LEIDEN UNIVERSITY

Universiteit Leiden

MASTER'S THESIS

Designing Diverse and Synthesizable Molecules using Multi-Objective Generative Flow Networks guided by Retrosynthetic Accessibility

Author:
Julius Cathalina

Supervisors:
Dr. M. Preuss
Prof.dr. G.J.P. van Westen

A thesis submitted in fulfilment of the requirements for the degree of Computer Science: Bioinformatics

in the

Leiden Institute of Advanced Computer Science

in collaboration with

Computational Drug Discovery Group
Leiden Academic Centre for Drug Research

January 30, 2023

"Drug research, as we know it today, began its career when chemistry had reached a degree of maturity that allowed its principles and methods to be applied to problems outside of chemistry itself and when pharmacology had become a well- defined scientific discipline in its own right."

Jürgen Drews

LEIDEN UNIVERSITY

Abstract

Leiden Institute of Advanced Computer Science

Leiden Academic Centre for Drug Research

Computer Science: Bioinformatics

Designing Diverse and Synthesizable Molecules using Multi-Objective Generative Flow Networks guided by Retrosynthetic Accessibility

by Julius Cathalina

Drug discovery is an error-prone and resource-hungry process, exacerbated by the complex nature of interactions between pharmaceuticals with biological targets and the intractable amount of viable molecules that exist. It is therefore necessary to optimize our search for molecules that possess the properties that we are interested in. In recent times, we have adopted an in-silico approach to drug discovery, but despite the technological advances, most available tools propose molecules that are either infeasible to synthesize or that are trivially similar to known compounds. Therefore, it is crucial to generate diverse batches of molecules while simultaneously ensuring that the desired properties are respected. Generative Flow Networks (GFlowNets) are novel models that, similarly to Reinforcement Learning (RL) algorithms, are able to discover candidates that maximize desired properties, but in a manner where they are sampled proportionally to the quality of the solution. This approach enables us to satisfy the criteria that we laid out for in-silico drug discovery. We compare our results to an RL-based molecule generator to assess the benefits gained from using GFlowNets, while simultaneously incorporating constraints that increase the feasibility of the molecules we generate. Our results show that GFlowNets do deliver the desired diversity without sacrificing solution quality, and that we can successfully incorporate complex constraints that help make the downstream library of generated molecules more accessible.

Acknowledgements

I would like to thank my supervisors, Mike and Gerard, for their patience and understanding throughout my research process and the creation of this thesis. I also want to thank Anthe, who was involved in the early phases of my research, and helped me understand the practical value of my research for chemists. Furthermore, I would like to thank Alan for guiding me in the structure of my paper, and most importantly, introducing me to GFlowNets. It inspired me immensely and led to this final iteration of my research. I am also immensely grateful to my wife, Iona, who supported me throughout this entire process even when I was unsure on how to proceed. Finally, I would like to thank my colleagues for always supporting me from the sidelines and motivating me to keep writing alongside my full-time job – it was quite the undertaking!

Contents

Acronyms

AI Artificial Intelligence 2, 3

CASP Computer-Aided Synthesis Planning 3, 5, 18, 19, 58–60
CI conditioning information 30

DAG Directed Acyclic Graph 9–12, 20
DL Deep Learning 3
DNNs Deep Neural Networks 2, 18, 22
DRL Deep Reinforcement Learning 2, 9

EA Evolutionary Algorithms 7, 9

GFN Generative Flow Network 2, 4, 5, 7, 9, 11–13, 17, 20, 21, 23, 24, 28–59, 65
GT Graph Transformer 28, 30, 31

HTS High Throughput Screening 1

MCTS Monte Carlo tree search 18
MLP Multi-Layer Perceptron 30
MOGFN Multi-Objective GFlowNet 17, 18, 21, 33, 55
MOO Multi-Objective Optimization 7, 16, 33
MPNN Message-Passing Neural Network 5

PC Preference-Conditioned 21, 33, 55
PCA Principal Component Analysis 5

QED Quantitative Estimation of Drug-likeness 39, 41, 42
QSAR Quantitative Structure Activity Relationship 1, 5, 9, 16, 56

RL Reinforcement Learning 2, 4, 5, 7, 9, 11, 57, 59
RNNs Recurrent Neural Networks 4, 5

SELFIES Self-Referencing Embedded String 6–8, 32, 33
SMILES Simplified Molecular Input Line Entry System 6, 7

SubTB Sub-Trajectory Balance 12–14, 21, 33–35, 60

t-SNE t-distributed Stochastic Neighbor Embedding 5

TB Trajectory Balance 12, 13, 21, 33–35, 38, 40, 41, 43, 47, 48, 50, 52–54, 59, 60

Chapter 1

Introduction

1.1 A brief history of drug discovery

Drug discovery as a formal discipline is relatively new when it comes to the field of chemistry, but the practice thereof can be traced back to many centuries ago, when humans began to extract medicinal compounds from plants and other resources found in nature [1]. The approaches used to discover new medicinal compounds has continuously been refined and evolved over time, with significant breakthroughs being made during the 19th and 20th century. The advancements in this field at the time included the isolation of morphine from opium extract, and concepts such as ligand-receptor interactions being established [2]. While these findings laid the groundwork for modern chemical theory, the emergence of computational chemistry in the latter half of the 1900s, enabled by improvements in computational power, made a lot of said theory practical.

Approaches such as Quantitative Structure Activity Relationship (QSAR) studies and (automated) High Throughput Screening (HTS) accelerated molecule design iterations. QSAR studies are used to analyse correlations between molecular structures and their respective biological behaviours, while the purpose of a technique such as HTS is to discover biologically relevant molecules by testing a large amount of compounds against biological targets of interest [3]. In the ideal case, this leads to candidate compounds (also called "hits" or "lead compounds") that exhibit the desired effects on the target, but are not necessarily viable as pharmaceuticals. In this way, HTS can be thought of as a quick filter to guide the search for relevant compounds.

Despite the innovation that computational chemistry initially brought, one well established hurdle in drug discovery was that the amount of potentially viable drug-like molecules that can exist represented a prohibitively large chemical space, estimated to be of a size larger than 10^{60} [4]. If chemists wanted computer-assisted lead compound generation, the aforementioned techniques, while powerful for filtering or lead optimization, would not be sufficient. Searching this vast space with approaches such as HTS, or any other technique that would require enumerating all the possible molecules, would not be tractable. To this end, approaches that concerned themselves with effectively searching limited parts of this space were developed.

1.2 De novo drug design

From the time of writing, it has been roughly three decades since the introduction of automated de novo drug design [5], a technique that has proven itself to be paramount in the advancement of drug discovery projects. The main purpose

of this technique is the generation of novel drug-like molecular structures that exhibit desired pharmacological properties [6]. The generated molecular structures should ideally be mappable to sets of descriptor values that characterize the pharmacological behaviour that we are looking for. A more detailed explanation of the mechanisms driving de novo drug design is given in section 2.1.

This technique offered a new way for chemists to discover possible lead molecules for their biological targets. However, the early days of de novo drug design were plagued by the lack of promising results, and it was subsequently set aside in favour of other approaches that were easier to implement in practice [7]. One possible explanation for this lack of results was that the researchers at the time undervalued the importance of drug-like properties and synthetic feasibility of the designed compounds [6]. In the last decade, de novo drug design had a resurgence, partially attributable to the emergence of compound libraries that were generated through techniques such as combinatorial chemistry. Steering away from the approach of optimizing a lead compound on potency over its drug-like properties helped as well, as that approach was susceptible to attrition in the later stages of the drug discovery pipeline. This resulted in many research projects where the compounds that were designed were also synthesized and evaluated in the wet-lab, bolstering the credibility of the technique.

1.3 Artificial-intelligence approaches in drug design

Due to the recent improvements in the Artificial Intelligence (AI) landscape, many new techniques have been proposed to generate chemical compounds exhibiting the desired physicochemical or biological properties. Notably, approaches based on a combination of Reinforcement Learning (RL) and Deep Neural Networks (DNNs), also called Deep Reinforcement Learning (DRL), have gained traction in this field [8, 9]. Results from prominent papers based on DRL, among other AI subfields in drug design, will be further elaborated on in chapter 2. RL has been used to effectively address problems involving high degrees of dynamic decision-making under uncertainty. In particular, games with high theoretical complexity such as chess and go have seen success through DRL, with groundbreaking examples such as AlphaZero from 2017 [10] achieving superhuman performance. While RL has been studied extensively within the realm of game mastery, it is not limited to this space. The same company behind the aforementioned AlphaZero have also contributed greatly to the domain of science with the creation of AlphaFold, a protein-structure prediction (a notoriously difficult task) tool that performed far beyond the state-of-the-art at the time, and even more recently they showed that RL can even push the boundaries of foundational mathematics with AlphaTensor as it discovered novel algorithms for matrix multiplication using fewer multiplications (and additions) than the best known methods [11, 12]

In 2021, a paper by Bengio et al. introduced a novel framework by the name of Generative Flow Network (GFN), which borrows many ideas from RL, with a crucial distinction being that instead of asymptotically moving towards sampling from the single best mode (that could be found) with the highest rewards, it samples around all the high modes proportionally to the rewards associated with them. Intuitively, this means that high sample diversity is embedded in (well-trained) GFN, granted that the high-rewards mode can actually be found efficiently (i.e., the reward landscape is not too sparse or complex). The theory behind GFN and the methods that

are used to enable its theoretical properties will also be further elaborated on in chapter 2.

1.4 The challenges of AI-based de novo drug design

Synthesizability

Breakthroughs in AI have advanced the field of drug discovery past the point of enumerating intractable quantities of molecules and screening them in hopes of finding promising hits. The generative algorithms that have been developed following the success of Deep Learning (DL) methods have been particularly impactful [8, 13]. These methods commonly consist of models that learn to map continuous numerical representations to a discrete chemical space, enabling the efficient exploration of said space. Given a model that can associate a compound with a quantifiable value of interest, it can be used in conjunction with the previous idea to exploit the learned mapping to find molecules that are optimal according to said value. This approach is also known as a goal-directed generation task, and is a popular approach used in modern de novo drug design. Despite its favourable theoretical properties, goal-directed generation frequently results in molecules that are unlikely, if not impossible, to be synthesized in a lab, effectively limiting the usefulness of such models for practical applications [14].

To combat this issue, it has become increasingly popular to include heuristics at some point during the molecule generation pipeline to prevent models from suggesting wildly unreasonable candidates. Many heuristics exist to estimate the synthesizability of molecules, ranging from simple ones such as SMILES length or molecular weight, to more complex ones such as the SA (Synthetic Accessibility) score, SC (Synthetic Complexity) score and SYBA (SYnthetic Bayesian Accessibility) [15, 16, 17]. Although these heuristics are a reasonable proxy for the synthesizability of compounds, they are often built on a limited number of assumptions of what makes a molecule complex to synthesize. Modern Computer-Aided Synthesis Planning (CASP) tools such as ASKCOS[18] and AiZynthFinder[19] capture more of the complexity inherent to synthetic planning with respect to chemical structure. They directly take into account possible reaction templates that can lead to the compound of interest based on explicit constraints given by the chemist. Additionally, these tools also have the advantage of returning the pathways found through retrosynthetic planning, adding a layer of interpretability to the molecule generation process. Furthermore, most of these CASP tools rely on a given stock of, e.g., commercially available molecules and specific policies. This does mean that the tool may misclassify certain compounds as inaccessible if the stock and/or policy does not have enough coverage. On the other hand, we can take advantage of this to limit the scope of the model's exploration to more accessible compounds, further improving the efficiency of our search.

The benefits gained from incorporating CASP tools to guide generative models come at a cost – synthesis planning is costly, and in the context of optimization tasks where these tools would be called copious amounts of times, the problem quickly becomes intractable [14]. This issue can partially be ameliorated by creating a proxy trained on the results of the actual CASP tool, and optionally combining it with the less expensive heuristics to retrieve a quantifiable indicator of synthesizability. This is the approach that is used throughout this paper. The above-mentioned heuristics and their relation to modern CASP tools are expanded on in section 3.3 and section 4.2.

Chemical Diversity

The ability to propose a diverse set of candidate molecular structures has many benefits for generative models, yet it is not a property that is trivial to guarantee. Having high chemical diversity means that a model is more likely to propose molecules with desirable properties that are structurally dissimilar to known compounds. These molecules could result in unexpected leads – and if chemically viable and synthetically accessible – have the edge of being favourable in terms of intellectual property. Furthermore, being able to sample more diverse sets of molecules means that the model explores the chemical space more broadly. In the context of goal-directed tasks, this also means that the model is less likely to get trapped in local extremes of the chemical search space. Intuitively, resilience to local extremes is essential given the magnitude of the search space, combined with the assumption that a miniscule fraction of it actually contains synthetically accessible compounds that also exhibit the desired properties.

Modern approaches to de novo drug design have implemented various solutions to enforce diversity within generative models. The most intuitive approach, is to use a dataset covering a broad chemical space to train the models, but within goal-directed task settings, this objective can be improved on. Olivecrona et al. have used "diversity filters" to penalize their models if they become too fixated on a narrow set of compounds [13, 20]. Liu et al. used a second, frozen version of their generative model as an exploration network, in a way that is analogous to an intelligent ϵ-greedy RL algorithm [21]. In a subsequent version, this idea was extended further by incorporating multiple Recurrent Neural Networks (RNNs) acting as crossover and mutation operators – essentially transforming it into an evolutionary algorithm approach to enforce exploration during RL-based optimization. The ideas used in these papers will be further detailed on in section 3.1. As mentioned above, GFN can be trained to attain the convenient property of implicitly modelling diversity. It does so (when trained properly) by sampling candidates proportionally to their expected goodness, unlike RL where the goal is to sample from the best mode that is known based on some reward function. GFN are discussed more extensively in section 2.2 and section 3.2.

1.5 Research questions

In this paper, we set out to answer the following (sub)questions regarding automated in-silico de novo drug design:

1. Can we exploit the reward signal to overcome the diversity "hurdle" present in many RL-based state-of-the-art generative models?
2. How can we incorporate retrosynthetic planning into generative models during training, and does it improve the feasibility of downstream molecular libraries?

1.6 Thesis structure

In an attempt to remain self-contained, this thesis will start out by giving a high-level overview of the theoretical background (see chapter 2) needed to understand the approaches and interpret the results. The main topics of synthesizability and its incorporation within (de-novo) drug design will be discussed in an in-silico context, along with their associated challenges. It then covers the broad topic of how molecules are

represented in computers, and how we can manipulate said representations to be compatible with deep learning techniques. The list of representations discussed in this thesis is far from exhaustive, but covers the most popular approaches at the time of writing. The motivation for using these representations will be given, and a superficial comparison between them will be given as to aid the reader in understanding why certain representations are more advantageous in different contexts. It then goes over the two approaches used for the sampling of molecules from the large chemical search space: RL and GFN. The fundamental ideas behind these approaches will be discussed, while referring to other papers where necessary for explanations on their more detailed technical aspects. QSAR will briefly be discussed as an explanation of how (unknown) molecules can be mapped to some quantifiable pharmacologic property value for the purposes of evaluating the goodness of our generated molecules, albeit with some uncertainty. Finally, it will discuss the concept of retrosynthetic planning – common heuristics for synthesizability will be briefly discussed to motivate why more advanced retrosynthetic planning is useful and should be incorporated into our generative models. Then the theory behind modern CASP tools for retrosynthetic planning themselves will be discussed.

In chapter 3, this thesis will cover the most influential publications that our underlying research builds on, and will briefly mention other papers that have implemented similar ideas or were instrumental in building insight to carry out this research. Chapter 4 will explain the methods used in this thesis, that being the types of neural networks being used (e.g., RNNs, Message-Passing Neural Network (MPNN)), the molecular representations and how they're encoded, and the scoring methods being used. Chapter 5 goes over the results, being the generated molecules that result from the different approaches. It will generally follow a pattern of showcasing sampled molecules every so many steps during optimization, along with their predicted properties, highlighting the shift between the distribution of said properties before and after optimization. A two-dimensional view of the molecules using, e.g., Principal Component Analysis (PCA) and t-distributed Stochastic Neighbor Embedding (t-SNE), will also be used to show approximately where the optimized molecules end up in chemical space in relation to the original training dataset, based on their molecular fingerprint. Chapter 6 will discuss the above-mentioned results, in an attempt to explain the observations and limitations of the current approach, and chapter 7 concludes the thesis by highlighting the main findings and summarizing the lessons learned throughout the research process.

Chapter 2

Theoretical background

2.1 Representing Chemistry in Computers

With the advent of powerful computers, the discipline of processing molecular information in-silico took shape in what is now known as cheminformatics. This presented an exciting way to advance research in chemistry. Researchers could now attempt to (albeit imperfectly) predict the behaviour of chemical compounds through simulation. Naturally, researchers needed a way to represent said compounds in a computer before they could do anything useful. One of the earliest widely accepted solutions to this representation problem was Simplified Molecular Input Line Entry System (SMILES). SMILES is an intuitive string-based representation of a molecule, invented in 1988 by David Weiniger[22]. While SMILES was and still is one of the most useful ways of representing molecules in a computer, it has certain properties that make it less suitable for tasks in modern automated de novo drug design. The SMILES specification rules and its limitations will be briefly discussed in the following section. Next to SMILES, many other representations have also been created, such as Self-Referencing Embedded String (SELFIES), DeepSMILES (SMILES catered towards deep learning tasks), and also intuitively – graph representations[23, 24, 25, 26]. The scope of this research will only concern itself with three of these representations: SMILES and SELFIES, and graphs, where SMILES will be used as the general interface for all the components to work together, while the generative models will be using SELFIES or graphs internally to leverage the 100% validity guarantee for the former case, and the additional features that can be extracted from graphs in the latter. In the following sections, we will also elaborate on how SELFIES achieves this validity guarantee, how it maps to molecular graphs and why we chose to use it for our first generator. Then we will talk about how we can make clever use of the intuitive graph representation of molecules to increase the stability of molecular generators in settings where this can otherwise easily collapse.

