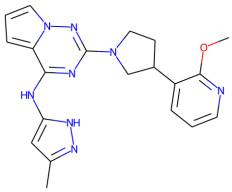
Catalog ID	% **residual activity**	Structure	**Nearest neighbor (NN) from training**	pIC50	**pIC50 for NN**
Z5499030000	23%			6.2	8.0
Z5499163489	33%			5.6	7.2
Z5499029829	10%			5.5	8.0
Z5499173970	43%			5.2	> 6.0
Z5499030009	42%			5.2	7.2
Z5499029808	44%			5.0	8.0
Z5499030299	-4%			> 4.5	7.4

Table 4.3: Data for DYRK1b kinase inhibition.

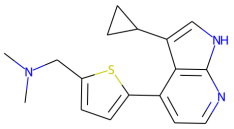
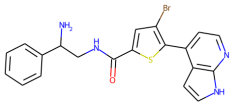
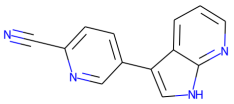
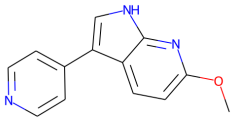
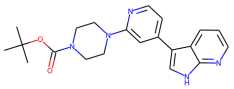
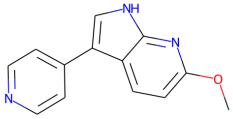
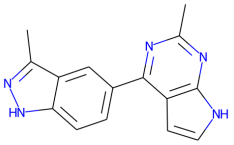
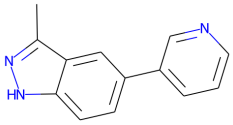
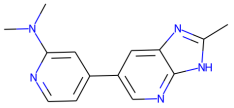
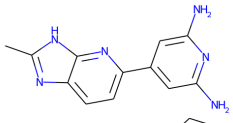
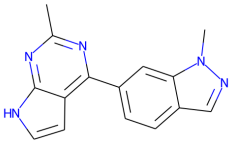
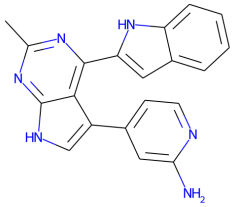
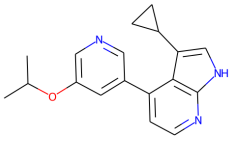
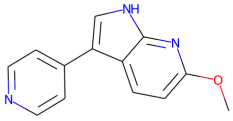
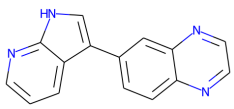
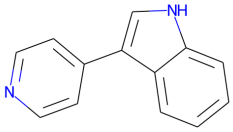
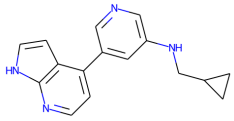
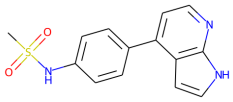
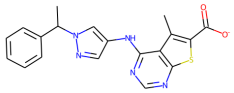
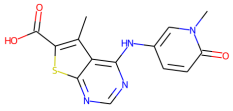
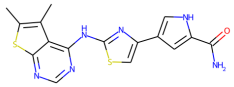
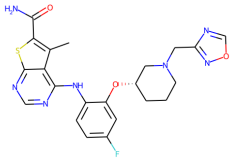
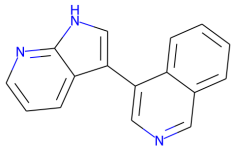
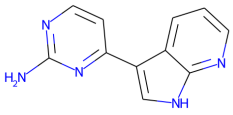
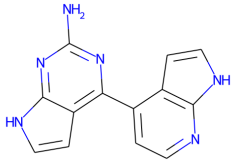
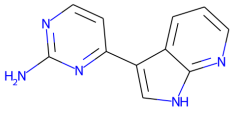
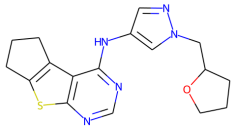
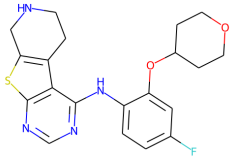
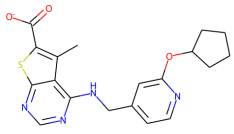
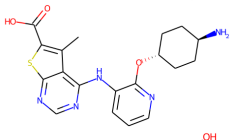
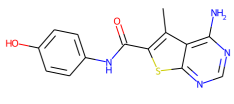
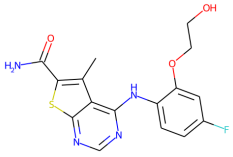
Catalog ID	% **residual activity**	Structure	**Nearest neighbor (NN) from training**	pIC50	**pIC50 for NN**
Z5499163408	17%			6.0	> 6.0
Z5499163447	18%			5.9	6.7
Z5499163493	16%			5.6	6.7
Z5499163516	13%			5.6	7.8
Z5499163504	33%			5.5	8.4
Z5499163514	37%			5.4	7.6
Z5499163463	47%			5.3	6.7

Table 4.4: Data for MKNK2 kinase inhibition.

Catalog ID	% **residual activity**	Structure	**Nearest neighbor (NN) from training**	pIC50	**pIC50 for NN**
Z5499173970	2%			6.3	6.3
Z3006785955	5%			6.2	> 6.0
Z5499173817	8%			6.1	6.0
Z650011466	18%			5.9	7.0
Z5499173972	22%			5.6	> 6.0
Z5499173979	17%			5.4	> 6.0
Z646116896	49%			5.1	7.0
Z5499173808	34%			5.1	8.0
Z5499173966	42%			4.9	7.6

Chapter 5

OpenChem: a Deep Learning toolkit for Computational Chemistry and Drug Design

5.1 Background

Deep learning is undergoing a rise in various fields of computational chemistry, including chemical reaction design, drug discovery, and material science. Machine learning has been a widely used technique in computational chemistry for the past 70 years, but mainly for building models for small molecule activity/property prediction from their chemical descriptors. Known as Quantitative Structure-Activity Relationship/Quantitative Structure-Property Relationship (QSAR/QSPR) [122], machine learning approaches have been applied to solving activity or property classification and regression problems. Both types of problems can be solved with deep learning models; however, until recently, chemical data was not big enough to train robust and reliable neural networks. Currently, computational chemistry is entering the age of big data. For instance, a repository of chemical bioactivity data known as Pubchem [123] comprises more than 100M chemical structures, and more than 250M experimentally measured bioactivities (https://pubchemdocs.ncbi.nlm.nih.gov/statistics). Thus, the use of deep neural networks to analyze big bioactivity data becomes increasingly justified. Moreover, there are many interesting classical problems in computational chemistry that have never been tackled with machine learning before the deep learning era. An excellent example of such a problem is the de-novo generation

of molecules with optimized properties. Previously, this problem was attacked with combinatorial methods [124, 125]. However, such an approach is not efficient since chemical space is big (with estimates up to 10^{60} molecules [62]), and an efficient sampling technique should be "smart", which makes deep learning models a good fit for this problem. Models for de-novo molecular design require a lot of unlabeled data, which is available at much less cost than labeled data. For example, the Enamine REAL database (https://enamine.net/hit-finding/compound-collections/real-database) contains 3.7 billion real organic molecules and can be used to train a deep generative neural network as to how to generate new realistic molecules. There have been various flavors of deep generative models for molecules in multiple representations, with most common ones being SMILES and molecular graphs. Examples of such model have been described in Chapter 2. There are also multiple property optimization strategies in the literature, such as reinforcement learning (described in Chapter 3), Bayesian optimization, and optimization in the latent space [32, 82, 126, 127].

From a practical perspective, there are two main frameworks used for developing deep learning neural networks, which are PyTorch [71] and Tensorflow [128]. We decided to use PyTorch since it is more suitable for easy experimentation and fast prototyping. In other words, we wanted to build a tool that would let scientists to quickly try an idea with as little engineering efforts as possible. Here comes another pitfall for computational chemists, who, in many cases, are not computer scientists or software engineers. Even with focused deep learning libraries such as PyTorch and Tensorflow, building a deep learning network does require extensive software engineering background. There exist several community-maintained libraries for computational chemistry, such as RDKit [56], DeepChem (https://deepchem.io) and ATOM Modeling Pipeline [129] which extends DeepChem. RDKit offers functionality for manipulating chemical objects such as atoms, bonds, and molecules. While this functionality is extremely useful for data processing, it is not designed for building machine learning models. Another library we mentioned, DeepChem, is aimed at building deep neural networks for chemistry and built upon Tensorflow. Developing a new model with DeepChem requires writing a lot of Tensorflow code; furthermore, DeepChem does not enable modular design features such as encapsulation and reusability of standard deep neural network blocks, such as encoders, decoders, embedding layers, etc.