Multi-Objective Optimization

More often than not, it is not sufficient to optimize drugs for a single biological target – the interactions between chemically active compounds and the systems with which they interact are complex, and can result in many side effects. For this reason, it is paramount to consider multiple properties that a drug may have when considering candidates for pharmaceuticals. We may, for example, want to maximize (or minimize if the goal is for a compound to **not** be active for a certain biological target) all the properties for a given candidate x as in $\max_{x \in \mathcal{X}}(R_1(x), R_2(x), ..., R_n(x))$, where $R_i(x)$ is the reward that is used as a candidate's score for the i^{th} property. However, finding candidates that meet all the criteria for the properties that we consider is difficult because of potentially conflicting properties, where improving one may result

in other properties deteriorating. This is known as the Multi-Objective Optimization (MOO) problem, and is also the driving force behind the design of the reward signal in this paper, which will be discussed in chapter 4. Because it may be infeasible to optimize all properties simultaneously, we need to use an approach that can find all the best candidates w.r.t. the trade-offs being made between each property – these are known as pareto-optimal candidates. To achieve this, candidates must be compared on each property to determine a so-called "dominant" candidate – that is, if a candidate is just as good as the one it is being compared to for all n properties, **and** is *strictly better* in at least one property, it is said to dominate (denoted by the symbol $\succ$) the other. A more formal definition is given below:

$$\text{Given two candidate solutions } \{x_1, x_2\} \in \mathcal{X}, (x_1 \succ x_2) \text{ iff:}$$
$$\forall i : R_i(x_1) \geq R_i(x_2) \wedge \exists i : R_i(x_1) > R_i(x_2)$$
$$\text{where } i \in \{1, ...n\}, \text{ and } n \text{ is the number of properties to optimize.}$$

Most standard RL algorithms expect a scalar reward signal, and while other approaches exist such as distributional RL [27], the scope of our paper is limited to the former setting. This counts for the GFN training objectives as well, which will be further elaborated on in subsection 2.2.3. As we are in a MOO setting, the scores for each objective has to be combined into a single scalar value before being passed as feedback for our model to train on. There exist many weighting schemes – for the scope of this paper, we will only consider an exponentiated weighted sum (WS) as shown in section 3.2 and a custom Pareto front-based (PF) scheme, which is further detailed in section 3.1.

2.1.1 SELFIES

SMILES is the standard string-based representation used as the work-horse of cheminformatics, and has been instrumental in addressing the early challenges associated with storing molecular information in computers [22]. However, with the advent of generative models where the exploration of novel molecules is the main goal, it became increasingly evident that SMILES were not the right tool for the job. These models often need to be able to mutate and try unexplored structures to find interesting candidates, but because of the ambiguity of SMILES (e.g., the same SMILES can define multiple different structures), and the –for the computer– complicated syntax, made it so that most of the SMILES proposed by these models resulted in invalid molecules. A lot of training time is wasted simply training a model to learn the correct syntax before the fine-tuning steps, and even then during the generation phase it is still likely that a portion of the candidates will be invalid. This drawback led to the development of other representations that are more fit for this task, such as DeepSMILES[25]. Among those new representations, SELFIES was presented as a fully robust string-representation (for small molecules), meaning that any SELFIES string always corresponds to a valid molecule. It is also independent of the model used and can be used as a stand-alone representation. This property of guaranteed validity also makes it particularly intuitive to include in areas where exploration through mutation is useful, such as in Evolutionary Algorithms (EA).

The validity guarantee of SELFIES is upheld by the following properties:

1. Branch length and ring size are stored together with their corresponding sizes

2. Each SELFIES symbol is generated based on derivation rules that enforce valid chemical valence-bonds

The first property eliminates syntactical errors in the string construction (e.g., unequal number of brackets), while the second makes sure that the strings always represent chemically sane molecules. We refer the reader to the paper introducing SELFIES for more in-depth explanations of how the derivation table works and for worked out examples [24].

2.1.2 Graph Representations

Graphs are, arguably, the most intuitive way to represent molecules in-silico – it is how molecules are most commonly visualized. In the deep learning field, techniques that allow us to leverage the benefits gained from the mathematical properties of graphs have become widespread. Graphs have always had said useful properties, but deep learning was not as mature as it is at the time of writing, and hardware limitations in the past meant that storing dense graphs efficiently represented a non-trivial bottleneck. Due to current advances, there are a myriad of frameworks that facilitate deep learning using graphs, and this is especially valuable in cheminformatics, as representing chemical compounds as graphs allow researchers to mitigate some of the limitations inherent to the usage of SMILES, and enables them to work entirely within the domain of graphs. This also means that we are able to encode more information into our in-silico molecules, such as adding molecular weights as a feature directly embedded in a node, which can later be used to compute node and/or graph-level predictions.

In the context of molecular representations, a graph $\mathcal{G}$ can be formulated as a set of nodes and edges $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where each node $v \in \mathcal{V}$ can represent, e.g., an atom, and each edge $(u, v) \in \mathcal{E}$ can represent a bond. In the case of such a graph, one can set attributes on nodes or edges to enrich the information contained within it. One clear example hereof is setting an edge attribute to specify the type of bond (e.g., double or triple bonds) between two atoms. This straight-forward approach to representing molecules as graphs can be extended further – one can also store molecular substructures (henceforth "fragments") as nodes, and make the edges represent bonds between two fragments.

Similarly to SELFIES, graphs can also be leveraged to design generative models that are guaranteed to propose valid molecules [28]. In addition to this, graphs circumvent using string-based distance metrics for molecules, such as the edit distance between two SMILES to determine the similarity of compounds. While there are other solutions to this, such as first transforming the SMILES string into a molecular fingerprint and comparing the fingerprints, using graph similarity metrics can save us the cost of this additional translation step.

2.1.3 Molecular Fingerprints

Molecular fingerprints can be generalized as any data structure that encodes the structure and properties of a molecule. There exist a variety of fingerprint types, but the core idea is that a kernel is applied to a molecule to generate a bit/count-vector. This vector can be thought of as an abstract representation of the features of a molecule, with the resolution increasing with the vector-size (to a certain degree). Different types of fingerprints are more relevant depending on the context. The foremost application of these fingerprints is to efficiently compute the similarity

between molecules. This allows us to, for example, infer how diverse our generated molecular libraries are [29].

One caveat of this method is that molecular similarity is not well-defined, and that fingerprint similarity is not a perfect proxy for it – it rests on the assumption that similar molecules will share more bits in the same locations. In practice, we observe that the data set being used for the similarity computation itself already has major impact on the results. Despite this, we still leverage it as a visual indicator of the chemical space that our generated molecules occupy, and as a heuristic for how diverse they are.

2.2 GFlowNet Fundamentals

2.2.1 Generative Flow Networks (GFlowNets)

The aforementioned methods follow the pattern where a model is first trained to learn how to generate a given representation of molecules in a general sense. One can optionally fine tune the model by biasing it towards a certain molecular search space (e.g., molecules that are known to be biologically active against the target of interest), effectively focussing the model to search more within a narrower (possibly more relevant) search space.

The optimization step is where a lot of the innovation in modern computational drug design is found. One such approach is using DRL to find the areas in the molecular search space that are likely to contain compounds that maximize the desired properties, represented by some reward metric (e.g., a QSAR model that gives a predicted activity value of a molecule against some biological target of interest). However, this approach is prone to a problem analogous to "tunnel vision" – the model can excessively fixate on one (or a small number) of molecules that generate the highest rewards, leading to a loss of diversity and limited exploration. There are several ways to combat this, such as adding diversity filters where we punish the model for generating overly similar molecules [20], and even EA can be employed (this method is discussed in detail in section 3.1). Despite the effectiveness of the aforementioned techniques employed to circumvent the exploratory limitations of RL, it is desirable to learn a distribution from which we could sample molecules around the most impactful modes of our reward function. In other words, we want to sample molecules from different areas in our reward landscape where the reward is high, and not simply exploit the best mode found so far. This is one of the primary use cases that GFN were designed for, making it particularly suited to drug discovery problems where we want to find as many diverse candidates as we can that meet our pre-defined criteria [30].

2.2.2 Definitions

Many of the definitions in this section will follow the notation used in the foundational GFN paper[31] for consistency. One can think of GFN as a Directed Acyclic Graph (DAG) representing a network of pipes, with particles flowing through each of the pipes. The network is constructed sequentially by taking actions from a given state s to get to a subsequent state s' reachable from s taking any available action $(s \rightarrow s') \in \mathcal{A}$, or equivalently, $s' \in Child(s)$. Importantly, because we are using a DAG for the network, we do not concern ourselves with cycles, and we maintain a (strict) partial order, i.e., $\forall s \in S \backslash \{s_f\} : s_t < s_{t+1}$. The collection of all states in the order they were reached is called a trajectory, and will be denoted as $\tau \in \mathcal{T}$. We will

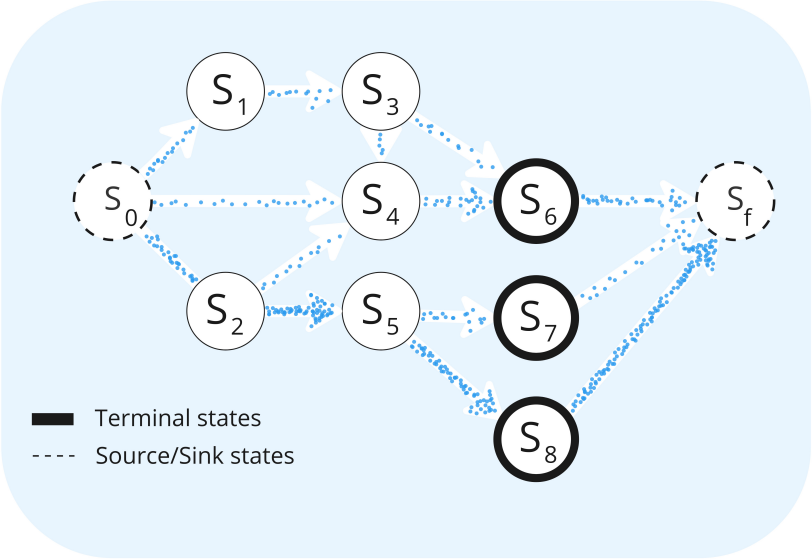

FIGURE 2.1: A simplified example of a pointed DAG as a flow network, where the blue dots represent flow. S_0 and S_f are the source and sink states, respectively, and the total flow theoretically goes through both. More particles going through an edge corresponds to paths likely to lead to high-reward terminal states, thus having a higher probability of being taken during sampling (e.g., the trajectory $\tau = (s_2 \to s_5 \to s_8)$ is most likely to be sampled in this example).

use special notation to disambiguate the completion status of the trajectories. For example, partial trajectories (i.e., not starting from the initial state or not ending in a terminal state) will be explicitly be written out as $\tau = (s_a, ..., s_b)$ where either $s_a \neq s_0$ or $s_b \neq s_f$. A generic set of trajectories will be denoted as simply $\mathcal{T}$. A DAG in this context can be formally defined as a graph where, given a set of trajectories, $\mathcal{T}$ there are no trajectories $\tau = (s_1, ..., s_n)$ where $s_1 = s_n$, except for a trajectory consisting of exactly one state. Additionally, these DAGs have one specific requirement – all the particles must originate from a single initial state (source), and all these particles will also go from all terminal states into a single final sink, as shown in Figure 2.1. We will henceforth denote the source as s_0, and the sink as s_f. This is known as a pointed DAG.

The justification for the "Flow Network" component in "Generative Flow Network" comes from augmenting the aforementioned pointed DAG with a (trajectory) flow measuring operator F, which is defined as any function that maps a set of complete trajectories to a non-negative value ($F : \mathcal{T} \mapsto \mathbb{R}_{\geq 0}$), forming the pair (DAG, F), known as a flow network. Given this definition, we can now define how to measure flow through a state or edge as shown in Eq. (2.1) and Eq. (2.2), respectively.

$$F(s) := F(\tau \in \mathcal{T} : s \in \tau) = \sum_{\tau \in \mathcal{T} : s \in \tau} F(\tau) \tag{2.1}$$

$$F(s, s') := F(\tau \in \mathcal{T} : s \to s' \in \tau) = \sum_{\tau \in \mathcal{T} : s \to s' \in \tau} F(\tau) \tag{2.2}$$

Following this analogy, the amount of particles flowing through each pipe represent our current estimate of how promising each path through our network is, in terms of

reward. Because there is a source and sink node that all particles must flow through, and the particles represent units of reward, it follows that the total reward in the network is equal to the flow going through S_0 at the start ($t = 0$) and through S_f at the end ($t = T$). This can be rewritten as $F(s_0)$, the total flow going through the initial state, and is known as the initial state flow (by extension this is also true for $F(s_f)$). As stated prior, this initial state flow equals the total flow, which is denoted as Z in GFN literature. These statements can be summarized into the equality shown in Eq. (2.3).

$$F(s_0) = \sum_{\tau \in \mathcal{T}} F(\tau) = Z = F(s_f) \tag{2.3}$$

Importantly, because Z is the total flow going through the network, it can be used as a partition function. This allows us to associate trajectory flows with probability measures, as in Eq. (2.4). In GFN literature, special notation is used for conditional probabilities – P_F and P_B – used to define the probability of transitioning from one state to another, in the forward and backward direction, respectively (see Eq. (2.5) and Eq. (2.6)). These conditional probabilities are analogous to a stochastic policy in the context of RL, e.g., $P_F(s \rightarrow s'|s)$ can be interpreted similarly to $\pi(a|s)$. This association subsequently enables the definition of Markovian Flows, which is a flow that adheres to Eq. (2.7) for (1) any state s, (2) edge $(s \rightarrow s')$ and (3) trajectory τ ending in s. As the name implies, each particle that comprises the flow in our GFN have behaviours that are only dependent on their current state. This is important, because it invokes a setting wherein we are dealing with conditional independence. Additionally, it is important to remember that because of the pointed DAG property, we know that the probability of visiting the source state $Pr(s_0) = 1$. From this follows that the probability of sampling a completed trajectory $\tau \in \mathcal{T}$ can be obtained from the product of the probabilities of each step taken in that trajectory, as shown in Eq. (2.8) and Eq. (2.9). Note that this only holds for complete trajectories, that is, trajectories that satisfy $\tau = (s_0, ..., s_f)$.

$$Pr(\tau) := \frac{F(\tau)}{\sum_{\tau \in \mathcal{T}} F(\tau)} = \frac{F(\tau)}{Z} \tag{2.4}$$

$$P_F(s'|s) := Pr(s \rightarrow s'|s) = \frac{F(s \rightarrow s')}{F(s)} \tag{2.5}$$

$$P_B(s|s') := Pr(s \rightarrow s'|s') = \frac{F(s \rightarrow s')}{F(s')} \tag{2.6}$$

$$Pr(s \rightarrow s'|\tau) = Pr(s \rightarrow s'|s) = P_F(s'|s) \tag{2.7}$$

$$Pr(\tau) = \prod_{t=1}^{n} P_F(s_t|s_{t-1}) \tag{2.8}$$

$$Pr(\tau) = \prod_{t=1}^{n} P_B(s_{t-1}|s_t) \tag{2.9}$$

The theory states that well-trained GFN can be used to sample objects in proportion to their associated goodness, given by $\pi(x) \propto R(x)$, where x is the sampled object, and R is some reward function that associates x with a non-negative reward ($R(x) \geq 0$ or $R : \mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$). In line with the existing literature, we also use the notation $\pi(x)$ here to signify the probability of sampling an object under the GFN's policy and reward conditions. The intuitive interpretation hereof is that the model becomes increasingly likely to generate molecules that have the desired properties as it trains. When thinking of reward in terms of the particles flowing through our network, we can use the previous equations to derive that $R(x) = F(\tau)$ (x is sampled from a completed trajectory). We can plug this into Eq. (2.4) to obtain Eq. (2.10), which can be interpreted as: an object is sampled with probability approximately equal to its reward divided by the total reward going through the network.

$$\pi(x) \approx \frac{R(x)}{Z} = \frac{R(x)}{\sum_{x' \in X} R(x')} \tag{2.10}$$

For completeness in terms of proofs and derivations of the above-mentioned equations, we refer the reader to the GFN foundations paper by Bengio et al.[31]

2.2.3 GFlowNet training objectives

In the original paper that first introduced GFN, the objective to learn a model that could sample $\pi(x) \propto R(x)$ was intuitively a flow-matching approach[30]. The authors posit that, given our DAG as a flow network, we need to find a flow where all flows are conserved, and the flow going into each terminal state should be equal to the reward for the respective terminal state. The former can be achieved through assigning a strictly positive estimated flow $F(s, s')$ to each edge (s, s'), and respecting the following balance:

$$\forall s' : \sum_{(s,s') \in \mathcal{E}} F(s, s') = R(s') + \sum_{(s',s'') \in \mathcal{E}} F(s', s'') \tag{2.11}$$

where $\mathcal{E}$ is the set containing all edges (s, s'), and $R(s') > 0$ if s' is a terminal state, and 0 otherwise. Assuming we achieve the balance defined in Eq. (2.11), we can define and use the forward policy as in Eq. (2.12) to sample completed trajectories from our flow network with a probability proportional to their reward.

$$P_F(s'|s) = \frac{F(s, s')}{\sum_{s''} F(s, s'')} \tag{2.12}$$

Since then, other objectives that have more favourable properties such as faster convergence rates and increased stability have been proposed (e.g., Trajectory Balance (TB) and Sub-Trajectory Balance (SubTB)), which are also the objectives used in our experiments. The reader is therefore encouraged to read the original paper[30] for

more detail on the flow matching implementation and the proofs that mathematically guarantee that it satisfies the properties necessary to train a model that can sample x with $\pi(x) \propto R(x)$. Because most non-trivial problems are likely to have copious amounts of states (and therefore result in large partition functions within the flow networks), we use a logarithmic approach in practice to make it computationally viable to compute the loss when training a GFN using the flow matching (and any subsequently discussed) objective, as will be shown in the following sections.

Trajectory Balance

TB was introduced as a training objective in the 2022 paper by Malkin et al. [32] empirically demonstrate that propagating reward signals through the network can be done much more efficiently than the original approach by considering the entire trajectory at once. More rigorously, using the equations defined above, for a complete trajectory where x is an object sampled from a complete trajectory $\tau = (s_0, ..., s_n = x)$, and $Pr(x) = \frac{F(x)}{Z}$, we get the balance shown in Eq. (2.13).

$$Z \prod_{t=1}^{n} P_F(s_t|s_{t-1}) = F(x) \prod_{t=1}^{n} P_B(s_{t-1}|s_t) \tag{2.13}$$

As we use the reward signal in this network as our flow function, we can replace $F(x)$ with $R(x)$ in our equation. Furthermore, to make this balance actionable to train the GFN, it is turned into an objective that we can optimize as shown in Eq. (2.14). Importantly, when training these models in practice we do not have complete information on all possible states, and must therefore estimate the forward and backward policies for these states using parameters θ. We denote such policy estimates under model parameters as $P_F(-|s;\theta)$ and $P_B(-|s;\theta)$, respectively. Additionally, the partition function is also estimated (Z_θ), and together with the estimated forward policy, determines an (estimated) Markovian flow (F_θ).

$$\mathcal{L}_{TB}(\tau;\theta) = \left(log \frac{Z_\theta \Pi_{t=1}^{n} P_F(s_t|s_{t-1};\theta)}{R(x)\Pi_{t=1}^{n} P_B(s_{t-1}|s_t;\theta)} \right)^2 \tag{2.14}$$

When this objective is satisfied, i.e., the estimated Markovian flow results in $F_\theta(x) = R(x)$ for all possible x, then it can be shown that $\mathcal{L}_{TB} = 0$ for all complete trajectories[32], and vice-versa. This then satisfies the GFN requirement of sampling with $\pi(x) \propto R(x)$.

Sub-Trajectory Balance (λ)

Building on top of the previous GFN training objectives, [33], Madan et al. proposed a version of trajectory balance that learned from partial (sub) trajectories and thereby have more control over the training stability and convergence rate. The λ hyperparameter is used to determine how much weight the algorithm assigns to path lengths. As λ approaches infinite, SubTB(λ) essentially only considers complete trajectories, which is similar to TB. Conversely, when λ approaches 0, SubTB(λ) assigns more weight to local states. Finally, setting λ to 1 equates to valuing all trajectory lengths equally. Furthermore, SubTB as an objective is not dependent on estimating Z, using only the estimated flow F_θ of a (partial) trajectory, as shown in Eq. (2.15).

$$\mathcal{L}_{SubTB}(\tau;\theta) = \left(log \frac{F_\theta(s_m)\Pi_{i=m}^{n-1} P_F(s_{i+1}|s_i;\theta)}{F_\theta(s_n)\Pi_{i=m}^{n-1} P_B(s_i|s_{i+1};\theta)} \right)^2 \tag{2.15}$$
$$\text{where } \tau = (s_m \rightarrow ... \rightarrow s_n)$$

$$\mathcal{L}_{SubTB}(\lambda,\tau;\theta) = \frac{\Sigma_{0\leq i<j\leq n}\lambda^{(j-i)}\mathcal{L}_{SubTB}(\tau_{i:j};\theta)}{\Sigma_{0\leq i<j\leq n}\lambda^{(j-i)}} \tag{2.16}$$

The complete SubTB(λ) is defined in Eq. (2.16). It considers all $\binom{length(\tau)+1}{2}$ unique possible sub-trajectories of a **complete** trajectory. Every iteration, the loss for a sub-trajectory is computed and multiplied with the accumulated λ value raised to the power of the length of the evaluated sub-trajectories, before being divided by the same amount to ensure that we end up with a convex combination of $\mathcal{L}_{SubTB}$ losses to use for gradient updates. As mentioned prior, plugging $\lambda = 1$ into Eq. (2.16), then, leads to a uniform weighting scheme.

The authors hypothesize, and empirically demonstrate, that SubTB leads to variance reduction during training compared to previous objectives. They also posit that it can learn faster in problems where the states near the end of the trajectories far out-number states near the initial state. The explanation given is that the estimated state flow function $F_\theta(s)$ can generalize quickly between states and thereby attenuates the negative effects of sparse signals at the near-terminal states. For in-depth explanations and experiments in various scenarios using SubTB, we refer the reader to the paper by Madan et al.[33]

Chapter 3

Related Work

3.1 DrugEx

The first version of DrugEx (DrugEx v1) was introduced as a SMILES-based RNN that was trained using RL to generate molecules by repeatedly appending a new token given a sequence (i.e., autoregressively)[34]. The novelty it proposed was a unique exploration strategy to promote diversity in the generated molecules. In addition to the generator itself (G_θ), it used a second model (G_ϕ), which is initially a clone of the generator with frozen weights. In the paper, these models were termed the "exploitation network" and "exploration network", respectively.

The exploitation network G_θ, is initially trained on a large ($> 1M$) set of SMILES, sampled from a chemical space that is assumed to contain drug-like molecules. This is done to learn the syntax to generate (mostly) valid SMILES in the relevant domain. After this initial training phase, G_ϕ is created using these weights, while G_θ is further trained on a set of SMILES representing molecules that, ideally, have properties that we are interested in optimizing. Before passing them to the model, the SMILES are pre-processed – they are all tokenized, appended by a stop token (here <EOS>) to indicate the end of the sequence, and padded with dummy tokens to reach a fixed length if necessary.

Importantly, during the RL training phase, a hyperparameter ϵ is considered that determines the probability with which the exploration network G_ϕ decides which token to add next given the current sequence of tokens. G_ϕ does not have its weights updated during training, ensuring that it keeps sampling from a much broader chemical space compared to the one G_θ is sampling from after fine-tuning and optimizing for the given objective. This was shown to increase the diversity of the generated candidates compared to other RL-based molecular generators at the time. In DrugEx v1, molecular diversity was defined as the average of the Tanimoto distance metric between every molecule pair in the set of molecules M, as follows:

$$Diversity(M) = \frac{1}{|M|^2} \sum_{(m_1,m_2)\in M\times M} Dist_{Tanimoto}(m_1, m_2)$$

For the creation of their dataset, 1.018.517 SMILES that met the following criteria were sourced from the 15^{th} version of the ZINC database[35]:

- $-2 < \hat{logP} < 6$
- 200 $<$ molecular weight $<$ 600

Additionally, they sourced ligands with known activity against the adenosine A2A receptor ($A_{2A}R$) from the 23^{rd} version of ChEMBL to (1) fine-tune the generator by biasing it towards the $A_{2A}R$ ligand chemical and (2) create biological activity prediction QSAR models to score the generated candidates downstream[36]. For their QSAR implementation, a Random Forest (RF) regression model was trained on the extracted ligands to score the generated molecules.

In DrugEx v1, to train G_θ, the objective function J parameterized by θ for the RL algorithm to maximize was formulated as follows:

$$J(\theta) = E[R(y_{1:T}|\theta)] = \sum_{t=1}^{T} log G_\theta(y_t|y_{1:t-1})(Q(y_{1:T} - \beta))$$

where Q is the above-mentioned RF-based QSAR model, R is the reward associated with a complete (valid) sequence, t is the current time-step, with T being the terminal step when the <EOS> token is sampled, and β is the minimum required reward, i.e., if the model generates a molecule that receives a reward below β, it is punished with a negative reward. A policy gradient approach was used to update the weights of G_θ given the expected reward.

DrugEx v2 updated the concept of the first version by adopting a MOO approach during the RL training phase – the adenosine A1 receptor (A_1AR) and the human Ether-à-go-go-Related Gene (hERG) were added as extra objectives for optimization. In addition to MOO, it also extended the idea of using multiple networks to improve diversity – There was now a third clone of G_θ, which we shall denote by G_ψ. Both G_θ and G_ψ started with the weights resulting from fine-tuning, while G_ϕ retained the frozen weights before fine-tuning, and like before, was not updated throughout the RL training phase. Here again, the exploration hyperparameter ϵ was used as the probability for G_ϕ to provide the next token. With probability $1 - \epsilon$, the next token would be decided by the combined output from G_θ and G_ψ. G_θ was updated on every step, while G_ψ was updated periodically based on a user-determined schedule. In this way, the authors reproduced a system analogous to an evolutionary algorithm, where G_θ is the agent, G_ψ would be the crossover operator and G_ϕ the mutation operator.

The definition of the molecular diversity was also updated to the following modified Solow-Polasky diversity measure[37]:

$$Diversity_{SolowPolasky}(M) = \frac{1}{n} \mathbb{1}_n^{\mathbf{T}} A^{-1} \mathbb{1}_n$$

where $n = |M|$ is the size of the set of molecules M, and $\mathbb{1}_n$ is used in this context to denote an all-ones vector of size n. A is a non-singular $n \times n$ matrix containing the output resulting from calculating and transforming the Tanimoto distance between each molecule pair (d_{ij}) in M. The transformed Tanimoto distance $d_{ij}*$ for each molecule pair is obtained by scaling said distance and raising Euler's number by it as follows: $d_{ij}* = e^{-\xi d_{ij}}$, where ξ is a strictly positive, non-zero integer. A bigger ξ value leads to less strict distance computation. We will also be using the Solow-Polasky diversity measure in this paper to determine the diversity of the molecules proposed by our models. Similarly to the original DrugEx implementation, we will adhere to a ξ value of 10.

The combination of the multiple objectives in this paper was achieved through several weighting schemes. Here we only consider the scheme termed "Pareto Front scheme" that we also used in our implementation of DrugEx. This scheme operates

on a so-called desirability score that is calculated for all i objectives. If the reward for that objective is bigger than some threshold t between 0 and 1 (here 0.5 for all objectives), then desirability is set to 1 (molecule is labelled *desirable*), otherwise the molecule is labelled *undesirable* and its desirability is set to the reward (which is also between 0 and 1) divided by said threshold. The molecules are divided into sets, sorted by the layer of Pareto-dominance that they belong to (highest to lowest) and subsequently ranked by Tanimoto distance, where a larger distance resulted in a higher ranking, denoted by an index k. Finally, the flat reward used as the reward signal for a molecule x is computed as follows:

$$R(x) = \begin{cases} \dfrac{k - N_{undesirable}}{2N_{desirable}}, & \text{if } \forall i : R_i(x) \geq t_i \\ \dfrac{k}{2N_{undesirable}}, & \text{otherwise} \end{cases}$$

3.2 Multi-Objective GFlowNet paper

In the original GFN paper [30], the framework was proposed using a simple reward single source. Thereafter, the original authors released a paper in which the theoretical foundations for GFNs were laid out formally [31], where they included a segment on conditional GFNs, and extended this to Pareto GFNs for multi-objective settings. The authors suggested that given d objectives, one could sample trajectories from the Pareto front by, e.g., assigning convex weights to the reward R_i for each objective, where the weights would sum to 1, as formalized in Eq. (3.1). This approach was termed "The Pareto additive terminal reward function", and is also the approach used in this paper for the multi-objective setting. What makes this particularly interesting for molecule generation purposes, is that a GFN trained in this manner, will be able to generate samples that are more focused towards a certain objective depending on how much weight is assigned to it. This makes it exciting in terms of exploring how properties of a given molecule can be optimized with respect to all the others, and gives researchers more control in their search through the vast chemical space.

$$\forall i : R_\omega(s) = \sum_i^d \omega_i R_i(s) \tag{3.1}$$

$$\text{where } \omega \in \{\omega \in \mathcal{W} \subset \mathbb{R}^d : \omega_i \geq 0, \sum_i^d \omega_i = 1\}$$

Among other things such as DNA-sequence design, the authors also experimented with fragment-based small molecule generation, albeit with different objectives (Soluble epoxide hydrolase (sEH), SA Score, QED and molecular weight) than ours. Their findings suggest that the Multi-Objective GFlowNet (MOGFN) model respects the conditioning, i.e., the reward for a specific property tends to increase proportionally to the fraction of the weight assigned to it. Conversely, the closer the weight is to 0 for a property, the more "random" its associated reward is. Note that it is still entirely possible to generate molecules that give high reward for that property, but it becomes decreasingly likely. Furthermore, the model retains the property of diverse candidate generation and does not fixate on singular modes, remaining in line with the GFN theory.

For the above-mentioned reasons, we built on top of this research to conduct our own experiments. Our approach is mainly differentiated through the inclusion of retrosynthetic accessibility information and multiple QSAR-computed objectives instead of one. We also uniquely compare these results to the output of an autoregressive LSTM model using SELFIES, while in the original paper the focus was on algorithm comparison. We further elaborate on the methods used for sampling and incorporating conditioning information into the MOGFN in chapter 4.

3.3 AiZynthFinder & RA Score

Our primary source of retrosynthetic accessibility information comes from the open-source CASP-tool AiZynthFinder, developed by MolecularAI[19]. The tool computes retrosynthetic routes recursively until certain criteria are met, e.g., all predicted precursors were found in a database of purchasable compounds. At the time of writing, the core is based on the Monte Carlo tree search (MCTS) approach presented in the 2018 paper by Segler et al.[38], where a DNNs is used to guide the tree search. When a node is selected in the tree search, it is expanded by the aforementioned DNNs to add new positions.During expansion, the compound of interest is first converted to a molecular fingerprint and then fed to the network. The output of the network is a probability distribution over all available transformations. Subsequently, the most promising positions are subject to the k most likely transformations from the probability distribution supplied by the DNNs, which continues until a stopping criterion is met, e.g., all the precursors suggested by the network are available in a list of user-defined options (stock). The reactions resulting from applying said transformations are then filtered based on another neural network that determines how likely these reactions are. Importantly, where the AiZynthFinder implementation differs from the one described in Segler et al., is that this final filtering step is omitted. The tree then updates the position values based on the rollout and moves to the selection phase once more until completion. Finally, the potentially solved path found by AiZynthFinder is returned to the user.

In addition to presenting a nuanced path to solve retrosynthetic routes for molecules, the output of AiZynthFinder also includes information such as:

- number of (solved) routes that were found
- number of precursors (in stock)
- a state score $\in [0.0, 0.1]$
- number of transformations applied

For researchers interested in training models to generate novel molecules that can be reasonably synthesized in a lab, such information is invaluable. Retrosynthetic analysis is a more robust way to verify a molecule's synthesizability than other simple heuristics, such as a molecule's similarity to known synthesized examples, SMILES length, and so forth. However, as mentioned prior, the analysis is expensive, and for the training of models where many calls to AiZynthFinder directly would have to be made for numerous molecules per iteration, this quickly becomes intractable. For this reason, Thakkar et al. developed and published RA Score, a score based on the state score given by AiZynthFinder[39]. The score can be seen as a confidence score: how accessible AiZynthFinder "believes" this molecule is, under its current policy, and the available stock. Conveniently, it also paves the way to inexpensively incorporate complex retrosynthetic analysis information into a training loop. By training

a (pseudo)classifier on roughly 400,000 AiZynthFinder results, RA Score was shown to be adept at approximating AiZynthFinder's results without explicitly performing the full analysis. Simply learning through mapping comes with its caveats, namely:

1. Examples that are vastly different from the training set may deviate too much from AiZynthFinder's actual results
2. It may ignore intricacies such as chirality when determining if a molecule is accessible
3. It inherits all the underlying limitations of the AiZynthFinder instance that was used to generate the training data, such as a false positive resulting from using a limited stock.

Interestingly, some of these do have their advantages. For point 2, it is entirely possible that a slight deviation in the molecule conformation is not relevant for the actual accessibility, even if a path could not be found directly by the underlying CASP tool. As mentioned in the introduction, point 3 can be used to purposefully limit the scope of the model's exploration to the researcher's specific circumstance. We encourage the reader to explore the AiZynthFinder documentation and read the RA Score paper by Thakkar et al. to appreciate their full applicability to de-novo drug design. For our experiments, we use AiZynthFinder as it is supplied by default at the time of writing: a pretrained policy with the default ZINC stock and USPTO reaction templates[1].