Another critical question is the reproducibility of computational experiments. Frequently, results reported in papers cannot be reproduced by independent researchers. This could happen due to various factors, including the absence of standardized package environments, well-tracked log files, and protocols for reproducing the results, to name a few.

To address the issues discussed above, we developed OpenChem, a deep learning library for computational chemistry built upon PyTorch framework. OpenChem offers modular design, where building blocks can be combined, ease of use by letting the users define a model with a single configuration file, and advanced deep learning features such as built-in multi-GPU support. In this paper, we introduce OpenChem design and present three case studies. All data and models to reproduce these examples are available from https://github.com/Mariewelt/OpenChem.

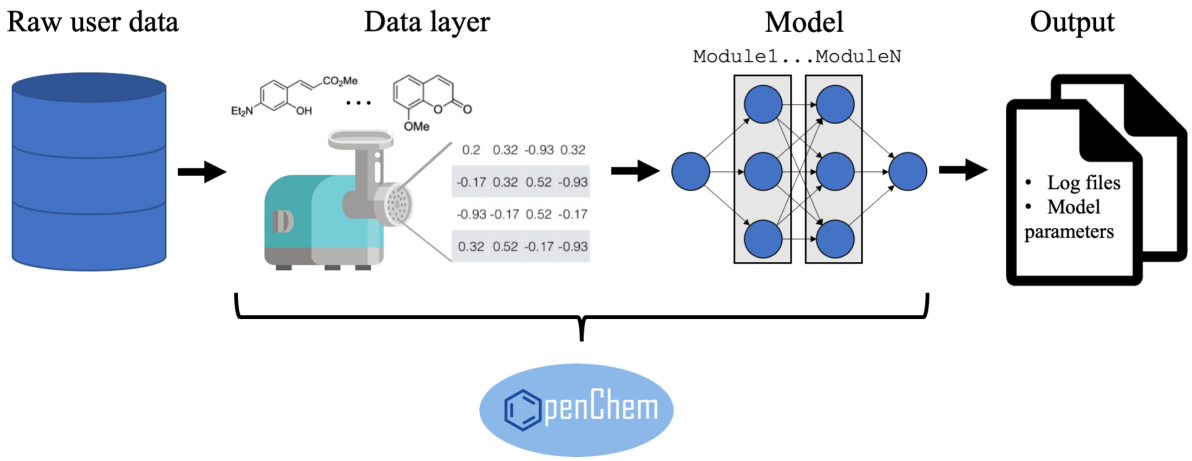


Figure 5.1: OpenChem pipeline.

5.2 Methods

5.2.1 OpenChem design

OpenChem is a deep learning toolkit for computational chemistry and drug design with PyTorch backend. The primary purpose of OpenChem is providing computational chemists with a tool for easy experimentation with deep learning models, i.e., quick implementation of architectures, fast

training, debugging, result interpretation and visualization, etc. The main idea is implementing a toolkit as a set of building blocks with a unified API that will be combined into a single custom architecture by a user.

OpenChem introduces several model types – *Feature2Label*, *Smiles2Label*, *Graph2Label*, *SiameseModel*, *GenerativeRNN*, *MolecularRNN* and *MolecularRNN3D*. These high-level model types consist of lower level modules, such as embeddings, encoders and multi-layer perceptron. Modules are built from layers, which could be PyTorch or custom layers. Examples of custom layers implemented in OpenChem are graph convolutions, convolutions combined with batch normalization and ReLU, and stack augmentation. Another OpenChem object type is a *Dataset*. OpenChem Dataset inherits from PyTorch Dataset and additionally provides features for converting inputs into tensors for OpenChem Model. OpenChem has dataset for converting SMILES into feature vectors, tokens and molecular graphs and for converting protein sequences into tokens.

Overall, OpenChem is implemented to offer users a modular design, i.e., blocks with the same input and output formats can be used interchangeably by adjusting the settings in the configuration file. For example, there are several different options for encoder block, such as RNN encoder, CNN encoder, or Graph CNN encoder, that could be used to calculate representation vectors for molecules. OpenChem allows choosing these options from the configuration file. OpenChem also supports built-in multi GPU training and offers several features for logging and debugging. Figure 5.2 summarizes the types of models, modules, and tasks that are currently implemented in OpenChem. Users can train predictive models for classification, regression, and multitask problems as well as develop generative models for producing novel molecules with optimized properties. OpenChem can work both with SMILES strings and molecular graphs. Data layers offer utilities for data preprocessing, such as converting SMILES strings to molecular graphs and calculating standard structural features for such graphs.

Models in OpenChem are defined in the Python configuration file as a dictionary of parameters. The dictionary must contain parameters that define how to run/train/evaluate a model as well as parameters defining model architecture. OpenChem provides scripts for model training, and it also natively supports multi-GPU training. After the training process is finished, OpenChem saves model parameters as well as log files to a designated folder, so that the experiment can be

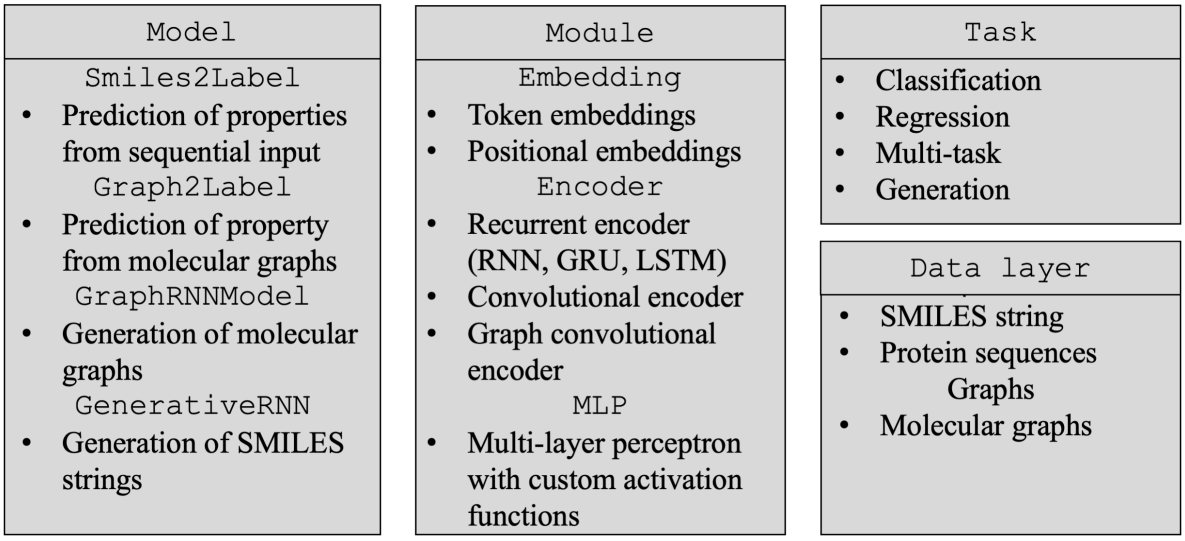


Figure 5.2: Main OpenChem objects.

reproduced later.

Configuration file . The configuration file must define a model, which should be any class derived from OpenChemModel and dictionary *model_params*, which specifies model hyperparameters. Detailed list of standard training parameters is provided in [100].

Below we consider three use cases, which will illustrate how models are defined and used in OpenChem.

5.3 Results

5.3.1 Case study 1: Graph Convolution Neural Network for predicting logP

Data and model description In this case study, we built a Graph Convolution Neural Network (GraphCNN) [93] for predicting the n-octanol/water partition coefficient (logP) from molecular graphs. This task is an example of a regression problem, as logP covers a range of continuous values. We used five atomic properties as node attributes – atom type, valence, charge, hybridization, and aromaticity. OpenChem provides a module for declaring an attribute, where a user can specify attribute type (node or edge), attribute label (e.g., categorical), list all possible values

for categorical attributes, etc. The GraphCNN model also requires a user-defined function for calculating node attributes. It is a Python function that receives the RDKit atom object as an input and returns a dictionary of atomic attributes for the input atom.

First, we loaded the logP dataset (described below) and split it into train and test subsets. OpenChem provides function *read_smiles_property_file*, that accepts path to the file and indices of columns to be read from the file as arguments and returns a list where each element is a column from the file. In this example, we loaded *logP_labels.csv* file, read columns number 1 and 2, split the data into training and test subsets and saved the subsets into new files using *save_smiles_property_file* utility from OpenChem. Next, we created graph data layers from the saved files with train and test subsets. OpenChem provides *GraphDataset* class, which can convert SMILES strings to molecular graphs and calculate node and edge attributes. *GraphDataset* also accepts a user-defined dictionary of atomic (node) attributes, such as valency, hybridization, aromaticity, etc., and functions for computing these attributes. Other parameters to *GraphDataset* include a path to the text file with data, a list of columns that will be read from the file, and column delimiter.