[1]https://github.com/MolecularAI/aizynthfinder

Chapter 4

Methods

GFNs can be described as a technique where the generative process is viewed as a DAG that represents a flow network. Within the context where we want to generate discrete objects such as a molecule, the generation process could be seen as a DAG where each node is a state of the molecule being built, and every child node is a molecule with, e.g., some arbitrary atom added to it, with every child node being unique w.r.t. the other child nodes. In the case of modern drug development, we must take multiple targets into account, as a drug with a strong biological activity on a desired target is rendered useless if it has unintended adverse side effects. Factors such as synthetic accessibility, as mentioned prior, are also important considerations, as a drug is only useful if we can actually produce it. Therefore, for GFNs to be truly practical within this field, it is necessary to be able to train them in a manner that attempts to optimize several objectives concurrently, as was highlighted in section 3.2.

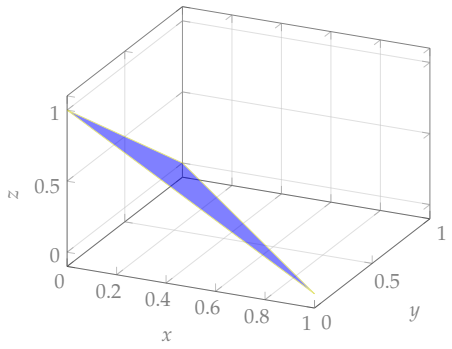

FIGURE 4.1: Example of simplex where preferences ω are sampled from in the case of 3 objectives.

4.1 Conditioning GFlowNet training objectives

To condition our GFNs, we continuously sample d preferences, denoted as ω, from a simplex (see Figure 4.1) to ensure that they sum up to 1. The manner in which the preferences are sampled are based on a Dirichlet distribution with a parameter α, a vector of real numbers that determine where the focal points of the simplex are. In our experiments, we adhered to uniform sampling (i.e., setting α to a vector of ones) to ensure all objectives get equal amounts of coverage in terms of preference. These preferences are used as the weight assigned to each objective that the GFN is to optimize. An objective that is associated with a higher preference value contributes more to the flow moving through the DAG as shown in Figure 2.1, and can be visualized as an extension of that as shown in Figure 4.2. In this case, if the preference

value of B (red particles) were to be greater than that of A (blue particles), the GFN would be more likely to sample the paths with the red particles.

The manner in which the final reward is calculated was shown in Eq. (3.1), and is also shown in Figure 4.3 as $(R(x) = (\omega_1 \times R_1(x), ..., \omega_d \times R_d(x))^\beta$. The β here is a parameter that makes the reward landscape "peakier". In practice, this means that the lower-values modes are flattened more than the higher-valued modes as the exponent grows, and our GFN is forced to find higher modes. This of course also means that the reward signals become more sparse, but if found, we expect to see an increase in the overall rewards associated with the GFN's generated molecules.

To actually implement these preferences into our training routine for the GFNs, we need to update the training objectives to make all flow-related estimates depend on said preferences. We then end up with Preference-Conditioned (PC)-GFN, as it is called in the original MOGFN paper by Jain et al.[40], where the TB balance equation is transformed from Eq. (2.13) to Eq. (4.1). Note that the loss function is now dependent on the preferences, as is the case for the total flow estimate Z_θ, reward function and the forward/backward policies. We similarly update the SubTB algorithm to achieve the same effect (see Eq. (4.2) and Eq. (4.3), albeit with a slight difference, as here we condition the state flow estimate F_θ on the preferences.

We will compare the performance of the PC version of both training objectives to assess (1) if SubTB(λ) can be trivially extended to the multi-objective setting, and (2) if its empirically demonstrated properties (i.e., faster convergence and stable learning) are maintained in this setting.

4.1.1 Encoding conditioning information

To be able to use the conditioning information to condition the rewards signal given to the GFN, we must encode it before adding it to the rest of the graph data. There exist several simple approaches to encode this information, such as one-hot encoding. In the paper by Jain et al. [40], that investigated MOGFN, they used thermometer encoding, initially introduced by Buckman et al.[41], which has the benefit of preserving positional information. As such, we used the same encoding algorithm in this study as well.

$$\mathcal{L}_{TB}(\tau, \omega; \theta) = \left(log \frac{Z_\theta(\omega) \Pi_{t=1}^{n} P_F(s_t|s_{t-1}, \omega; \theta)}{R(x|\omega) \Pi_{t=1}^{n} P_B(s_{t-1}|s_t, \omega; \theta)} \right)^2 \tag{4.1}$$

$$\mathcal{L}_{SubTB}(\tau, \omega; \theta) = \left(log \frac{F_\theta(s_m|\omega) \Pi_{i=m}^{n-1} P_F(s_{i+1}|s_i, \omega; \theta)}{F_\theta(s_n|\omega) \Pi_{i=m}^{n-1} P_B(s_i|s_{i+1}, \omega; \theta)} \right)^2 \tag{4.2}$$

$$\Updownarrow$$

$$\left(\log F_\theta(s_m|\omega) + \sum_{i=m}^{n-1} \log P_F(s_{i+1}|s_i, \omega; \theta) - \log F_\theta(s_n|\omega) + \sum_{i=m}^{n-1} \log P_B(s_i|s_{i+1}, \omega; \theta) \right)^2$$

$$\text{where } \tau = (s_m \rightarrow ... \rightarrow s_n)$$

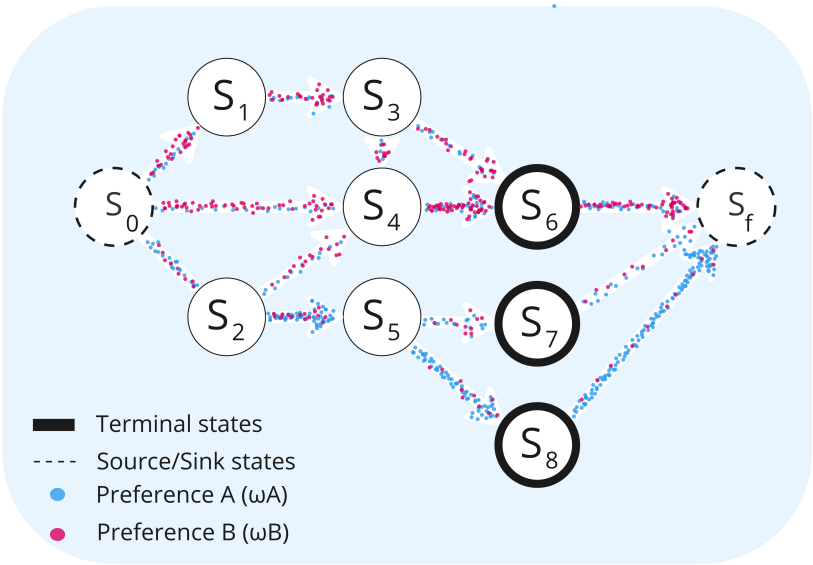

FIGURE 4.2: An example of an extended pointed DAG as a multi-objective flow network, where the red and blue dots represent the flow of their respective objective. The preferences prefixed by ω denote the weights assigned to their respective objective.

$$\mathcal{L}(\lambda, \omega) = \frac{\Sigma_{0 \leq i < j \leq n} \lambda^{(j-i)} \mathcal{L}_{SubTB}(\tau_{i:j}, \omega; \theta)}{\Sigma_{0 \leq i < j \leq n} \lambda^{(j-i)}} \tag{4.3}$$

4.2 Reward functions

4.2.1 AiZynthFinder proxy (RAScore port)

To create the reward function for retrosynthetic accessibility, we ported the best reported RAScore model architecture [42] from TensorFlow to PyTorch to align it with the rest of the models, and trained with the exact same parameters used in their DNNs predictor model. The results demonstrate that RAScore is a suitable proxy to AiZynthFinder, which is much more computationally expensive and thus infeasible for usage within learning loops where it would be called numerous times. RAScore's objective is to predict which molecules AiZynthFinder would find a solution for, and in practice it acts as a pseudo-classification model (the output layer is a softmax layer and therefore not guaranteed to be exactly 0 or 1) that outputs values that can be interpreted as binary indicators for retrosynthetic accessibility. The exact (hyper)parameters used for the RA Score model can be found in the appendix in Table B.

4.2.2 Synthetic Accessibility Score

The SAScore implementation that was used in this study is the standard distributed implementation (last modified in September 2013) by Ertl et al., based on the paper that introduced the score [15]. The pre-calculated fingerprint score file distributed alongside the code (used to instantiate the SAScore module) was used.

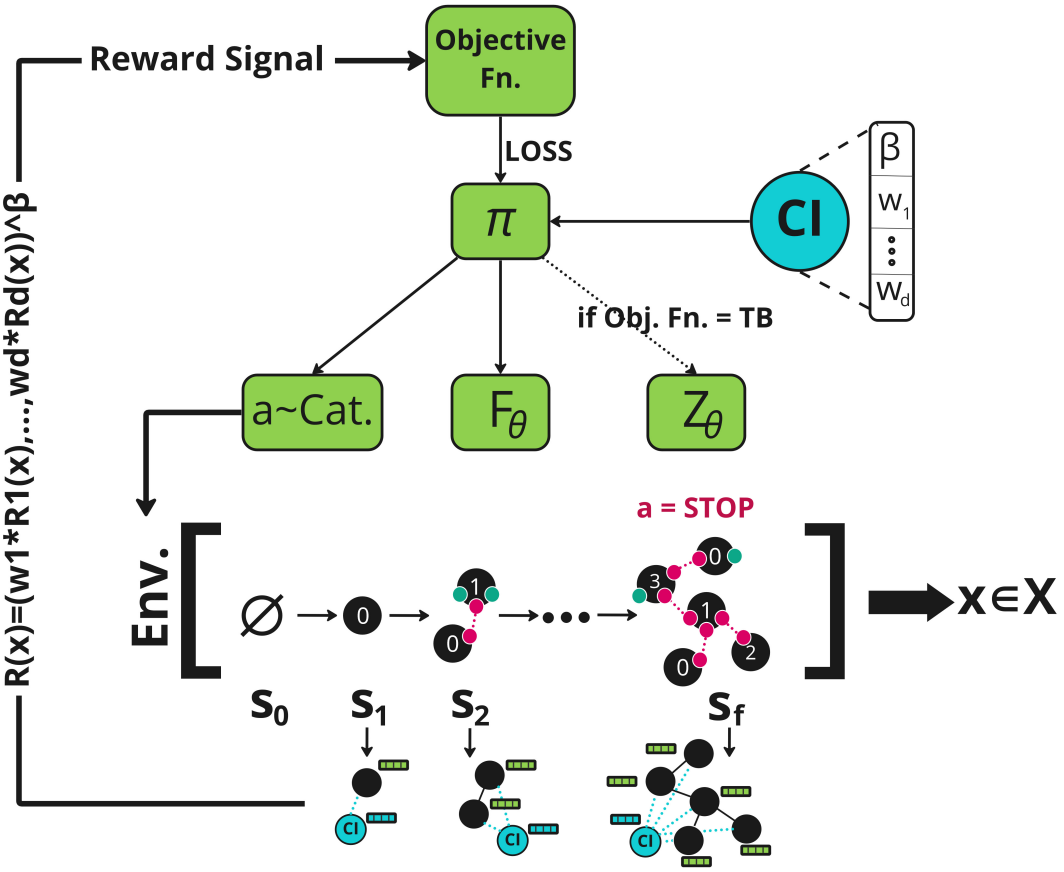

FIGURE 4.3: A high-level overview of the generative loop using GFN (denoted as π in the figure) in this study. The algorithms available were Trajectory Balance (TB) and SubTB. The conditioning information (CI) was used to weigh the rewards for different objectives and exponentiate the rewards. The GFN produces categoricals containing actions and their logits, and also the flow predictions. The categoricals are sampled at each step to produce an action to perform in the graph environment, until the termination action is sampled. The final (complete) graph ($x \in \mathcal{X}$) is then turned into a molecule before calculating its reward, which is then used for the learning step.

4.2.3 Biological activity prediction

For the prediction of biological activity of the generated molecules, which was used as the reward signal, we used the same approach as presented in DrugEx V2[9]. Specifically, we used Scikit-learn's Random Forest Regressor model with default settings except for *n_estimators*, which was set to 1000. We also included the negative examples, which were artificially scored at just below the lowest value in the dataset (here, 4) as this was shown to aid the regressor in properly detecting false positives. One notable difference is that we used ChEMBL version 26 instead of 23 for the training data, consisting of known ligands for our objectives.

4.2.4 Score normalization

It should be noted that all scores used in our experiments were normalized to be on a scale of 0 to 1, inclusive, where higher is better. The following transformations were performed:

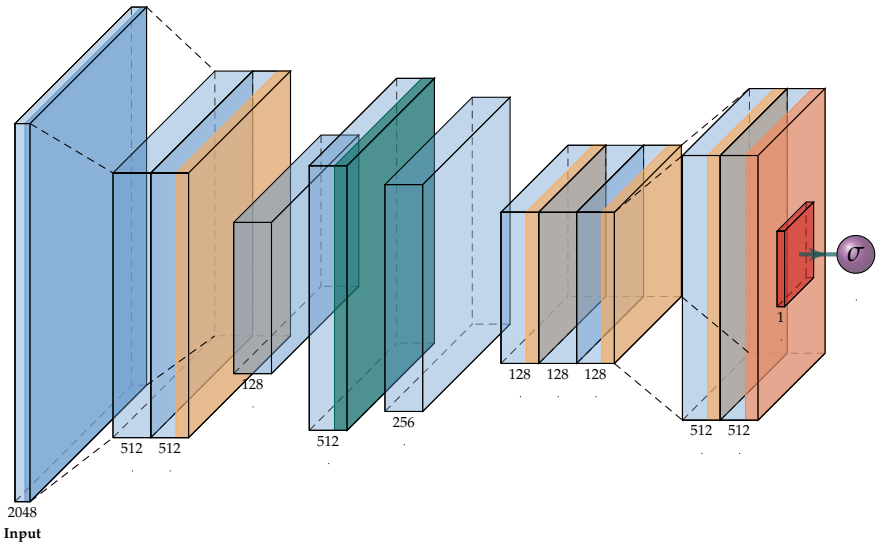

FIGURE 4.4: Best performing RAScore architecture from publication. Each box represents a layer in the model, with the input embedding size denoted below. The input is a 2048-bit feature-count fingerprint calculated with radius 3. The activation functions are denoted by the color of the right side of each box (if present). Orange=ReLU, Green=SeLU, Red=ELU. At the final layer, the embeddings are projected into a single scalar that is passed through a softmax layer to predict if a synthetic route could have been found by AiZynthFinder.

- $A_{2A}AR$: Divided by the 95^{th} percentile of the scores in its training set
- $A_{2A}AR$: Divided by the 95^{th} percentile of the scores in its training set
- $A_{2A}AR$: Divided by the 95^{th} percentile of the scores in its training set
- $RAScore$: Already is a number between 0 and 1
- $SAScore$: Subtracted from 10 and then divided by 9, i.e., $(10 - \text{SA Score})/9$
- Molecular Weight: Any value past 300 decays from 1 to 0, reaching 0 at 1000

4.3 Fragment-based environment

For the molecule fragments, we reused the 105 fragments (of which 33 are duplicates to include symmetry groups from the perspective of the model) used in the original GFN paper by Bengio et al. [30]. The SMILES for all 105 fragments along with the indices of their atoms where other fragments can be attached to (henceforth called stems) can be found in Figure B.1. Using a fragment-based approach intuitively results in a more stable searching environment – there are fewer steps necessary to end up with a drug-like small molecule. Additionally, given fragments that were extracted using chemical reasoning, one can ensure that any intermediate step during the assembly is a valid molecule [28]. Further benefits include straight-forward seeding of start molecules to bias the search space of generative models. These properties alone make fragment-wise generation of molecules an attractive option, and is the main approach used in our experiments as well.

Action space

The action space for this environment consisted of 3 discrete actions: (1) Adding a node to the graph, (2) Setting the attribute of an edge to indicate which stems should be involved in the bond between the fragments that the edge is connecting, and (3) a stop action to signal that the graph building process is done, thus completing the trajectory. These 3 actions are sampled from a categorical distribution containing separate logits for each action. A simplified visual aid is shown in Figure 4.5. Additionally, the logits are masked to prevent illegal actions from being taken. An example hereof is setting the logit of the stop action to a minuscule log value (-1e3 in our experiments) on the first t steps to ensure that graphs consist of at least t nodes. There is also the option of restricting the model to deciding which nodes to add and when to stop, while respecting the stem count of each node, making sure that a saturated node can never be sampled (i.e., disabling setting edge attributes). Removing saturated nodes as an option in the action space is also achieved through the hyper-minimization of logits by setting them to a small log-value, as was mentioned above. The approaches used for masking illegal actions is expanded on in section 4.3.

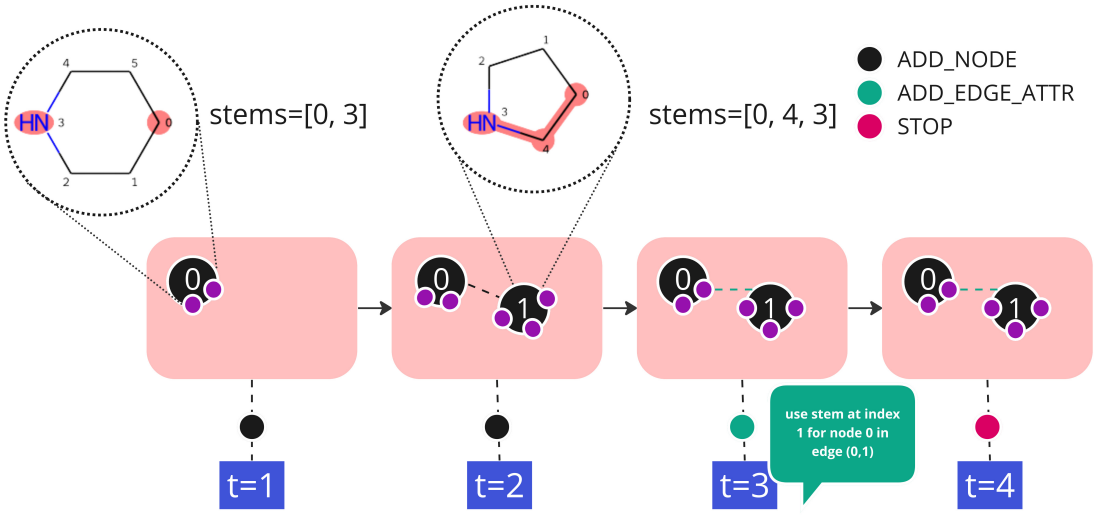

FIGURE 4.5: A high-level depiction of fragment-based graph construction through sampled actions. At the first step (t=1), a node ('node 0') is added (action ADD_NODE is sampled), which represents a molecule fragment with two stems where other fragments can eventually be added. At t=3, the action for setting edge attributes is sampled, and the model specifies that it wants the second stem of 'node 0' to be used in the edge with 'node 1'. The graph is then terminated at t=4 as the model sampled the stop action.

Adding a node is done by selecting a source node to attach the new node to, along with the index of the fragment that the new node represents. The nodes added using this action are initially agnostic of the stems involved in the new edge. The atom indices of the specific stems are *optionally* added afterwards as attributes on the relevant edge through the action that sets edge attributes, as it is possible that the

stop action is sampled before all edges have attributes set. The stems used between edges are specified once graphs are converted to their molecule representations.

Finally, we also have an exploration factor ϵ which allows for a random action to be applied on with probability equal to ϵ. This "coin-flip" happens for all trajectories separately at each time-step. Varying this intuitively results in our model exploring more, but with slower convergence as a trade-off.

Conversion from low-level graph to molecule

Once a graph is considered complete (i.e., the stop action was sampled and applied to its trajectory), it is converted to an RDKit molecule and sanitized before being passed into the scoring pipeline. For each edge, if the stem was not specified previously by the model through an edge attribute, then the first available stem is automatically picked to create the bond between the molecule fragments. Molecules that are considered invalid by RDKit render the trajectory that generated them illegal, which subsequently incurs a penalty (in this context, a negative reward value) for the generative model. This intuitively discourages the model from taking actions that result in illegal trajectories, but increasing the magnitude of the penalty too far may also negatively impact exploration. Illegal actions can come about in various ways, such as the model selecting stems to attach new fragments to that are already used. This is mostly handled by logit-masking as will be explained in the next section, however, we assume that there is still a small chance of the model producing molecules that do not comply with the sanitization method used.

Validity pseudo-guarantee through masking

As mentioned above, the generative models used in this study produce categoricals containing logits for each action type available in the environment. For the fragment environment, logits are computed for all fragments in the graph, regardless of the "legality" of the fragment in the context it's in. These logits correspond to a fragment's probability of being used as a source node for the next fragment. The same is true for the logits denoting probabilities of adding edge attributes, and the logits for terminating the graph building process. However, it would be quite inefficient for the model to learn both when actions are illegal *and* find candidates that maximize multiple objectives. To aid the model, the underlying graph contains information about the current action space in the form of masks. These masks are boolean tensors that are of the same shape as their corresponding logit tensors in the action space.

At each graph update step, the graph is evaluated and actions that should no longer be possible are marked. For example, to make sure we never violate valences, we first create a mask, i.e., a tensor the same size as the current number of nodes in the graph. The index of nodes that have a degree equal to the stem count of the molecule fragment that they represent are set to 0 – this corresponds to all stems of the current molecule fragment being occupied. An example hereof can be seen in Figure 4.6. During the creation of the categorical, we multiply this mask with the logits that were computed by the generative model. This sets all the logits where the mask equals 0, to a miniscule number, resulting in those options becoming virtually impossible to sample. If all nodes are masked, this simply means there are no stems left in the molecule – intuitively, this also means the add node action won't be sampled at all, which is what we desire.

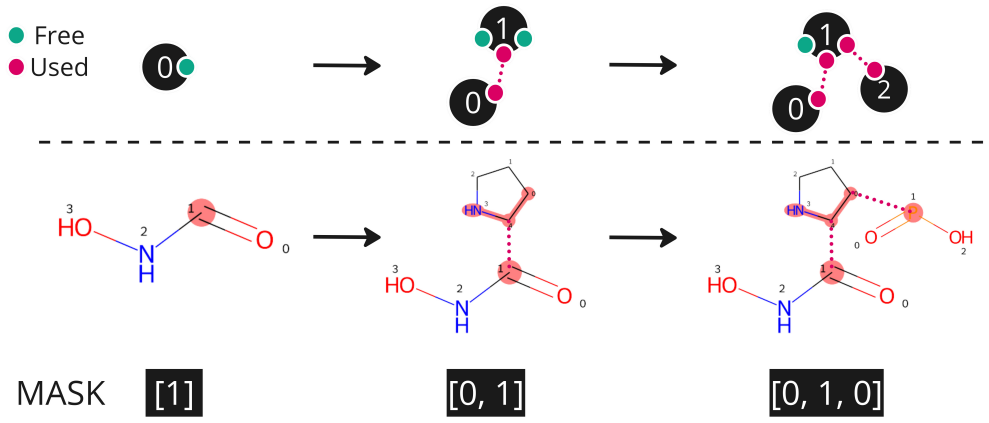

FIGURE 4.6: A simplified demonstration of the (source) node masking procedure during graph construction. At each step, all nodes in the graph are evaluated to assert that their total number of stems is less than their current degree. If the degree equals the number of stems for a given source node, the mask is set to 0 at its index. This prevents the model from attaching nodes to said node in subsequent steps.

As we also allow the model to specify which stems to use in the formation of a bond between two fragments, this can also result in illegal actions if left unchecked through selecting the same stem twice, or selecting any stem on a node that is already fully saturated. For this case, we create a zero-filled mask (all actions are initially illegal) of size ($|\mathcal{E}| \times 2$*max possible stems), where the number of max possible stems equates to the number of stems of the fragment with the most stems. We multiply this by 2, as we have to take into account the stems for both nodes involved in this edge. For every edge, the edge attribute is checked to see if it's already been set – if not, we set (for each half of the tensor, as it accounts for both nodes) a slice equal to the length of the number of stems a node has left to 1 in the mask. This handles two cases simultaneously: (1) The model can only add an edge attribute if it hasn't been set yet, which prevents cycles, and (2) the model will never be able to choose stems that are taken, preventing indexing errors.

Additionally, when allowing for random actions to be taken by setting the exploration factor $\epsilon > 0$, we also rely on these masks to ensure that the random actions are only sampled from the set of legal actions: $a_{random} \sim \mathcal{A}^*$. To enforce random actions when the condition is met, we invert the masking logic we have been using so far – we set the logits for all legal actions to an arbitrarily large value, so that the model will sample uniformly from these options.

With the above-mentioned masking approaches, we essentially guarantee that the model can not produce actions that are explicitly illegal, skipping the need for a "burn-in" phase where it needs to learn which actions should not be taken. Masking the stop action is trivial, and used for setting the minimum node count, but care should be taken here – in the edge case where the nodes sampled used up all stems before the minimum node count has been reached, this should be handled explicitly. Of course, this can also be leveraged to make the model biased towards producing longer graphs by, e.g., avoiding fragments with few stems early on.

Hyperparameter	Value(s)	Description
# Embeddings	128	The size of the embeddings.
# Transformer Layers	6	Dictates number of graph convolutions.
ϵ	0.05	The exploration factor: $p(a_{random} \in \mathcal{A}^*)$.
β	96	Reward exponentiation factor: $R(x\|\omega)^\beta$.
# CI dim.	$32 + \|\omega\|$	The dimensionality of the conditioning info.
Batch size	128	The batch size during training.
$\tau_{sampling}$	0.95	The lagging model sampling factor.
Min-Max Nodes	$2 - 9$	The min. and max. required nodes/graph.
Learning rate	5×10^{-4}	The learning rate used to train all GFlowNets.
λ	1	The trajectory length weighting factor (SubTB only).
α	$\mathbb{1}_{\|\omega\|}$	The Dirichlet parameters used to sample preferences.

TABLE 4.1: Fragment GFN initialization hyperparameters

4.3.1 GFlowNet architecture

The reward-conditioned GFN used in our fragment-based experiments consists of several components:

- A graph transformer for the construction of node and graph-level embeddings based on the CI.
- A linear layer that predicts the reward given CI embeddings, i.e., the conditional flow predictor ($\log F_\theta(s|\omega)$).
- An *optional* MLP representing $logZ_\theta(\omega)$ that predicts the initial state flow $F(s_0|\omega)$. Only used in algorithms relying on the initial state flow estimate (e.g., Trajectory Balance)
- An embedding layer that takes in a combination of node embeddings indexed by both "sides" of the edges to generate the edge embeddings. This approach is expanded on below.
- A linear layer (or MLP with ReLU activations between each layer) that outputs logits for each available action type given their embeddings.

A visual overview of the architecture of this GFN can be seen in Figure 4.7. The reason for the intricate approach to generating edge embeddings is two-fold: (1) The edges are sided in the sense that the nodes involved have their own unique set of stems to choose from. The edge embeddings therefore take into account both nodes by indexing the node embeddings by both edge directions, and summing them together to get the final features used to create the embedding. These embeddings are then concatenated with their respective indexed node embeddings before being passed to the output layer, producing 2 sets of logits – one corresponding to the source node stems, and the other to the destination node stems. These two sets are finally concatenated to produce the final logits for the action of setting edge attributes.

The hyperparameters used for the fragment-based molecule generation experiments closely follow the setup used in Jain et al. [40], and can be found in Table 4.1.

Graph Transformer

The current implementation of the GFN depends on an inner Graph Transformer (GT) for the creation of node and graph-level embeddings. These embeddings are

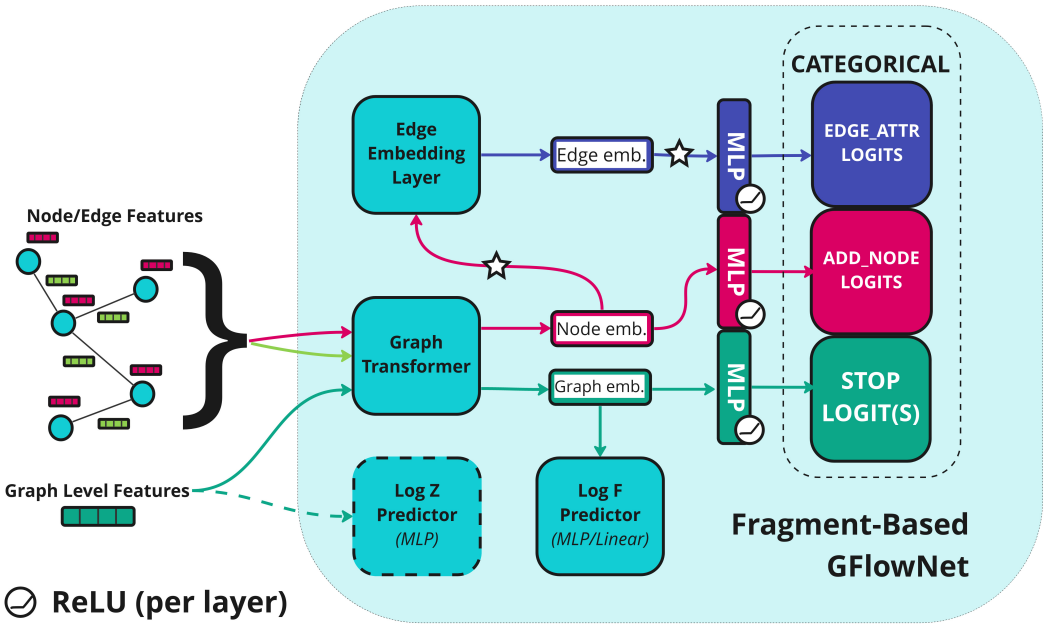

FIGURE 4.7: The general architecture of the GFN used in the fragment-based setting. The GFN contains an inner graph transformer model, and an (optional) logZ predictor and a flow/reward predictor (logF). The GFN accepts a graph along with CI passed as graph-level features, which the transformer generates embeddings for. The edge embeddings are created from the nodes at both sides of the edge and then summed (denoted by the star symbol). All embeddings are then passed through MLPs/linear layers to obtain the predicted logits for each corresponding action. These are then wrapped into a categorical object, where the actions can be sampled from.

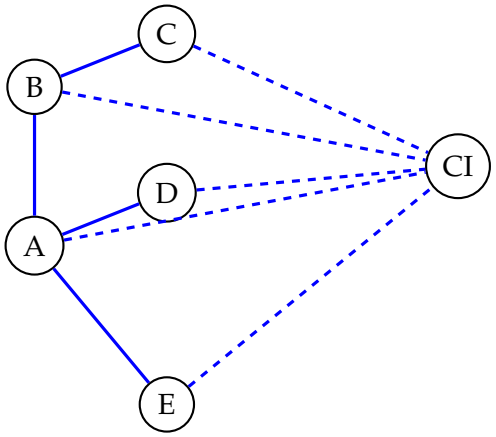

FIGURE 4.8: Schematic of how conditioning data is incorporated into the learning loop of GFNs. For each graph in a batch that is sampled, a virtual node is created that includes information such as the weights for each score used to generate a scalar reward for a generated molecule. This information is encoded into a single virtual node ("CI" in the figure) per batch, and is fully connected to all other nodes. The nodes (A-E) in this schematic represent placeholders for molecule fragments.

subsequently used to compute the logits to create the categorical that actions will be sampled from down-stream. A visual overview of the GT architecture is shown in Figure 4.9. The GT accepts three inputs, (1) edge features, (2) node features and (3) a feature-vector containing conditioning information (CI) such as the reward exponentiation factor β, and the preferences for each objective. All of these features are first passed into Multi-Layer Perceptron (MLP)s that return their intermediate embeddings. The intermediate node and CI embeddings are concatenated to form a new embedding, which we refer to as the *augmented node embedding*. Similarly, the intermediate edge embedding is also concatenated with a manually created (artificial) embedding, to create the *augmented edge embedding*. This artificial embedding essentially represents the virtual edges necessary to fully connect the CI node to the rest of the graph. The two augmented embeddings are then passed through the transformer layers (as seen in Figure 4.9) to output the node embedding containing CI as a result of the convolution. This output is then concatenated with the original CI embedding to form the final per-node embedding. It is also passed through a global mean pool layer, which is then concatenated with the final per-node embedding and CI embeddings to form the final graph-level embedding.

4.3.2 Sampling process

A high-level overview of the core generative loop used in this paper can be seen in Figure 4.3. The GFN contains an inner model that can perform graph convolutions. The inner models used in this paper were either Graph Transformers or Message-Passing Neural Networks. In addition to graph data, the GFN also accept a tensor containing conditioning information encapsulating data such as the reward exponentiation, and the weights associated with each objective. The reward exponentiation is useful for tuning the GFN to focus more on sampling candidates with higher Pareto-performance, but this comes at the cost of diversity, and vice versa [40]. The GFN used here can output (1) a categorical distribution containing all possible actions with their respective logits, (2) the (log) flow predictions $\log F_\theta$ between

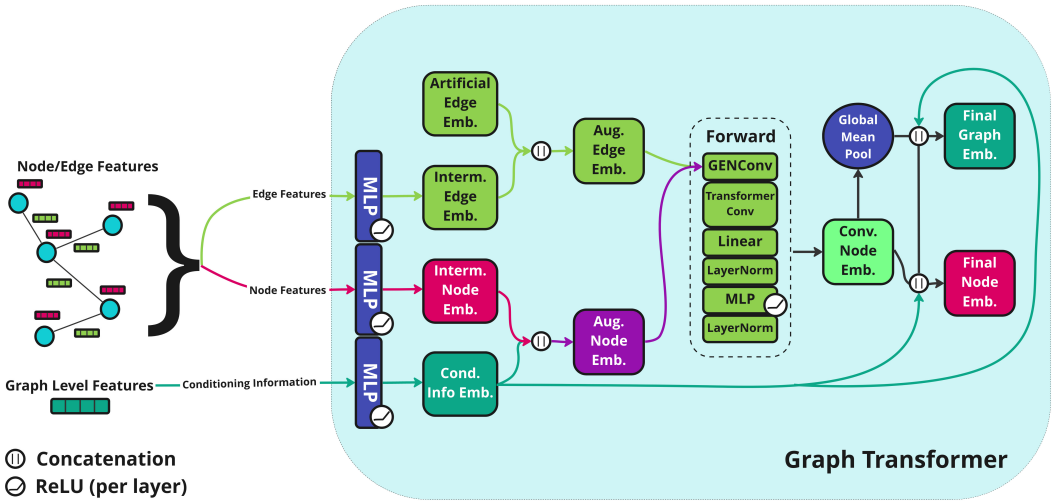

FIGURE 4.9: High-level overview of the GT architecture. The GT accepts node/edge features along with the CI, and creates intermediate embeddings for each of them. Then the edge embeddings and node embeddings are augmented by fully connecting a virtual node containing the CI to the rest of the graph. These embeddings are then processed by the transformer layer and then (1) the node embeddings are concatenated with the CI embeddings to obtain the final node embeddings, and (2) the final node embeddings are combined with the result of a global mean pool of the post-convolution node embeddings to obtain the final graph embeddings.

all states, and if using the Trajectory Balance algorithm, also (3) the (log) initial state flow prediction $\log Z_\theta \approx F(s_0)$.