Our model consists of 5 layers of Graph Convolutions with the hidden size of 128, followed by 2 layers of multilayer perceptron (MLP) with ReLU nonlinearity and hidden dimensionalities of 128 and 1. We trained the model for 100 epochs with Adam 18 optimizer with initial learning rate of 0.01, and *MultiStepLR* learning scheduler with step size 15, and gamma factor of 0.5, meaning that the learning rate is decreased by half every 15 epochs of training starting from the initial learning rate of 0.01. For external evaluation, we used R^2 score. We also printed the intermediate progress report on the training and evaluation metrics every 10 epochs and saved the model checkpoint every 5 epochs. We specified all these parameters in the dictionary in Figure 5.3.

Results. We trained a GraphCNN for predicting the n-octanol/water partition coefficient (logP) directly from the molecular graph. The modeling dataset of $14.5K$ molecules was obtained based on the public version of the PHYSPROP database [130]. The dataset was curated according to our well-established protocol [112] for structural standardization, the cleaning of salts, and the

```
model = Graph2Label

model_params = {
  'task': 'regression',
  'random_seed': 42,
  'use_clip_grad': False,
  'batch_size': 256,
  'num_epochs': 101,
  'logdir': 'logs/logp_gcnn_logs',
  'print_every': 10,
  'save_every': 5,
  'train_data_layer': train_dataset,
  'val_data_layer': test_dataset,
  'eval_metrics': r2_score,
  'criterion': nn.MSELoss(),
  'optimizer': Adam,
  'optimizer_params': {
    'lr': 0.0005,
  },
  'lr_scheduler': StepLR,
  'lr_scheduler_params': {
    'step_size': 15,
    'gamma': 0.8
  },
  'encoder': GraphCNNEncoder,
  'encoder_params': {
    'input_size': train_dataset[0]["node_feature_matrix"].shape[1],
    'encoder_dim': 128,
    'n_layers': 5,
    'hidden_size': [128]*5,
  },
  'mlp': OpenChemMLP,
  'mlp_params': {
    'input_size': 128,
    'n_layers': 2,
    'hidden_size': [128, 1],
    'activation': [F.relu, identity]
  }
}
```

Figure 5.3: OpenChem case study 1 config examples.

removal of mixtures, inorganics, and organometallics was performed using ChemAxon. In the case of replicate compounds, InChI Keys were generated to resolve duplicates. In case of conflicting

property values, the entries were discarded. Using five-fold cross-validation, we obtained the model accuracy expressed as $R^2 = 0.90$ and root mean square error $RMSE = 0.56$. This is significantly better than traditional QSPR models ($R^2 = 0.86$ and $RMSE = 0.78$.) obtained on the same dataset using physicochemical descriptors and E-state indices [131].

5.3.2 Case study 2: Tox21 Challenge

Data and model description. In this case study, we built a Recurrent Neural Network (RNN) (see Fig 5.4). for multitask prediction of bioactivity for 12 receptors using data from the Tox21 challenge [132]. This model receives SMILES string 6 for a ligand as an input and returns a vector of size 12, where each component is interpreted as a probability that the input ligand binds to a corresponding receptor. In other words, multitask allows solving 12 independent binary classification problems with a single model. Since any SMILES string is a sequence of characters, we can use Recurrent Neural Networks to calculate a representation vector for a given molecule [32]. Each symbol of the SMILES string s_t is processed sequentially. At each time step, the model takes a single character s_t from the SMILES string, converts it to a numerical embedding vector x_t with a learnable embedding dictionary layer. Then x_t is passed to the LSTM layer:

$$h_{t+1} = W_x x_t + b_x + W_h h_t + b_h,$$

where h_t is the intermediate hidden state for the SMILES prefix of length t. When the whole SMILES string is processed, hidden state h_T from the final time step is used as a representation vector for the next feed-forward layer, which outputs the vector of prediction for the input SMILES.

Defining model in OpenChem. Tox21 dataset is available as a benchmark dataset, and it can be loaded from the OpenChem GitHub with *read_smiles_property_file* function. As Tox21 is a multi-target dataset, some of the labels are not available and, therefore, left empty. To account for dummy labels, we used *MultitaskLoss* from OpenChem, which is a binary cross-entropy loss, averaged across multiple classes. *MultitaskLoss* also does not accumulate losses for dummy labels to the final loss in backpropagation. We filled them with a dummy index, that was ignored during

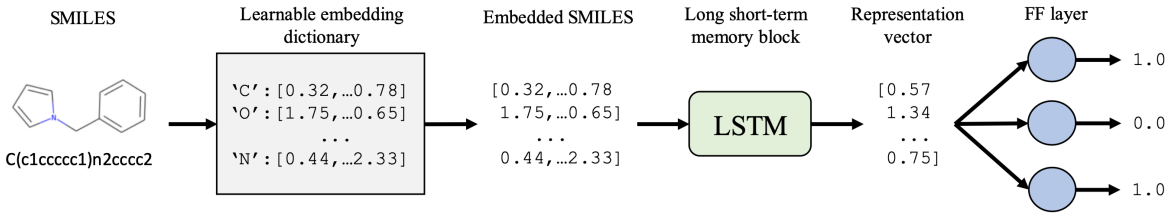


Figure 5.4: Scheme of multitask SMILES2Label model. Input SMILES string is converted to a matrix of embeddings by the dictionary of learnable embeddings. Next, the matrix of embeddings is passed to the RNN encoder with LSTM layer. The RNN encoder converts the matrix of embedding to a representation vector, which is used by the multilayer perceptron to make predictions for the input SMILES.

training. We also extracted unique tokens from the whole dataset before splitting it into train and test to avoid the situation when some of the tokens are not present in one of the pieces of the dataset. After this step, we split data randomly into training and test and saved these subsets to new files with OpenChem *save_smiles_property_file* utility.

Next, we created the SMILES data layer from input files and added data augmentation by SMILES enumeration [133] for training dataset. The idea behind it is to include non-canonical notation for SMILES. Augmentation is enabled by setting the argument *augment=True* when creating an object of class *SmilesDataset*. Since this task is multi-target, we needed to implement a custom evaluation function for calculating classification accuracy separately for each task. As for accuracy metrics, we used the AUC-ROC averaged across all classes. Next, we defined the model architecture with *Smiles2Label* modality. This model consists of Embedding block, Recurrent Encoder with 4 LSTM layers, and MLP. We also used dropout with a high probability to enable regularization to avoid model overfitting. The configuration dictionary is shown in Figure 5.5.

Results. In this example, we trained a multitask model for predicting biological activity for 12 assays from the Tox21 challenge. The similarity (and dissimilarity) between the tasks is exploited to enrich a model [134]. We obtained the mean AUC of 0.84 with the following per target AUC values on the test set:

- NR-AR 0.85
- NR-AR-LBD 0.90

```
model = Smiles2Label

model_params = {
  'task': 'multitask',
   'criterion': MultitaskLoss(ignore_index=9, n_tasks=12).cuda(),
        ...
  'embedding': Embedding,
  'embedding_params': {
    'num_embeddings': train_dataset.num_tokens,
    'embedding_dim': 128,
    'padding_idx': train_dataset.tokens.index(' ')
  },
  'encoder': RNNEncoder,
  'encoder_params': {
    'input_size': 128,
    'layer': "LSTM",
    'encoder_dim': 128,
    'n_layers': 4,
    'dropout': 0.8,
    'is_bidirectional': False
  },
  'mlp': OpenChemMLP,
  'mlp_params': {
    'input_size': 128,
    'n_layers': 2,
    'hidden_size': [128, 12],
    'activation': [F.relu, torch.sigmoid],
    'dropout': 0.0
  }
}
```

Figure 5.5: OpenChem case study 2 config examples.

- NR-AhR 0.87
- NR-Aromatase 0.84
- NR-ER 0.76
- NR-ER-LBD 0.82
- NR-PPAR-gamma 0.80
- SR-ARE 0.78
- SR-ATAD5 0.85

- SR-HSE 0.84
- SR-MMP 0.87
- SR-p53 0.86

These results are comparable to the results reported in the literature previously [132]. Our single multitask model approached the accuracy of the winning model [135], which was a complex ensemble combination of different models and descriptor schemes.

5.3.3 Case study 3: Generation of molecular graphs with maximized melting temperature

Data and model description. In the final case study, we built a MolecularRNN model described in 2.3 for the generation of molecular graphs and further optimization of the specific property of the generated molecules. This model generates molecular graphs in an auto-regressive manner, i.e., atom by atom. At each time step, the model predicts the chemical type of a new atom and the edge connections (including edge types) between the new atom and previously generated ones. Here we only briefly mention the overall training pipeline. Please see the full example at https://github.com/Mariewelt/OpenChem and detailed model description in 2.3.

Defining the model in OpenChem. The MolecularRNN model was pretrained on the curated ChEMBL dataset [60] of $1.5M$ molecules. We performed optimization of model parameters to maximize the melting temperature of the generated molecular graphs. The MolecularRNN model had 4 GRU [47] layers with 256 hidden activations in both *NodeRNN* and *EdgeRNN*. The model learns a dictionary of embeddings for 9 atoms types (C, N, O, F, P, S, Cl, Br, I) and three bond types (single, double, and triple). We used the kekulized form of molecules according to RDKit [56] which eliminated the need to specify aromatic bonds explicitly. We also defined structural parameters of the molecular graphs, such as minimum and the maximum number of nodes and valency constraints for each atom type.

The configuration dictionary is shown in Figure 5.6.

Results. Currently, common modeling frameworks do not allow simultaneous ML model building, new compound generation, and property optimization. OpenChem library bridges this gap. Briefly, to optimize melting temperature (T_{melt}), we started with our pretrained model and used the policy gradient algorithm. We trained a GraphCNN regression model to predict T_{melt}. The model has RMS error of 39.5 that is comparable to the state-of-the-art for the same dataset of 54k molecules [94]. See section 2.3.1.2 complete technical details.

```
model = GraphRNNModel
model_params = {
  'task': graph_generation,
          ....
  'num_node_classes': num_node_classes,
  'num_edge_classes': num_edge_classes,
  'max_num_nodes': max_num_nodes,
  'start_node_label': start_node_label,
  'max_prev_nodes': max_pred_nodes,
  'label2atom': label2atom,
  'edge2type': edge2type,
  'restrict_min_atoms': restrict_min_atoms,
  'restrict_max_atoms': restrict_max_atoms,
  'max_atom_bonds': max_atom_bonds,
  'EdgeEmbedding': Embedding,
  'edge_embedding_params': {
    'num_embeddings': num_edge_classes,
    'embedding_dim': edge_embedding_dim,
  }
  'NodeEmbedding': Embedding,
  'node_embedding_params': {
    'num_embeddings': num_node_classes,
    'embedding_dim': node_embedding_dim,
  }
  'NodeMLP': OpemChemMLPSimple,
  'node_mlp_params': {
    'input_size': 128,
    'n_layers': 2,
    'hidden_size': [128, num_node_classes],
    'activation': [nn.ReLU(inplace=True),
                   identity],
    'init': "xavier_uniform",
    }
  'NodeRNN': GRUPlain,
  'node_rnn_params': {
    'input_size': node_rnn_input_size,
    'embedding_size': 128,
    'hidden_size': 256,
    'num_layers': 4,
    'has_input': True,
    'has_output': True,
    'output_size': 128,
    'has_output_nonlin': False
    }
  'EdgeRNN': GRUPlain,
  'edge_rnn_params': {
    'input_size': edge_embedding_dim,
    'embedding_size': 64,
    'hidden_size': 128,
    'num_layers': 4,
    'has_input': True,
    'has_output': True,
    'output_size': num_edge_classes,
    }
  }
}
```

Figure 5.6: OpenChem case study 3 config examples.

Chapter 6

Conclusion

In this thesis, we have created and implemented multiple deep generative neural networks for generation of molecules and a deep RL approach termed ReLeaSE for de novo design of novel chemical compounds with desired properties.

We proposed a SMILES-based generative model for molecules with an augmented memory stack. In a valid SMILES molecule, in addition to correct valence for all atoms, one must count ring opening and closure, as well as bracket sequences with several bracket types. Therefore, only memory-augmented neural networks such as Stack-RNN or Neural Turing Machines are the appropriate choice for modeling these sequence dependencies. We demonstrated that the use of augmented memory stack improves quality of the generated SMILES strings.

We then tried using a SMILES-based generative model in a ReLeaSE pipeline. As a proof of principle, we tested our approach on three diverse types of end points: physical properties, biological activity, and chemical substructure bias. The use of flexible reward function enables different library optimization strategies where one can minimize, maximize, or impose a desired range to a property of interest in the generated compound libraries. As a by-product of these case studies, we have generated a data set of more than 1M novel compounds. Here, we have focused on presenting the new methodology and its application for initial hit generation. However, ReLeaSE could also be used for lead optimization, where a particular privileged scaffold is fixed and only substituents are optimized. Our future studies will explore this direction.

We also proposed MolecularRNN, the model for generating realistic molecular graphs. Molec-

ularRNN learns diverse distributions through unsupervised pretraining, generating 100% valid molecules in inference, while still receiving negative feedback from invalid ones during training. Combined with policy gradient optimization, MolecularRNN solves the problem of generating molecules with desired properties. We demonstrate that MolecularRNN pretrained on vast unlabeled datasets learns diverse and realistic distributions over the space of molecular graphs. With valency-based rejection sampling our model produces 100% valid molecules, and with structural penalty we train the model to avoid making mistakes by providing the model a feedback from negative samples. Optimized MolecularRNN outperforms other state-of-the-art methods on the benchmark tasks.

Furthermore, we coupled MolecularRNN with the predictive model as a critic in a ReLeaSE pipeline to optimize melting temperature, a property that cannot be calculated directly from a molecular graph. In further studies we plan address problems of multi-objective property optimization and completion of a molecular graph from a given scaffold.

Additionally, we extended MolecularRNN to not only generate 2D molecular graphs, but also 3D conformations. MolecularRNN3D version is capable of generating molecules in 3D space from scratch, or just computing a conformation for a give 2D molecule. We demonstrated, that generated conformations are realistic and close to the ones that can be obtained with a more computationally extensive algorithms.

Furthemore, we showed how deep generative models for molecules coupled with property optimization can be used to design novel hit molecules for kinase proteins. Herein, we proposed several new improvements to the heuristics used to optimize properties of molecules created by generative neural networks with reinforcement learning and sparse rewards. Sparse rewards are commonly observed when maximizing the bioactivity of generated molecules for a specific target protein. Thus, classic reinforcement learning algorithms such as policy gradient or Q-learning are not sufficient for such tasks. In contrast, our proposed tweaks, i.e., fine-tuning with transfer learning, experience replay, and real-time reward shaping, aim to extract informative feedback from the sparse reward signal and keep a healthy balance between exploration and exploitation. As a result of our study, we came up with a list of crucial points to consider when optimizing generative models with reinforcement learning.

1. We recommend considering the sparsity of the rewards and the desired level of balance between exploration and exploitation when selecting the right strategy for performing optimization in each case. The real-time reward shaping can be helpful in a sparse rewards scenario while unnecessary in cases when the reward feedback is sufficient (such as QED or LogP optimization).
2. the fine-tuning by transfer learning achieves a high level of exploitation, especially when used with known molecules. However, it will unlikely discover any chemotypes beyond the ones used for training.
3. the experience replay requires a rich and diverse pool of experience trajectories. Otherwise, this technique may also result in over-exploitation of replay examples. However, it can be a powerful tool to explore the chemical space and deal with sparse rewards in tandem with a policy gradient.

The optimized protocol was subject to a blind experimental validation. Out of fifteen tested compounds that were predicted active, four were confirmed in an EGFR enzyme assay. Two out of four compounds had nanomolar EGFR inhibition activity comparable to that of staurosporine. The overall hit rate was 27%. Additionally, five compounds with the same scaffold as active compounds but predicted as inactive were used as a negative control. All five compounds were confirmed as inactive. The obtained hit rate is on par with traditional virtual screening projects where molecule selection is guided by an expert medicinal chemist. However, in this work, we show that a properly trained AI model can mimic medicinal chemists' skills in the autonomous generation of new chemical entities (NCEs) and selection of molecules for experimental validation. This is a prime example of the transfer of the decision power from human experts to AI. Such capabilities could be an important step toward true self-driving laboratories and serve as an example of the synergy between machine and human intelligence. In summary, we do not think there is a current universal recipe for optimizing the properties of generated molecules with reinforcement learning. Each task is unique and requires thorough reward function engineering and hyperparameter search. However, as we have demonstrated with the EGFR inhibitor design example, with the right choice of the training protocol, generative models can be a powerful technique for automated and inexpensive de novo molecular design that can be executed even

with limited computational and financial resources.