At each step, the GFN receives the current state in graph data format, along with its conditioning information. These inputs are combined in the graph convolution process, producing the node and graph-level embeddings. The inclusion of conditioning information within the graph data is realized through the creation of an additional node (one per graph) containing this information, which is then fully connected to every other node in the graph. The graph-level embeddings are used to generate logits for all graph-level predictions, such as if the action for terminating a graph should be taken, or the flow (reward) predictions associated with the current graph. The node-level embeddings are used in e.g., predicting which node to add next, which node to add it to, and in the case where each node is a fragment, which atom of another fragment to add said node to. Logits for each action are stored as an order-preserving categorical distribution. From these categoricals, an action can be sampled that can be used to step through the environment, until the 'stop' action is sampled, at which point the graph is turned into a molecule (if valid) and given final scores for all d objectives. These rewards are turned into a final scalar reward and assigned to the terminal states, before being passed to the algorithm (e.g. Trajectory Balance), which calculates the loss to update the generative model's parameters.

4.4 Autoregressive (DrugEx) SELFIES environment

The DrugEx implementation used in our experiments only differs from DrugEx V2 in regard to the string representation of the molecules. After initial preprocessing

(e.g., removal of metals, chirality, etc.), the ChEMBL dataset is converted to SELFIES and tokenized using the SELFIES library built-in split functionality. The longest SELFIES string (after preprocessing) consisted of 109 tokens. The usage of SELFIES resulted in 100% of the generated molecules being valid, as expected. This also meant that certain checks that were in the original DrugEx V2 were unecessary, such as the validity of a molecule having an effect on the desirability, which would further affect downstream computation of the reward signal. Throughout the remainder of this paper, our DrugEx implementation will at times be referred to as **DrugEx+R** or **Drugex@SELFIES** in figures.

For two of our experiments (DrugEx V2 base task and extended with RA Score), we will use our implementation of DrugEx as a soft baseline to compare the performance and output of the GFNs. We use the term soft baseline for DrugEx, as a proper comparison between the two molecular generators can not be made. For example, DrugEx is not preference-conditioned, and is an autoregressive SELFIES generator. On the other hand, our GFN implementation can reach the same state from multiple paths, and the weighting scheme to transform the multiple objectives into scalars differs. Therefore, any comparisons using the soft baseline are purely to give insight into the differences between the two approaches, and can be seen as an assessment of GFNs for the generation of promising drug candidates in the future.

4.5 Technologies used

We listed the dependencies necessary to reproduce our experiments in Table 4.2. RDKit was used for the processing of chemistry data, such as molecule construction from graphs and sanitization. PyTorch was used for the implementation of the generative models, and PyTorch Geometric for the handling of graph data within the PyTorch framework. NetworkX was used for the underlying graph structures and environment on which sampled actions were executed. Scikit-learn was used for the implementation of the pChEMBL value-based QSAR random-forest regression models, as described in the DrugEx V2 paper [9]. This work extends and modifies code from DrugEx [1] and recursionpharma's GFlowNet implemenation repositories [2]. All experiments were performed on Linux machines using Python 3.9.

Dependency	Version
Python	3.9
PyTorch	1.1.2 (CUDA 11.3)
PyTorch Geometric	2.1.0
SciPy	1.9.2
NumPy	1.23.4
Pandas	1.5.0
RDKit	2022.03.5
NetworkX	2.8.7
scikit-learn	1.1.2
SELFIES	2.1.0
AiZynthFinder	3.6.0

TABLE 4.2: The dependencies necessary for our experiments.

[1] https://github.com/XuhanLiu/DrugEx
[2] https://github.com/recursionpharma/gflownet

Chapter 5

Results

The code used to run these experiments can be found on GitHub[1]. For our experiments, we first compared the performance of the GFN using two different MOO algorithms (all conditioned on preference):

- MOO-PC TB
- MOO-PC SubTB (λ)

Thereafter, we compared the performance of the autoregressive RNN-based SELFIES generator (DrugEx) to the fragment-based molecular graph building GFN using the PC version of TB at two different reward exponentiation values. The SELFIES generator will initially simply be used as a soft baseline to demonstrate the boost to diversity gained from leveraging the GFN's ability to be trained using algorithms to make the probability of sampling an object proportional to its associated reward.

The tasks that we will be investigating are as follows:

- The original DrugEx V2 Task (against SELFIES generator)
- Extending the DrugEx V2 Task with synthesizability (against SELFIES generator)
- Finding active (retro)synthesizable $A_{2A}AR$ agonists
- The performance of MOGFNs with 5 objectives

5.1 Trajectory Balance vs. Sub-Trajectory Balance

We trained our GFN using our MOO-PC SubTB algorithm(4.2) to assess if a different training objective results in a noticeably different distribution of molecules. Importantly, we empirically assessed two of the claims reported in the original SubTB paper[33]:

1. Improved convergence compared to other GFN training objectives at the time of writing, i.e., the target distribution is reached earlier.
2. Performs better in environments with sparse reward landscapes compared to the other training objectives at the time of writing, i.e. we can more reliably find the modes of the reward function.

Based on the results found by running the multi-target DrugEx V2 experiments under the same exact conditions as the experiment using TB, we did not observe a

[1] https://github.com/jcathalina/Rxitect/tree/msc-thesis

FIGURE 5.1: Score density of molecules at the start of training (orange) vs. the end of training (blue) for GFN using SubTB with $\lambda = 1$ as training objective.

FIGURE 5.2: Score density of molecules at the start of training (orange) vs. the end of training (blue) for GFN using TB as training objective.

noticeable change in the means of the distributions of the scores for any of the objectives at the beginning of training or after convergence (Figure 5.1, Figure 5.2). However, we observe that the distribution of scores early on during training are much peakier for the GFN trained with TB, particularly in the lower ranges of the scores. This can be explained by the faster convergence attributed to the SubTB algorithm, which we also observed during training, as shown in Figure 5.3. While both training objectives resulted in similar distributions after convergence, the peakiness persisted when using TB, which can be interpreted as it being a less stable algorithm.

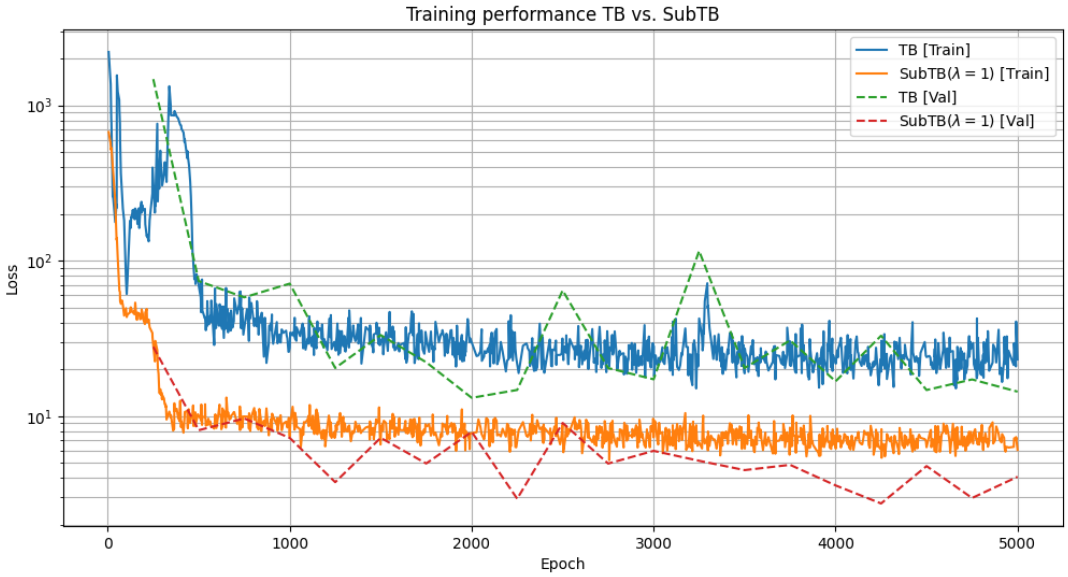

FIGURE 5.3: Learning curves depicting loss over time of SubTB (orange and red lines) vs TB (blue and green lines).

Despite this, given the same budget, we did not see a drastic difference in the predicted goodness of the molecules being generated. This observation led to us choose TB as the main GFN training objective throughout further experiments, as it was more efficient for rapid iteration during experimentation. It is entirely possible that SubTB could have discovered new modes given a higher budget. Based on the theory, we would also expect an improvement over TB if we increased the maximum trajectory length, as SubTB empirically performs better in these scenarios [33].

5.2 Recreating the DrugEx V2 task

As detailed in 3.1, DrugEx V2 concerned itself with the optimization of molecules for both a single-target and multi-target case. For both tasks, the biological targets to optimize for were the $A_{2A}AR$, A_1AR, and $hERG$. In the single target case, the objective was to only maximize $A_{2A}AR$ and minimize the other two, while in the multi-target case only $hERG$ should be minimized. We conduct the same experiment using the exact same regression models for the scoring of our samples to allow for comparison between the molecules generated by the GFNs and RNN generative models. We reiterate here that we used SELFIES instead of SMILES for the RNN models to guarantee validity of generated molecules.

5.2.1 Generated molecule diversity

To get insight into the sampling behaviour of our models, we computed the internal Tanimoto similarity and Solow-Polasky diversity measure for all batches. We had each model generate 100,000 unique candidates before converting them to 2048-bit Morgan fingerprints with a radius of 3. The internal Tanimoto similarity for both models in the base DrugEx V2 task setting can be seen in Figure 5.4. In this setting, DrugEx produced molecules with an overall lower mean and median internal Tanimoto similarity score. However, the GFN-produced candidates appear to contain outliers that are vastly different from the rest of the set, indicated by the long tail

approaching 0. When inspecting the Solow-Polasky diversity measure in the same setting, we again confirm the prior observation (Figure 5.5 – both the internal mean and median Solow-Polasky diversity of the DrugEx set lie higher than the GFN set, but the GFN set has a long tail approaching 1.

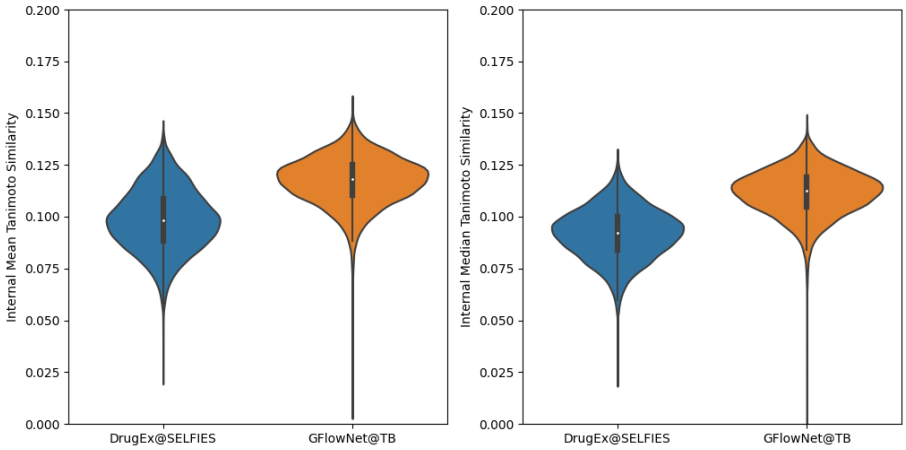

FIGURE 5.4: The internal mean (left) and median (right) Tanimoto similarity measures for DrugEx shown in blue and GFN shown in orange. Based on 100,000 unique samples for all cases.

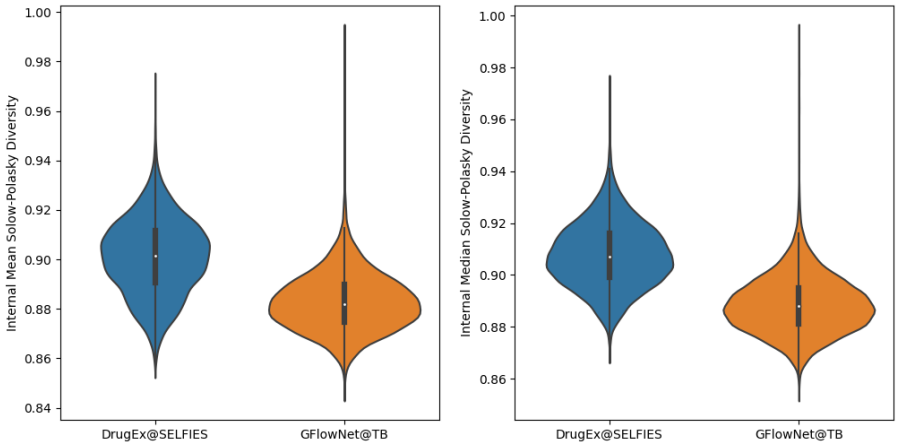

FIGURE 5.5: The internal mean (left) and median (right) Solow-Polasky diversity measures for DrugEx shown in blue and GFN shown in orange. Based on 100,000 unique samples for all cases.

When considering the extended case with RA Score as an additional objective, we notice a similar pattern. Here as well, the internal diversity of the DrugEx set is higher than that of the GFN set, with GFN displaying the same long tail indicating a few outliers that are very dissimilar to the rest of the set. Interestingly, in the extended case we do observe that the mean and median diversity of the molecule set generated by DrugEx is more concentrated around the middle, while the opposite was true in the base case, and vice versa for the GFN molecule set.

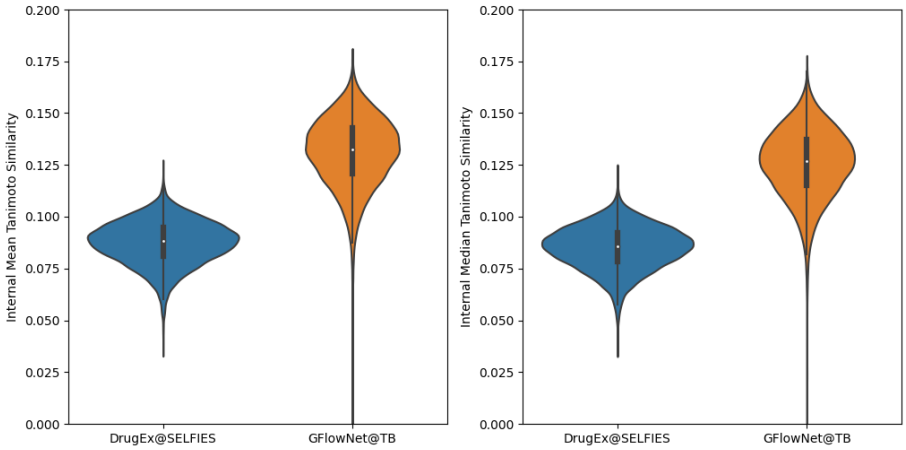

FIGURE 5.6: The internal mean (left) and median (right) Tanimoto similarity measures for DrugEx shown in blue and GFN shown in orange. Based on 100,000 unique samples for all cases.

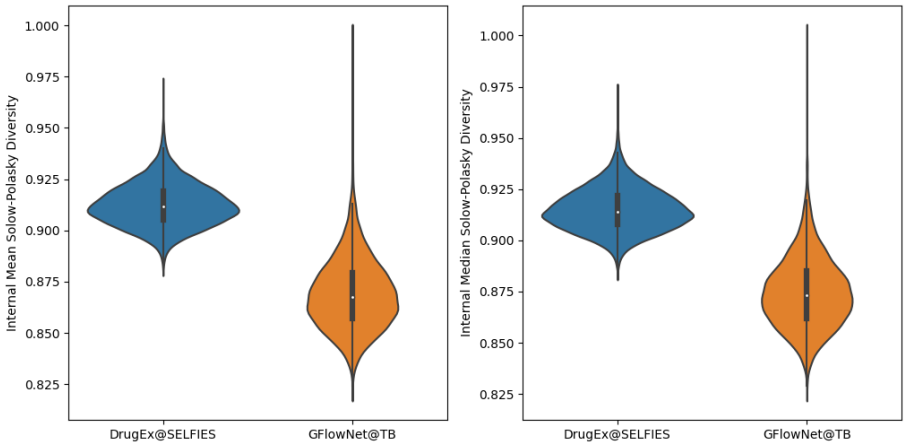

FIGURE 5.7: The internal mean (left) and median (right) Solow-Polasky diversity measures for DrugEx shown in blue and GFN shown in orange. Based on 100,000 unique samples for all cases.