Additionally, we gave a lower bound for the size of chemical space using empirical estimation with deep generative models. We generated a library of virtual molecules called AI-COSMOS consisting of 10 billion drug-like molecules. Computational library design methods are often criticized for their inability to control synthetic accessibility of de novo–generated molecules (13). Computationally generated compounds are often quite complex; for instance, they may include exotic substituents. In many cases, these compounds may require multistep custom syntheses or could even be synthetically inaccessible with the current level of technology. In the pharmaceutical industry, the ease of synthesis of a prospective hit molecule is of primary concern as it strongly affects the cost of the manufacturing process required for the industrial-scale production. In this experiment we used SAS [54] and IBM's retrosynthetic tool RXN4Chemistry [67] to assess the synthetic accessibility of the de-novo generated compounds and compare it to the combinatorial libraries such as Enamine REAL. This analysis shows that synthetic accessibility of the generated molecules does not differ much from the one of the aforementioned libraries. We further used AI-COSMOS library to find novel hit molecules for 3 understudied kinases – CSNK2A2, DYRK1b and MKNK2.

Finally, all the algorithms and methods developed in this thesis are presented in a form of toolkit called OpenChem. Deep learning methods emerged as a powerful approach for a variety of different tasks, including predictive, discriminative, and generative models. The OpenChem library was created to enable high-performance implementations of deep learning algorithms for drug discovery and molecular modeling applications. Built upon PyTorch framework, OpenChem is optimized for execution on GPUs and large datasets. One could quickly train ML models from datasets with hundreds of thousands or even millions of data points. OpenChem's modular API allows easy experimentation and fast model prototyping without substantial programming effort. Calculations with OpenChem could be scaled in the Cloud and HPC clusters. It provides well-tracked log files and sharable protocols and models for reproducible results. We described just three examples of practical tasks that can be solved with OpenChem. However, the functionality of the proposed framework includes a wide variety of tasks, covering binary, multiclass, multitask classification, regression, generative modeling, and property optimization.

In all three demonstrated examples, we quickly obtained a state-of-the-art performance of models without extensive programming of each model. Our plans include extending the model list and adding new tasks such as message passing neural networks and multi-property optimization with reinforcement learning.

Bibliography

[1] Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. In *Advances in Neural Information Processing Systems*, pages 6410–6421, 2018. (document), 2.1, 2.3.2.1, 2.3, 3.1.1, 3.1.3.3, 3.2, 3.1.3.3

[2] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. *arXiv preprint arXiv:1802.04364*, 2018. (document), 2.1, 2.3.2.1, 2.3, 3.1.1, 3.1.3.3, 3.1.3.3, 3.2

[3] Jack W Scannell, Alex Blanckley, Helen Boldon, and Brian Warrington. Diagnosing the decline in pharmaceutical r&d efficiency. *Nature reviews Drug discovery*, 11(3):191–200, 2012. 1.1

[4] Steven M Paul, Daniel S Mytelka, Christopher T Dunwiddie, Charles C Persinger, Bernard H Munos, Stacy R Lindborg, and Aaron L Schacht. How to improve r&d productivity: the pharmaceutical industry's grand challenge. *Nature reviews Drug discovery*, 9(3): 203–214, 2010. 1.1

[5] Jayme L Dahlin and Michael A Walters. How to triage pains-full research. *Assay and drug development technologies*, 14(3):168–174, 2016. 1.1

[6] Stephen J Capuzzi, Eugene N Muratov, and Alexander Tropsha. Phantom pains: Problems with the utility of alerts for p an-a ssay in terference compound s. *Journal of chemical information and modeling*, 57(3):417–427, 2017. 1.1

[7] Vinicius M Alves, Eugene N Muratov, Stephen J Capuzzi, Regina Politi, Yen Low, Rodolpho C Braga, Alexey V Zakharov, Alexander Sedykh, Elena Mokshyna, Sherif

Farag, et al. Alarms about structural alerts. *Green Chemistry*, 18(16):4348–4360, 2016. 1.1

[8] Yolanda Gil, Mark Greaves, James Hendler, and Haym Hirsh. Amplify scientific discovery with artificial intelligence. *Science*, 346(6206):171–172, 2014. 1.1

[9] Chayakrit Krittanawong, HongJu Zhang, Zhen Wang, Mehmet Aydar, and Takeshi Kitai. Artificial intelligence in precision cardiovascular medicine. *Journal of the American College of Cardiology*, 69(21):2657–2664, 2017. 1.1

[10] Katie Chockley and Ezekiel Emanuel. The end of radiology? three threats to the future practice of radiology. *Journal of the American College of Radiology*, 13(12):1415–1420, 2016. 1.1

[11] Han Altae-Tran, Bharath Ramsundar, Aneesh S Pappu, and Vijay Pande. Low data drug discovery with one-shot learning. *ACS central science*, 3(4):283–293, 2017. 1.1

[12] Erik Gawehn, Jan A Hiss, and Gisbert Schneider. Deep learning in drug discovery. *Molecular informatics*, 35(1):3–14, 2016. 1.1

[13] Matthew Ragoza, Joshua Hochuli, Elisa Idrobo, Jocelyn Sunseri, and David Ryan Koes. Protein–ligand scoring with convolutional neural networks. *Journal of chemical information and modeling*, 57(4):942–957, 2017. 1.1

[14] Alexander Aliper, Sergey Plis, Artem Artemov, Alvaro Ulloa, Polina Mamoshina, and Alex Zhavoronkov. Deep learning applications for predicting pharmacological properties of drugs and drug repurposing using transcriptomic data. *Molecular pharmaceutics*, 13(7): 2524–2530, 2016. 1.1

[15] Marwin HS Segler and Mark P Waller. Modelling chemical reasoning to predict and invent reactions. *Chemistry–A European Journal*, 23(25):6118–6128, 2017. 1.1

[16] Justin S Smith, Olexandr Isayev, and Adrian E Roitberg. Ani-1: an extensible neural network potential with dft accuracy at force field computational cost. *Chemical science*, 8 (4):3192–3203, 2017. 1.1, 2.4.1

[17] Kristof T Schütt, Farhad Arbabzadah, Stefan Chmiela, Klaus R Müller, and Alexandre Tkatchenko. Quantum-chemical insights from deep tensor neural networks. *Nature*

communications, 8(1):1–8, 2017. 1.1

[18] Volker Schnecke and Jonas Boström. Computational chemistry-driven decision making in lead generation. *Drug discovery today*, 11(1-2):43–50, 2006. 1.1

[19] Ricardo Macarron. Critical review of the role of hts in drug discovery. *Drug discovery today*, 7(11):277–279, 2006. 1.1

[20] Gisbert Schneider and Uli Fechner. Computer-based de novo design of drug-like molecules. *Nature Reviews Drug Discovery*, 4(8):649, 2005. 1.1

[21] Harald Mauser and Wolfgang Guba. Recent developments in de novo design and scaffold hopping. *Current opinion in drug discovery & development*, 11(3):365–374, 2008. 1.1

[22] Benjamin Sanchez-Lengeling, Carlos Outeiral, Gabriel L Guimaraes, and Alan Aspuru-Guzik. Optimizing distributions over molecular space. an objective-reinforced generative adversarial network for inverse-design chemistry (organic). 2017. 1.1

[23] Pavel G Polishchuk, Timur I Madzhidov, and Alexandre Varnek. Estimation of the size of drug-like chemical space based on gdb-17 data. *Journal of computer-aided molecular design*, 27(8):675–679, 2013. 1.1, 2.5.1

[24] Jérémy Besnard, Gian Filippo Ruda, Vincent Setola, Keren Abecassis, Ramona M Rodriguiz, Xi-Ping Huang, Suzanne Norval, Maria F Sassano, Antony I Shin, Lauren A Webster, et al. Automated design of ligands to polypharmacological profiles. *Nature*, 492 (7428):215–220, 2012. 1.1

[25] Daniel Reker, Petra Schneider, and Gisbert Schneider. Multi-objective active machine learning rapidly improves structure–activity models and reveals new protein–protein interaction inhibitors. *Chemical science*, 7(6):3919–3927, 2016. 1.1

[26] Christopher Lipinski and Andrew Hopkins. Navigating chemical space for biology and medicine. *Nature*, 432(7019):855–861, 2004. 1.1

[27] Nathan Brown, Ben McKay, François Gilardoni, and Johann Gasteiger. A graph-based genetic algorithm and its application to the multiobjective evolution of median molecules. *Journal of chemical information and computer sciences*, 44(3):1079–1087, 2004. 1.1

[28] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018. 1.1, 2.2.2.1, 3.1.1

[29] Marwin HS Segler, Thierry Kogej, Christian Tyrchan, and Mark P Waller. Generating focused molecule libraries for drug discovery with recurrent neural networks. *ACS central science*, 4(1):120–131, 2017. 1.1, 3.1.1

[30] Artur Kadurin, Alexander Aliper, Andrey Kazennov, Polina Mamoshina, Quentin Vanhaelen, Kuzma Khrabrov, and Alex Zhavoronkov. The cornucopia of meaningful leads: Applying deep adversarial autoencoders for new molecule development in oncology. *Oncotarget*, 8(7):10883, 2017. 1.1

[31] David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36, 1988. 2.1, 3.1.1, 3.1.2.1, 3.2.2

[32] Mariya Popova, Olexandr Isayev, and Alexander Tropsha. Deep reinforcement learning for de novo drug design. *Science advances*, 4(7):eaap7885, 2018. 2.1, 3.1.1, 5.1, 5.3.2

[33] Francesca Grisoni, Michael Moret, Robin Lingwood, and Gisbert Schneider. Bidirectional molecule generation with recurrent neural networks. *Journal of chemical information and modeling*, 60(3):1175–1183, 2020. 2.1

[34] Marcus Olivecrona, Thomas Blaschke, Ola Engkvist, and Hongming Chen. Molecular de-novo design through deep reinforcement learning. *Journal of cheminformatics*, 9(1):48, 2017. 2.1, 2.2.2.1, 3.1.1

[35] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018. 2.1, 2.3.2.1

[36] Qi Liu, Miltiadis Allamanis, Marc Brockschmidt, and Alexander Gaunt. Constrained graph variational autoencoders for molecule design. In *Advances in Neural Information Processing Systems*, pages 7795–7804, 2018. 2.1

[37] Yibo Li, Liangren Zhang, and Zhenming Liu. Multi-objective de novo drug design with conditional graph generative model. *Journal of cheminformatics*, 10(1):33, 2018. 2.1

[38] Jiaxuan You, Rex Ying, Xiang Ren, William L Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. *arXiv preprint arXiv:1802.08773*, 2018. 2.1, 2.3.1, 2.3.1.1

[39] Fergus Imrie, Anthony R Bradley, Mihaela van der Schaar, and Charlotte M Deane. Deep generative models for 3d linker design. *Journal of chemical information and modeling*, 60 (4):1983–1995, 2020. 2.1

[40] Miha Skalic, José Jiménez Luna, Davide Sabbadin, and Gianni De Fabritiis. Shape-based generative modeling for de-novo drug design. *Journal of chemical information and modeling*, 2019. 2.1

[41] Tomohide Masuda, Matthew Ragoza, and David Ryan Koes. Generating 3d molecular structures conditional on a receptor binding site with deep generative models. *arXiv preprint arXiv:2010.14442*, 2020. 2.1

[42] Gregor Simm, Robert Pinsler, and José Miguel Hernández-Lobato. Reinforcement learning for molecular design guided by quantum mechanics. In *International Conference on Machine Learning*, pages 8959–8969. PMLR, 2020. 2.1

[43] Gregor N. C. Simm, Robert Pinsler, Gábor Csányi, and José Miguel Hernández-Lobato. Symmetry-Aware Actor-Critic for 3D Molecular Design. *arXiv*, nov 2020. URL http://arxiv.org/abs/2011.12747. 2.1

[44] Tarun Gogineni, Ziping Xu, Exequiel Punzalan, Runxuan Jiang, Joshua Kammeraad, Ambuj Tewari, and Paul Zimmerman. TorsionNet: A Reinforcement Learning Approach to Sequential Conformer Search. *arXiv*, jun 2020. URL http://arxiv.org/abs/2006.07078. 2.1

[45] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, pages 1045–1048. Makuhari, 2010. 2.2.1

[46] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*,

9(8):1735–1780, 1997. 2.2.1

[47] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014. 2.2.1, 2.5.3, 5.3.3

[48] Tristan Deleu and Joseph Dureau. Learning operations on a stack with neural turing machines. *arXiv preprint arXiv:1612.00827*, 2016. 2.2.1

[49] John E Hopcroft and Jeffrey D Ullman. *Formal languages and their relation to automata*. Addison-Wesley Longman Publishing Co., Inc., 1969. 2.2.1

[50] Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to transduce with unbounded memory. *Advances in neural information processing systems*, 28, 2015. 2.2.1

[51] A. Gaulton, L.J. Bellis, A.P. Bento, J. Chambers, M. Davies, A. Hersey, Y. Light, S. McGlinchey, D. Michalovich, B. Al-Lazikani, and J.P. Overington. Chembl: a large-scale bioactivity database for drug discovery. *Nucleic acids research*, 40(D1):D1100–D1107, 2011. 2.2.2.1, 2.3.2.1

[52] John J Irwin, Teague Sterling, Michael M Mysinger, Erin S Bolstad, and Ryan G Coleman. Zinc: a free tool to discover chemistry for biology. *Journal of chemical information and modeling*, 52(7):1757–1768, 2012. 2.2.2.1

[53] Guy W Bemis and Mark A Murcko. The properties of known drugs. 1. molecular frameworks. *Journal of medicinal chemistry*, 39(15):2887–2893, 1996. 2.2.2.3, 3.2.3

[54] Peter Ertl and Ansgar Schuffenhauer. Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. *Journal of cheminformatics*, 1(1):1–11, 2009. 2.2.2.3, 2.3.2, 2.5.2, 6

[55] Anh Nguyen, Jason Yosinski, and Jeff Clune. Understanding neural networks via feature visualization: A survey. In *Explainable AI: interpreting, explaining and visualizing deep learning*, pages 55–76. Springer, 2019. 2.2.2.4

[56] Greg Landrum et al. Rdkit: Open-source cheminformatics, 2006. 2.3.1.2, 2.5.2, 3.2.3, 5.1,

5.3.3

[57] G Richard Bickerton, Gaia V Paolini, Jérémy Besnard, Sorel Muresan, and Andrew L Hopkins. Quantifying the chemical beauty of drugs. *Nature chemistry*, 4(2):90, 2012. 2.3.2, 2.5.2, 3.1.1, 3.1.3.3

[58] Daniil Polykovskiy, Alexander Zhebrak, Benjamin Sanchez-Lengeling, Sergey Golovanov, Oktai Tatanov, Stanislav Belyaev, Rauf Kurbanov, Aleksey Artamonov, Vladimir Aladinskiy, Mark Veselov, et al. Molecular sets (moses): A benchmarking platform for molecular generation models. *arXiv preprint arXiv:1811.12823*, 2018. 2.3.2, 2.3.2.1

[59] John J Irwin and Brian K Shoichet. Zinc - a free database of commercially available compounds for virtual screening. *Journal of chemical information and modeling*, 45(1): 177–182, 2005. 2.3.2.1

[60] David Mendez, Anna Gaulton, A Patrícia Bento, Jon Chambers, Marleen De Veij, Eloy Félix, María Paula Magariños, Juan F Mosquera, Prudence Mutowo, Michał Nowotka, et al. Chembl: towards direct deposition of bioassay data. *Nucleic acids research*, 47(D1): D930–D940, 2019. 2.4.2, 2.5.2, 2.5.2, 2.5.2, 3.1.1, 3.2.3, 4.1, 4.2.1, 4.2.1, 4.3.1, 5.3.3

[61] R Zubatyuk, JS Smith, BT Nebgen, S Tretiak Nature . . . , and undefined 2021. Teaching a neural network to attach and detach electrons from molecules. *nature.com*. URL https://www.nature.com/articles/s41467-021-24904-0. 2.4.2

[62] R. S. Bohacek, C. McMartin, and W. C. Guida. The art and practice of structure-based drug design: a molecular modeling perspective. *Medicinal research reviews*, 16(1):3–50, 1996. 2.5.1, 5.1

[63] Christopher A. Lipinski. Lead- and drug-like compounds: the rule-of-five revolution. *Drug Discovery Today: Technologies*, 1(4):337–341, dec 2004. ISSN 17406749. doi: 10.1016/j.ddtec.2004.11.007. 2.5.1, 2.5.2

[64] Diversity Libraries - Enamine, . URL https://enamine.net/compound-libraries/diversity-libraries. 2.5.2