In addition to the above-mentioned observations, it should be noted that while the DrugEx molecules had higher internal diversity, it took many more attempts for DrugEx to achieve 100,000 unique samples compared to the GFN, as shown in Table 5.1. The GFNs were able to almost always sample molecules that have not been visited before. Before adding the additional RA Score objective, DrugEx only sampled 72,534 unique molecules out of the 500,000 attempts (14.51%). After adding the additional objective, this increased to 46.22% of the 500,000 attempts producing unique molecules.

	Base Task	With RA Score
GFlowNet(TB)	$> 99\%$	$> 99\%$
DrugEx(SELFIES)	14.51%	46.22%

TABLE 5.1: The percentage of unique molecules generated for each case after sampling 500K molecules.

5.3 The effects of maximizing retrosynthetic accessibility

After generating 100,000 unique molecules for all 4 possible combinations of models and tasks, we visualized their score distribution in Figure 5.8 and Figure 5.11 for the base task and RA Score task, respectively.

5.3.1 Base task performance of both models

We observed that both models performed relatively similar for each objective that was maximized for (Figure 5.8). Note that the RA Score was also evaluated in this case, but not directly maximized. We immediately notice that for the $A_{2A}AR$, DrugEx was able to produce higher scoring molecules overall, but had much more variance when compared to the GFN molecules, which were mostly concentrated around a predicted normalized score of 0.7. We see the opposite of this for the A_1AR case, where the GFN molecules are mostly associated with scores higher than 0.8, whereas the distribution of the DrugEx sampled molecules is wider, with its mean lying closer to 0.7. For the $(1 - hERG)$ case, the GFN produced more high scoring molecules, yet a few of the DrugEx molecules were found with the highest scores. Finally, we found that without optimizing directly for the RA Score, DrugEx molecules are predicted to be more accessible than the GFN sampled molecules.

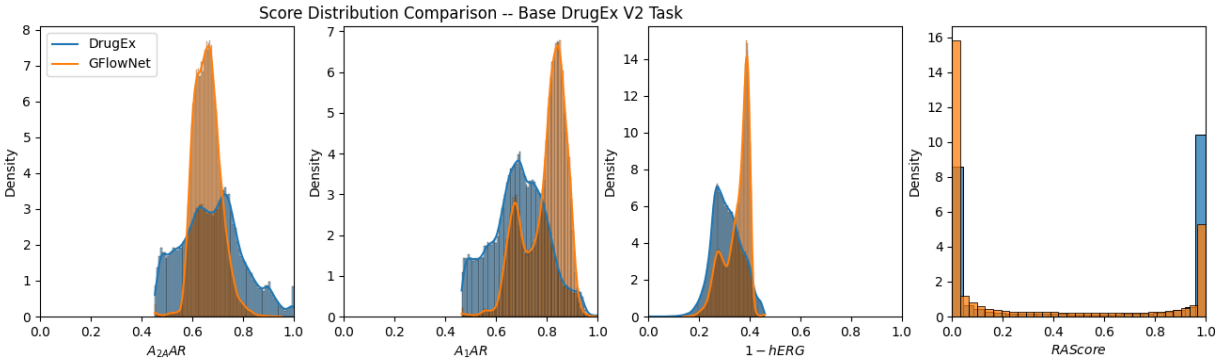

FIGURE 5.8: Score distribution comparison between molecules generated by DrugEx (blue) and GFN using TB (orange) for the base DrugEx V2 task (no direct RA Score maximization). Based on 100,000 unique samples for all cases.

When looking at Figure 5.9, we observe that the SA Score (not directly optimized) for the DrugEx molecules is distributed close to 1. Note that we transformed SA score so that it is on a scale of 0 to 1 where higher is better. This is the case despite the RA Score being confident about the retrosynthetic accessibility for approximately half of the samples.

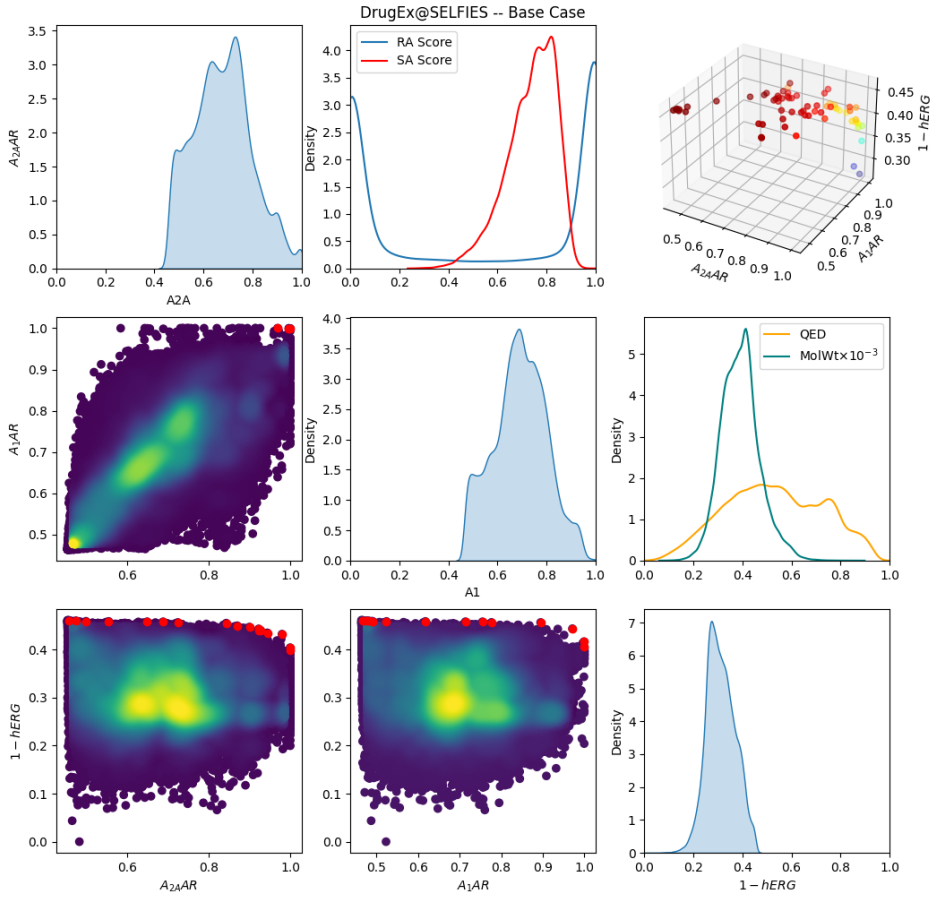

FIGURE 5.9: Score distribution pair plot from 100,000 samples generated by DrugEx in the context of the original DrugEx V2 task. The Pareto-optimal candidates per objective-pair are highlighted in red (bottom-diagonal). The diagonal contains the score distributions for the main objectives. The top-diagonal contains indirect objectives that are commonly associated with drug-likeness (QED, molecular weight), and also the RA Score and SA Score as accessibility heuristics. The top-right panel visualizes the 3D Pareto-frontier for the main objectives.

When comparing these results to Figure 5.10, we noticed a higher score for the molecular weight (lighter molecules) but a lower Quantitative Estimation of Drug-likeness (QED) score. A similar observation to the DrugEx results for the SA Score was made here – the SA score lies fairly close to 1, despite the RA Score distribution indicating that almost all sampled molecules are not retrosynthetically accessible. We also observed that the GFN more produced more Pareto optimal molecules overall, and displayed a different correlation pattern between all the possible objective pairs, as seen in the bottom-diagonal. Interestingly, the $A_{2A}AR$ and A_1AR objectives are optimized simultaneously in both models denoted by the positive trend in the pairplots. However, for DrugEx, the $(1 - hERG)$ objective does not show any particular trend and for GFN it even appears to perform worse for $(1 - hERG)$ if either of the other objectives are maximized.

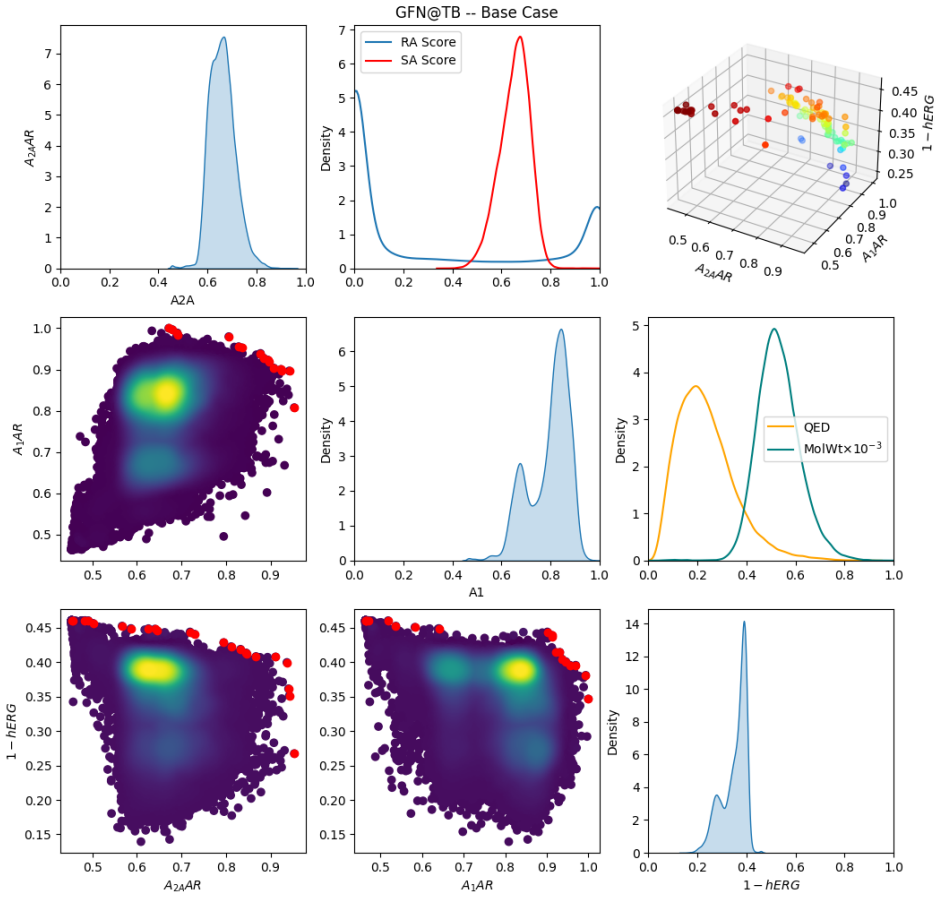

FIGURE 5.10: Score distribution pair plot from 100,000 samples generated by GFN using TB as the objective function in the context of the original DrugEx V2 task. The Pareto-optimal candidates per objective-pair are highlighted in red (bottom-diagonal). The diagonal contains the score distributions for the main objectives. The top-diagonal contains indirect objectives that are commonly associated with drug-likeness (QED, molecular weight), and also the RA Score and SA Score as accessibility heuristics. The top-right panel visualizes the 3D Pareto-frontier for the main objectives.

5.3.2 RA Score task performance of both models

After adding RA Score as an additional fourth objective, we observed a noticeable shift in the comparative performances of the models (Figure 5.11). Most notably, in the case of DrugEx, the $A_{2A}AR$ and A_1AR score distribution shifted to the left, i.e., the overall scores decreased. For the same objectives, the GFN was able to maintain equal performance. For the $(1 - hERG)$ objective, the opposite was true. The most striking observation was that the GFN was able to maximize the RA Score for almost all the molecules it sampled, whereas in the case of DrugEx, its RA Score remained similar to before optimization.

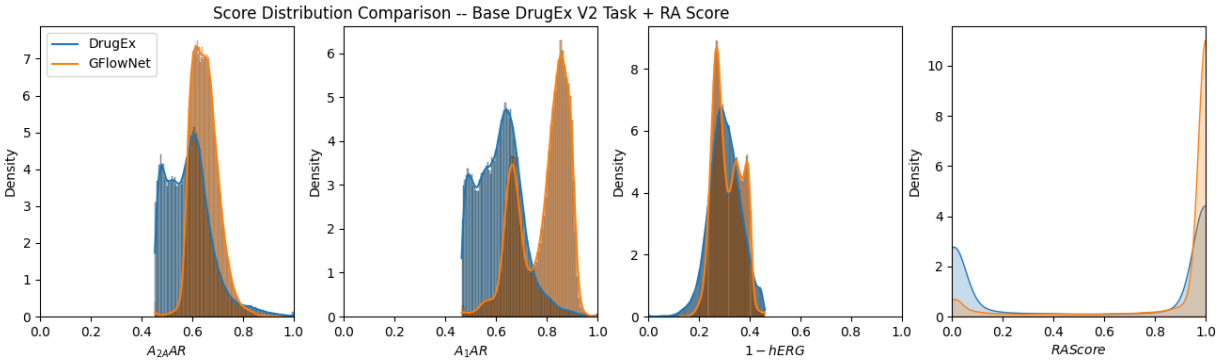

FIGURE 5.11: Score distribution comparison between molecules generated by DrugEx (blue) and GFN using TB (orange) for the extended DrugEx V2 task, including RA Score maximization. Based on 100,000 unique samples for all cases.

When looking at the pairplot for DrugEx in this setting (Figure 5.12), we observe a slight increase in RA Score, while the SA Score remains the same. We also see a slight improvement in the QED score. Despite the overall decreases in objective scores, the positive trend between $A_{2A}AR$ and A_1AR is still visible, albeit more spread out than before. Interestingly, we also noted weak visual evidence of a positive trend between those objectives and $(1 - hERG)$ as well here.

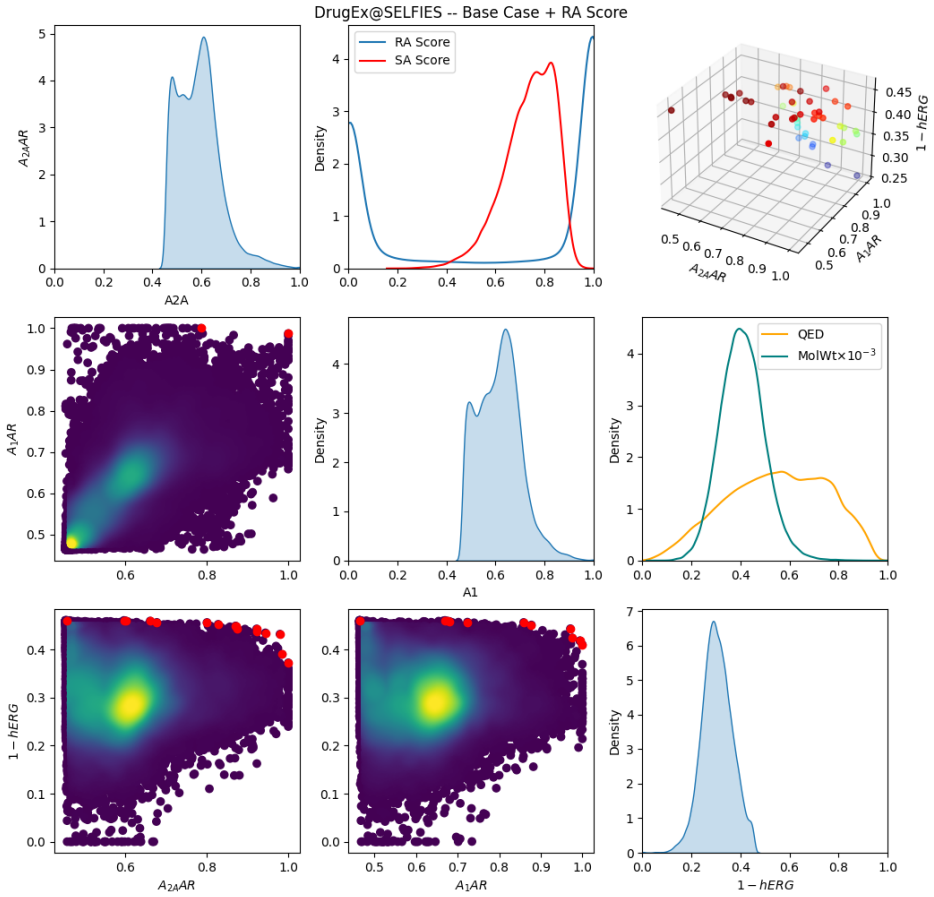

FIGURE 5.12: Score distribution pair plot from 100,000 samples generated by DrugEx in the context of the DrugEx V2 task extended with RA Score. The Pareto-optimal candidates per objective-pair are highlighted in red (bottom-diagonal). The diagonal contains the score distributions for the main objectives. The top-diagonal contains indirect objectives that are commonly associated with drug-likeness (QED, molecular weight), and also the RA Score and SA Score as accessibility heuristics. The top-right panel visualizes the 3D Pareto-frontier for the main objectives.

We subsequently compared the DrugEx results to the GFN results in Figure 5.13. As mentioned above, the RA Score has been maximized for almost all the molecules produced by the GFN – however, this did not have any effect on the SA Score. The trends between the objectives remained similar to those seen in the base task setting, although it seems more samples were concentrated in the lower ranges of $(1 - hERG)$ compared to before. We also observed marginal improvements in the overall QED score. Finally, the markedly higher scores for A_1AR achieved by the GFN compared to DrugEx in this setting are reflected in the Pareto fronts, and most samples are concentrated around high scores for A_1AR.