[65] David S. Wishart, Craig Knox, An Chi Guo, Dean Cheng, Savita Shrivastava, Dan Tzur, Bijaya Gautam, and Murtaza Hassanali. DrugBank: a knowledgebase for drugs,

drug actions and drug targets. *Nucleic Acids Research*, 36(Database issue):D901, jan 2008. ISSN 03051048. doi: 10.1093/NAR/GKM958. URL /pmc/articles/PMC2238889//pmc/articles/PMC2238889/?report=abstracthttps://www.ncbi.nlm.nih.gov/pmc/articles/PMC2238889/. 2.5.2

[66] Scott A. Wildman and Gordon M. Crippen. Prediction of physicochemical parameters by atomic contributions. *Journal of Chemical Information and Computer Sciences*, 1999. ISSN 00952338. doi: 10.1021/ci990307l. 2.5.2

[67] IBM RXN for Chemistry, . URL https://rxn.res.ibm.com/rxn/sign-in. 2.5.2, 6

[68] Philippe Schwaller, Teodoro Laino, Théophile Gaudin, Peter Bolgar, Christopher A. Hunter, Costas Bekas, and Alpha A. Lee. Molecular Transformer: A Model for Uncertainty-Calibrated Chemical Reaction Prediction. *ACS Central Science*, 5(9):1572–1583, sep 2019. ISSN 23747951. doi: 10.1021/ACSCENTSCI.9B00576. URL https://pubs.acs.org/doi/abs/10.1021/acscentsci.9b00576. 2.5.2

[69] Alexander N. Shivanyuk, Sergey V. Ryabukhin, Andrey V. Bogolyubsky, Dmytro M. Mykytenko, Alexander A. Chupryna, William Heilman, Alexander N. Kostyuk, and Andreya Tolmachev. Enamine real database: Making chemical diversity real. *Chimica Oggi*, 2007. ISSN 0392839X. 2.5.2

[70] eMolecules. URL https://www.emolecules.com/. 2.5.2

[71] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach 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. In *Advances in Neural Information Processing Systems*, 2019. 2.5.3, 5.1

[72] rxn4chemistry/rxn4chemistry: Python wrapper for the IBM RXN for Chemistry API, . URL https://github.com/rxn4chemistry/rxn4chemistry. 2.5.3

[73] Benjamin Sanchez-Lengeling and Alán Aspuru-Guzik. Inverse molecular design using

machine learning: Generative models for matter engineering. *Science*, 361(6400):360–365, 2018. 3.1.1

[74] Petra Schneider, W Patrick Walters, Alleyn T Plowright, Norman Sieroka, Jennifer Listgarten, Robert A Goodnow, Jasmin Fisher, Johanna M Jansen, José S Duca, Thomas S Rush, et al. Rethinking drug design in the artificial intelligence era. *Nature Reviews Drug Discovery*, 19(5):353–364, 2020. 3.1.1

[75] Michael Moret, Lukas Friedrich, Francesca Grisoni, Daniel Merk, and Gisbert Schneider. Generative molecular design in low data regimes. *Nature Machine Intelligence*, 2(3): 171–180, 2020. 3.1.1

[76] José Jiménez-Luna, Francesca Grisoni, and Gisbert Schneider. Drug discovery with explainable artificial intelligence. *Nature Machine Intelligence*, 2(10):573–584, 2020. 3.1.1

[77] Mariya Popova, Mykhailo Shvets, Junier Oliva, and Olexandr Isayev. Molecularrnn: Generating realistic molecular graphs with optimized properties. *arXiv preprint arXiv:1905.13372*, 2019. 3.1.1

[78] Rocío Mercado, Tobias Rastemo, Edvard Lindelöf, Günter Klambauer, Ola Engkvist, Hongming Chen, and Esben Jannik Bjerrum. Practical notes on building molecular graph generative models. *Applied AI Letters*, 1(2), 2020. 3.1.1

[79] N De Cao and T Kipf. Molgan: An implicit generative model for small molecular graphs, 2018. *arXiv preprint arXiv:1805.11973*. 3.1.1

[80] Jaechang Lim, Sang-Yeon Hwang, Seokhyun Moon, Seungsu Kim, and Woo Youn Kim. Scaffold-based molecular design with a graph generative model. *Chemical science*, 11(4): 1153–1164, 2020. 3.1.1

[81] Gabriel Lima Guimaraes, Benjamin Sanchez-Lengeling, Carlos Outeiral, Pedro Luis Cunha Farias, and Alán Aspuru-Guzik. Objective-reinforced generative adversarial networks (organ) for sequence generation models. *arXiv preprint arXiv:1705.10843*, 2017. 3.1.1, 3.2

[82] Evgeny Putin, Arip Asadulaev, Quentin Vanhaelen, Yan Ivanenkov, Anastasia V Aladinskaya, Alex Aliper, and Alex Zhavoronkov. Adversarial threshold neural computer for molecular de novo design. *Molecular pharmaceutics*, 15(10):4386–4397, 2018. 3.1.1, 5.1

[83] Thomas Blaschke, Ola Engkvist, Jürgen Bajorath, and Hongming Chen. Memory-assisted reinforcement learning for diverse molecular de novo design. *Journal of cheminformatics*, 12(1):1–17, 2020. 3.1.1

[84] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Multi-objective molecule generation using interpretable substructures. In *International conference on machine learning*, pages 4849–4859. PMLR, 2020. 3.1.1

[85] Jannis Born, Matteo Manica, Joris Cadow, Greta Markert, Nil Adell Mill, Modestas Filipavicius, Nikita Janakarajan, Antonio Cardinale, Teodoro Laino, and María Rodríguez Martínez. Data-driven molecular design for discovery and synthesis of novel ligands: a case study on sars-cov-2. *Machine Learning: Science and Technology*, 2(2):025024, 2021. 3.1.1

[86] Alex Zhavoronkov, Yan A Ivanenkov, Alex Aliper, Mark S Veselov, Vladimir A Aladinskiy, Anastasiya V Aladinskaya, Victor A Terentiev, Daniil A Polykovskiy, Maksim D Kuznetsov, Arip Asadulaev, et al. Deep learning enables rapid identification of potent ddr1 kinase inhibitors. *Nature biotechnology*, 37(9):1038–1040, 2019. 3.1.1, 4.2.3.1

[87] Nathan Brown, Marco Fiscato, Marwin HS Segler, and Alain C Vaucher. Guacamol: benchmarking models for de novo molecular design. *Journal of chemical information and modeling*, 59(3):1096–1108, 2019. 3.1.1

[88] R. Williams. A class of gradient-estimation algorithms for reinforcement learning in neural networks. In *International Conference on Neural Networks*, 1987. 3.1.2.2, 3.2.2

[89] Christopher A Lipinski, Franco Lombardo, Beryl W Dominy, and Paul J Feeney. Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings. *Advanced drug delivery reviews*, 23(1-3):3–25, 1997. 3.1.3.1

[90] Robert Kralovics, Francesco Passamonti, Andreas S Buser, Soon-Siong Teo, Ralph Tiedt, Jakob R Passweg, Andre Tichelli, Mario Cazzola, and Radek C Skoda. A gain-of-function mutation of jak2 in myeloproliferative disorders. *New England Journal of Medicine*, 352 (17):1779–1790, 2005. 3.1.3.1

[91] Matthew E Welsch, Scott A Snyder, and Brent R Stockwell. Privileged scaffolds for library design and drug discovery. *Current opinion in chemical biology*, 14(3):347–361, 2010. 3.1.3.1

[92] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008. 3.1.3.2

[93] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 3.1.3.3, 5.3.1

[94] Y. Tetko, I.V. ans Sushko, S. Novotarskyi, L. Patiny, I. Kondratov, A.E. Petrenko, L. Charochkina, and A.M. Asiri. How accurately can we predict the melting points of drug-like compounds? *Journal of chemical information and modeling*, 54(12):D1100–D1107, 2014. 3.1.3.3, 5.3.3

[95] Artem Cherkasov, Eugene N Muratov, Denis Fourches, Alexandre Varnek, Igor I Baskin, Mark Cronin, John Dearden, Paola Gramatica, Yvonne C Martin, Roberto Todeschini, et al. Qsar modeling: where have you been? where are you going to? *Journal of medicinal chemistry*, 57(12):4977–5010, 2014. 3.2.1

[96] Alexander Tropsha. Best practices for qsar model development, validation, and exploitation. *Molecular informatics*, 29(6-7):476–488, 2010. 3.2.1

[97] Maja J Mataric. Reward functions for accelerated learning. In *Machine learning proceedings 1994*, pages 181–189. Elsevier, 1994. 3.2.1

[98] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *International conference on machine learning*, pages 1515–1528. PMLR, 2018. 3.2.1, 3.2.2.1

[99] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019. 3.2.1

[100] Maria Korshunova, Boris Ginsburg, Alexander Tropsha, and Olexandr Isayev. Openchem: A deep learning toolkit for computational chemistry and drug design. *Journal of Chemical Information and Modeling*, 61(1):7–13, 2021. 3.2.2, 5.2.1

[101] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989. 3.2.2.1

[102] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015. 3.2.2.1

[103] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. 3.2.2.1

[104] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018. 3.2.2.1

[105] Hoang Thanh-Tung and Truyen Tran. Catastrophic forgetting and mode collapse in gans. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10. IEEE, 2020. 3.2.3

[106] Kinase Library - Enamine, . URL `https://enamine.net/hit-finding/focused-libraries/kinase-library`. 3.2.3

[107] Cheryl H Arrowsmith, James E Audia, Christopher Austin, Jonathan Baell, Jonathan Bennett, Julian Blagg, Chas Bountra, Paul E Brennan, Peter J Brown, Mark E Bunnage, et al. The promise and peril of chemical probes. *Nature chemical biology*, 11(8):536–541, 2015. 4.1

[108] Robert M Garbaccio and Emma R Parmee. The impact of chemical probes in drug discovery: a pharmaceutical industry perspective. *Cell chemical biology*, 23(1):10–17, 2016. 4.1

[109] Susan Klaeger, Stephanie Heinzlmeir, Mathias Wilhelm, Harald Polzer, Binje Vick, Paul-Albert Koenig, Maria Reinecke, Benjamin Ruprecht, Svenja Petzoldt, Chen Meng, et al. The target landscape of clinical kinase drugs. *Science*, 358(6367):eaan4368, 2017. 4.1

[110] Chemical Probes — SGC. URL `https://www.thesgc.org/chemical-probes`.

4.1

[111] Piya Lahiry, Ali Torkamani, Nicholas J Schork, and Robert A Hegele. Kinase mutations in human disease: interpreting genotype–phenotype relationships. *Nature Reviews Genetics*, 11(1):60–74, 2010. 4.1

[112] Denis Fourches, Eugene Muratov, and Alexander Tropsha. Trust, but verify: on the importance of chemical structure curation in cheminformatics and qsar modeling research. *Journal of chemical information and modeling*, 50(7):1189, 2010. 4.2.1, 5.3.1

[113] Alexander J Bridges, Hairong Zhou, Donna R Cody, Gordon W Rewcastle, Amy McMichael, HD Hollis Showalter, David W Fry, Alan J Kraker, and William A Denny. Tyrosine kinase inhibitors. 8. an unusually steep structure- activity relationship for analogues of 4-(3-bromoanilino)-6, 7-dimethoxyquinazoline (pd 153035), a potent inhibitor of the epidermal growth factor receptor. *Journal of medicinal chemistry*, 39(1):267–276, 1996. 4.2.1

[114] Carrow I Wells, Hassan Al-Ali, David M Andrews, Christopher RM Asquith, Alison D Axtman, Ivan Dikic, Daniel Ebner, Peter Ettmayer, Christian Fischer, Mathias Frederiksen, et al. The kinase chemogenomic set (kcgs): An open science resource for kinase vulnerability identification. *International journal of molecular sciences*, 22(2):566, 2021. 4.2.1

[115] Jin H Park, Yingting Liu, Mark A Lemmon, and Ravi Radhakrishnan. Erlotinib binds both inactive and active conformations of the egfr tyrosine kinase domain. *Biochemical Journal*, 448(Pt 3):417, 2012. 4.2.1

[116] Jennifer Stamos, Mark X Sliwkowski, and Charles Eigenbrot. Structure of the epidermal growth factor receptor kinase domain alone and in complex with a 4-anilinoquinazoline inhibitor. *Journal of Biological Chemistry*, 277(48):46265–46272, 2002. 4.2.1

[117] Daniel Merk, Francesca Grisoni, Lukas Friedrich, and Gisbert Schneider. Tuning artificial intelligence on the de novo design of natural-product-inspired retinoid x receptor modulators. *Communications Chemistry*, 1(1):1–9, 2018. 4.2.3.1

[118] Oleksandr O Grygorenko, Dmytro S Radchenko, Igor Dziuba, Alexander Chuprina,

Kateryna E Gubina, and Yurii S Moroz. Generating multibillion chemical space of readily accessible screening compounds. *Iscience*, 23(11):101681, 2020. 4.2.3.1

[119] XGBoost Documentation — xgboost 1.5.2 documentation. URL https://xgboost.readthedocs.io/en/stable/. 4.3.1

[120] Hyperopt Documentation. URL http://hyperopt.github.io/hyperopt/. 4.3.1

[121] Products • BioSolveIT. URL https://www.biosolveit.de/products/#FTrees. 4.3.1

[122] Eugene N. Muratov, Jürgen Bajorath, Robert P. Sheridan, Igor V. Tetko, Dmitry Filimonov, Vladimir Poroikov, Tudor I. Oprea, Igor I. Baskin, Alexandre Varnek, Adrian Roitberg, Olexandr Isayev, Stefano Curtalolo, Denis Fourches, Yoram Cohen, Alan Aspuru-Guzik, David A. Winkler, Dimitris Agrafiotis, Artem Cherkasov, and Alexander Tropsha. QSAR without borders. *Chemical Society reviews*, 2020. ISSN 14604744. doi: 10.1039/d0cs00098a. 5.1

[123] Yanli Wang, Stephen H Bryant, Tiejun Cheng, Jiyao Wang, Asta Gindulyte, Benjamin A Shoemaker, Paul A Thiessen, Siqian He, and Jian Zhang. Pubchem bioassay: 2017 update. *Nucleic acids research*, 45(D1):D955–D963, 2017. 5.1

[124] Sung Jin Cho, Weifan Zheng, and Alexander Tropsha. Rational combinatorial library design. 2. rational design of targeted combinatorial peptide libraries using chemical similarity probe and the inverse qsar approaches. *Journal of Chemical Information and Computer Sciences*, 38(2):259–268, 1998. 5.1

[125] Valerie J Gillet, Wael Khatib, Peter Willett, Peter J Fleming, and Darren VS Green. Combinatorial library design using a multiobjective genetic algorithm. *Journal of chemical information and computer sciences*, 42(2):375–385, 2002. 5.1

[126] Zhenpeng Zhou, Steven Kearnes, Li Li, Richard N Zare, and Patrick Riley. Optimization of molecules via deep reinforcement learning. *Scientific reports*, 9(1):1–10, 2019. 5.1

[127] Hisaki Ikebata, Kenta Hongo, Tetsu Isomura, Ryo Maezono, and Ryo Yoshida. Bayesian molecular design with a chemical language model. *Journal of computer-aided molecular*

design, 31(4):379–391, 2017. 5.1

[128] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. {TensorFlow}: A system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016. 5.1

[129] Amanda J Minnich, Kevin McLoughlin, Margaret Tse, Jason Deng, Andrew Weber, Neha Murad, Benjamin D Madej, Bharath Ramsundar, Tom Rush, Stacie Calad-Thomson, et al. Ampl: a data-driven modeling pipeline for drug discovery. *Journal of chemical information and modeling*, 60(4):1955–1968, 2020. 5.1

[130] J A Beauman and P H Howard. Physprop database. *Syracuse Research, Syracuse, NY, USA*, 1995. 5.3.1

[131] Igor V Tetko, Vsevolod Yu Tanchuk, and Alessandro EP Villa. Prediction of n-octanol/water partition coefficients from physprop database using artificial neural networks and e-state indices. *Journal of chemical information and computer sciences*, 41(5):1407–1421, 2001. 5.3.1

[132] Stephen J Capuzzi, Regina Politi, Olexandr Isayev, Sherif Farag, and Alexander Tropsha. Qsar modeling of tox21 challenge stress response and nuclear receptor signaling toxicity assays. *Frontiers in Environmental Science*, 4:3, 2016. 5.3.2, 5.3.2

[133] Esben Jannik Bjerrum. Smiles enumeration as data augmentation for neural network modeling of molecules. *arXiv preprint arXiv:1703.07076*, 2017. 5.3.2

[134] Rich Caruana. Multitask Learning. *Machine Learning*, 1997. ISSN 08856125. doi: 10.1023/A:1007379606734. 5.3.2

[135] Thomas Unterthiner, Andreas Mayr, Günter Klambauer, and Sepp Hochreiter. Toxicity prediction using deep learning. *arXiv preprint arXiv:1503.01445*, 2015. 5.3.2