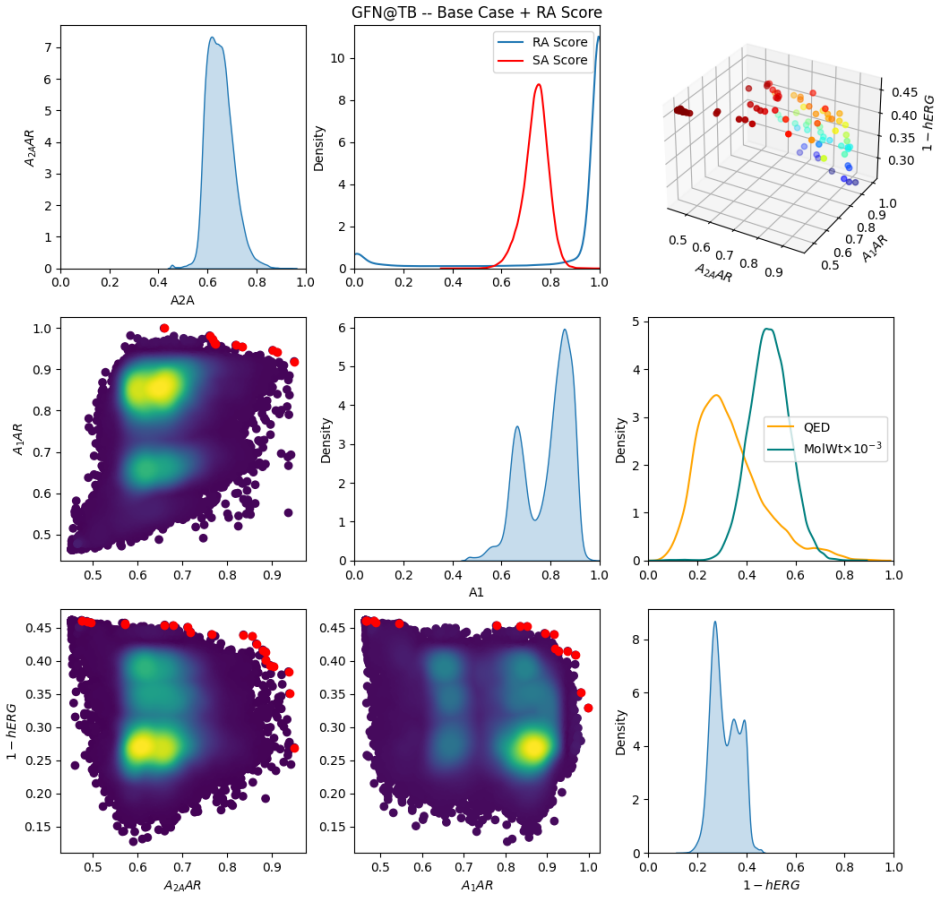

FIGURE 5.13: Score distribution pair plot from 100,000 samples generated by GFN using TB in the context of the DrugEx V2 task extended with RA Score. The Pareto-optimal candidates per objective-pair are highlighted in red (bottom-diagonal). The diagonal contains the score distributions for the main objectives. The top-diagonal contains indirect objectives that are commonly associated with drug-likeness (QED, molecular weight), and also the RA Score and SA Score as accessibility heuristics. The top-right panel visualizes the 3D Pareto-frontier for the main objectives.

Finally, we investigated the Pareto optimal candidates generated by the models in each setting by feeding them directly to AiZynthFinder for evaluation. We observed that in all cases, including RA Score as an objective resulted in better state scores (see Figure 5.14).

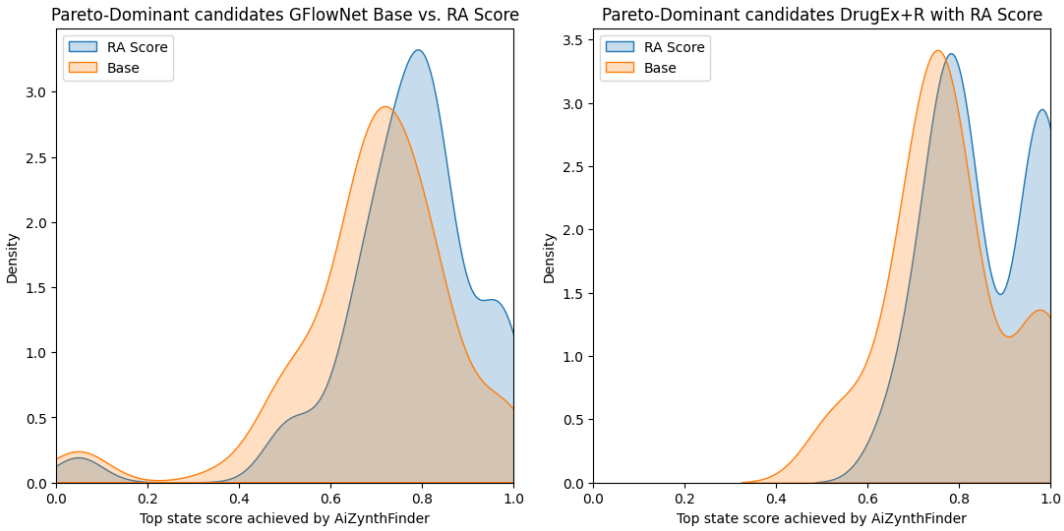

FIGURE 5.14: The top state score distribution given by AiZynthFinder for the Pareto-optimal candidates sampled by the GFN (left) and DrugEx (right), compared before and after introducing RA Score as an additional objective tot maximize.

5.4 Chemical space exploration

To visualize the chemical space being sampled by our models, we projected the generated molecules onto a two-dimensional plane against the known ligands used to train the QSAR scorers. For the base task, the result of this can be seen for both the GFN and DrugEx molecules in figures 5.15, 5.16 and 5.17. We observed that the molecules produced by the GFN consistently formed a cluster that was mostly disjoint from the space covered by the known ligands. DrugEx on the other hand, having been exposed to the known ligands during pre-training, produces molecules that have a lot of overlap with said ligands. The only instance wherein this is not the case for DrugEx is in the case of $(1 - hERG)$, where the objective is to *not* generate active ligands for the target.

A small sample taken from the GFN trained using objectives from the base task can be seen in Figure 5.19. We observed that it typically produced larger, more complex molecules with a high representation of nitrogen and sulphur atoms. When we include RA Score, the GFN produces noticeably simpler molecules, as seen in Figure 5.20. Nitrogen and sulphur atoms are still frequent, but we observed an increase in the occurrence of fluorine and chlorine atoms. When comparing these molecules to the ones generated by DrugEx, we observed that in the setting where RA Score was an additional objective, both models produced much more similar molecules compared to the base task. This was visualized in Figure 5.18 – the similarity of the molecules is higher in the second setting, indicated by the increase in overlap in the 2-dimensional projection from the base task to the task extended with RA Score. For reference, samples from the Pareto-optimal molecules generated by DrugEx for both settings can be found in Appendix A in Figure A.4 and Figure A.5.

Finally, we also observed that including RA Score results in increased coverage of the known-ligand chemical space in both cases, especially in the case of DrugEx. This observation is less pronounced for the GFN, but there we did observe that some molecules it sampled were similar to known active ligands, as can be seen for example for the $A_{2A}AR$ in Figure 5.21.

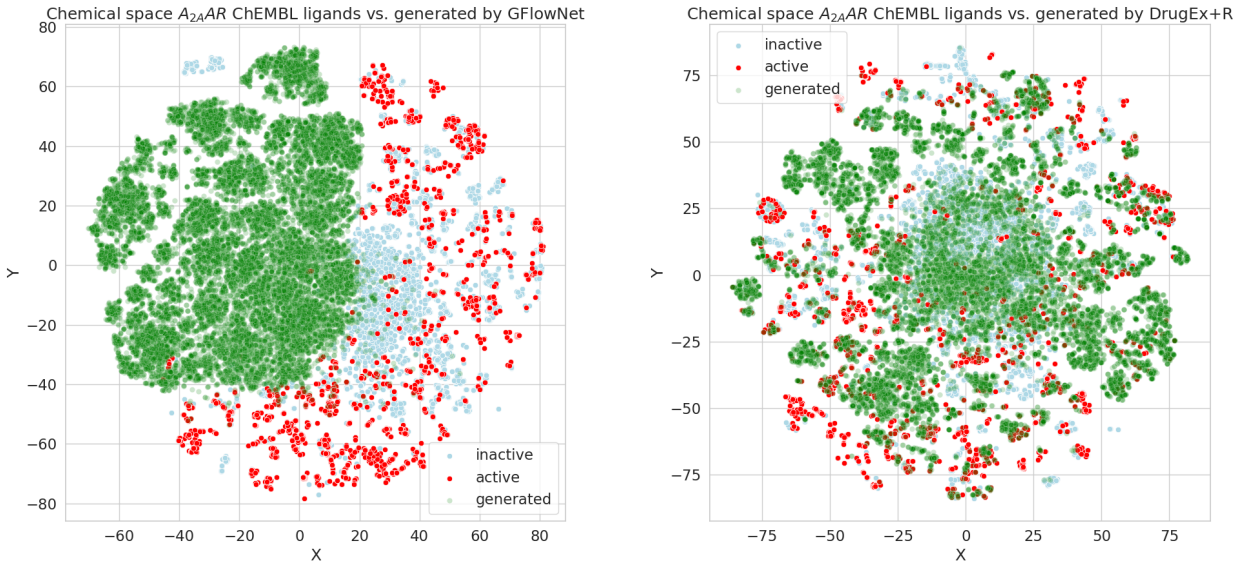

FIGURE 5.15: Visualization of the chemical space of GFN-sampled molecules (left) and DrugEx-sampled molecules (right) against known ChEMBL $A_{2A}AR$ ligands in the base task setting. The molecules generated by the models are shown in green. The known ligands are shown in blue (inactive) or red (active).

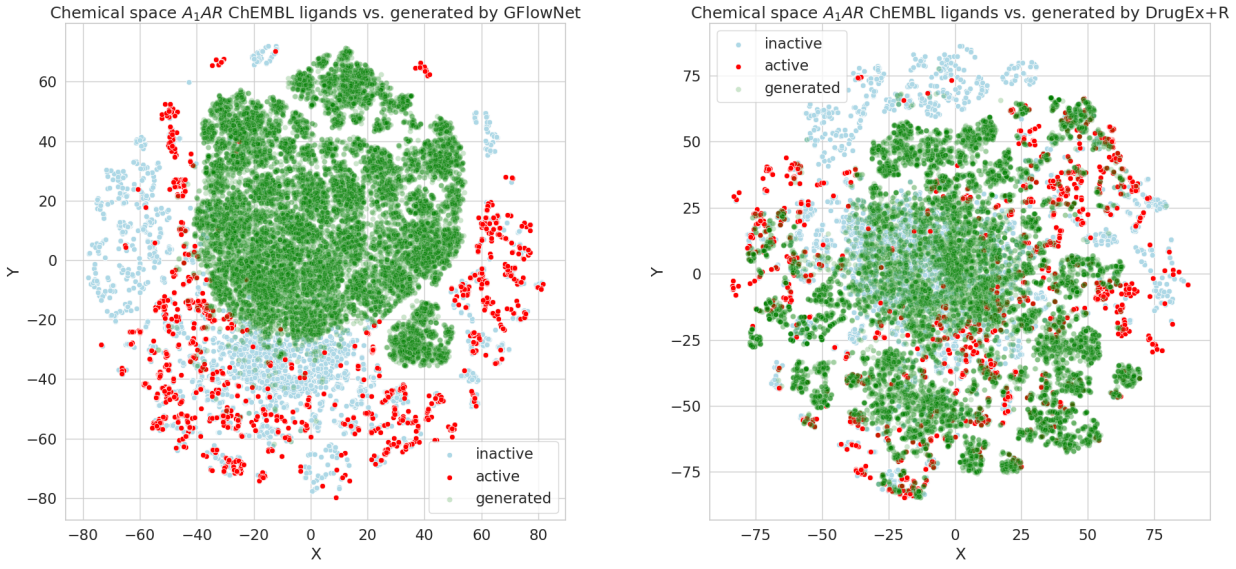

FIGURE 5.16: Visualization of the chemical space of GFN-sampled molecules (left) and DrugEx-sampled molecules (right) against known ChEMBL A_1AR ligands in the base task setting.

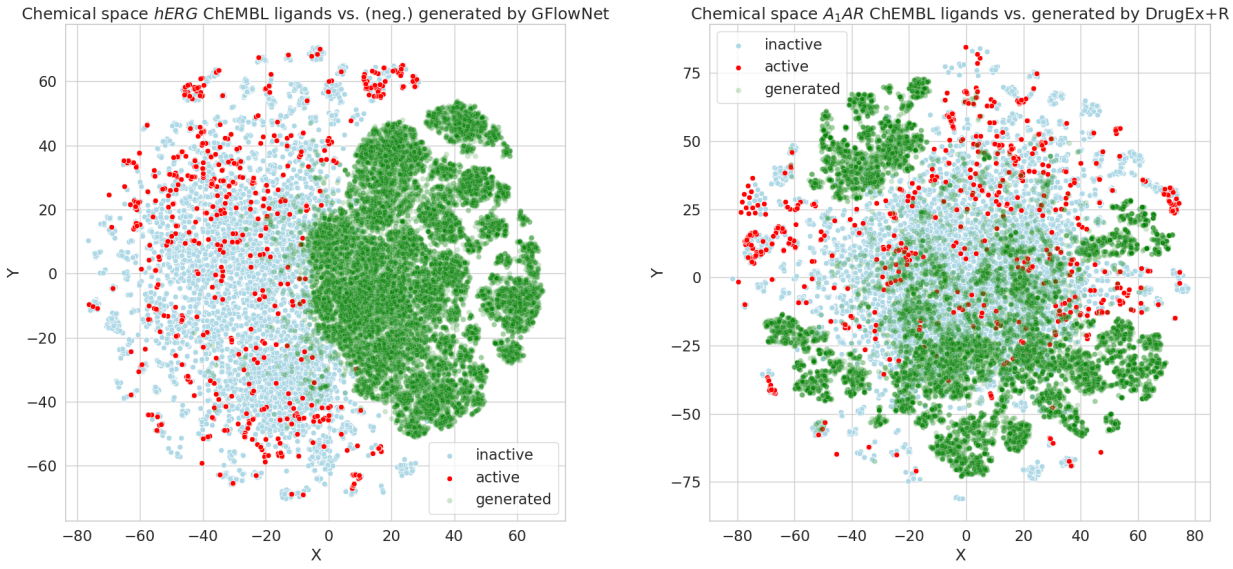

FIGURE 5.17: Visualization of the chemical space of GFN-sampled molecules (left) and DrugEx-sampled molecules (right) against known ChEMBL ($1 - hERG$) ligands in the base task setting.

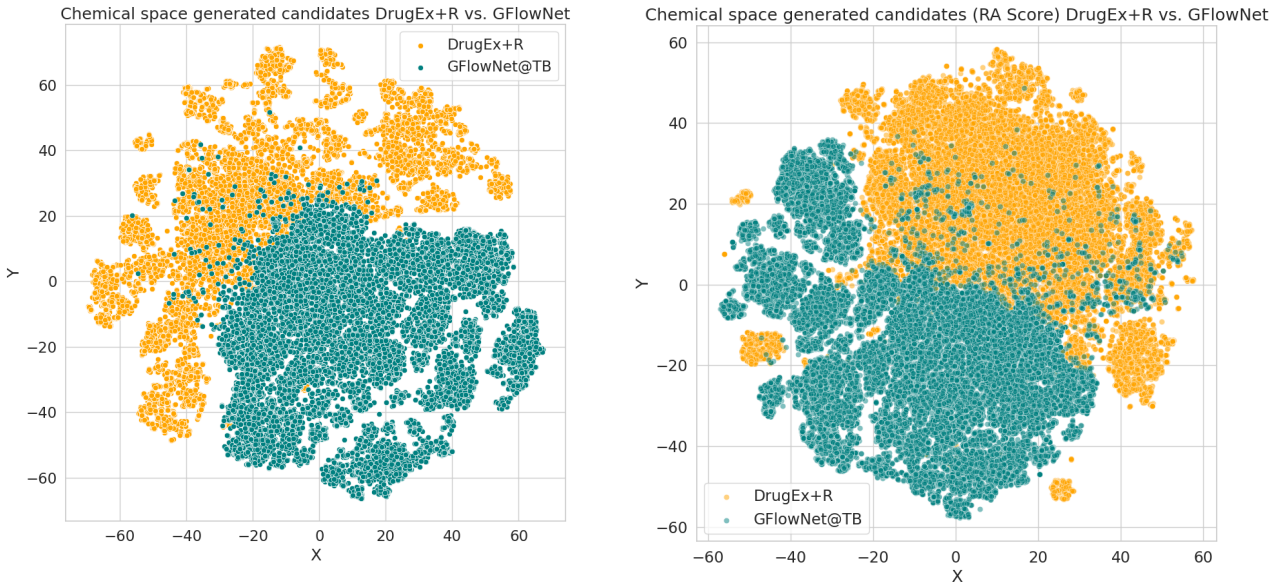

FIGURE 5.18: The chemical space visualized for molecules generated by DrugEx (yellow) and GFN (cyan) for the base DrugEx V2 task (left) and the task extended with RA Score (right).

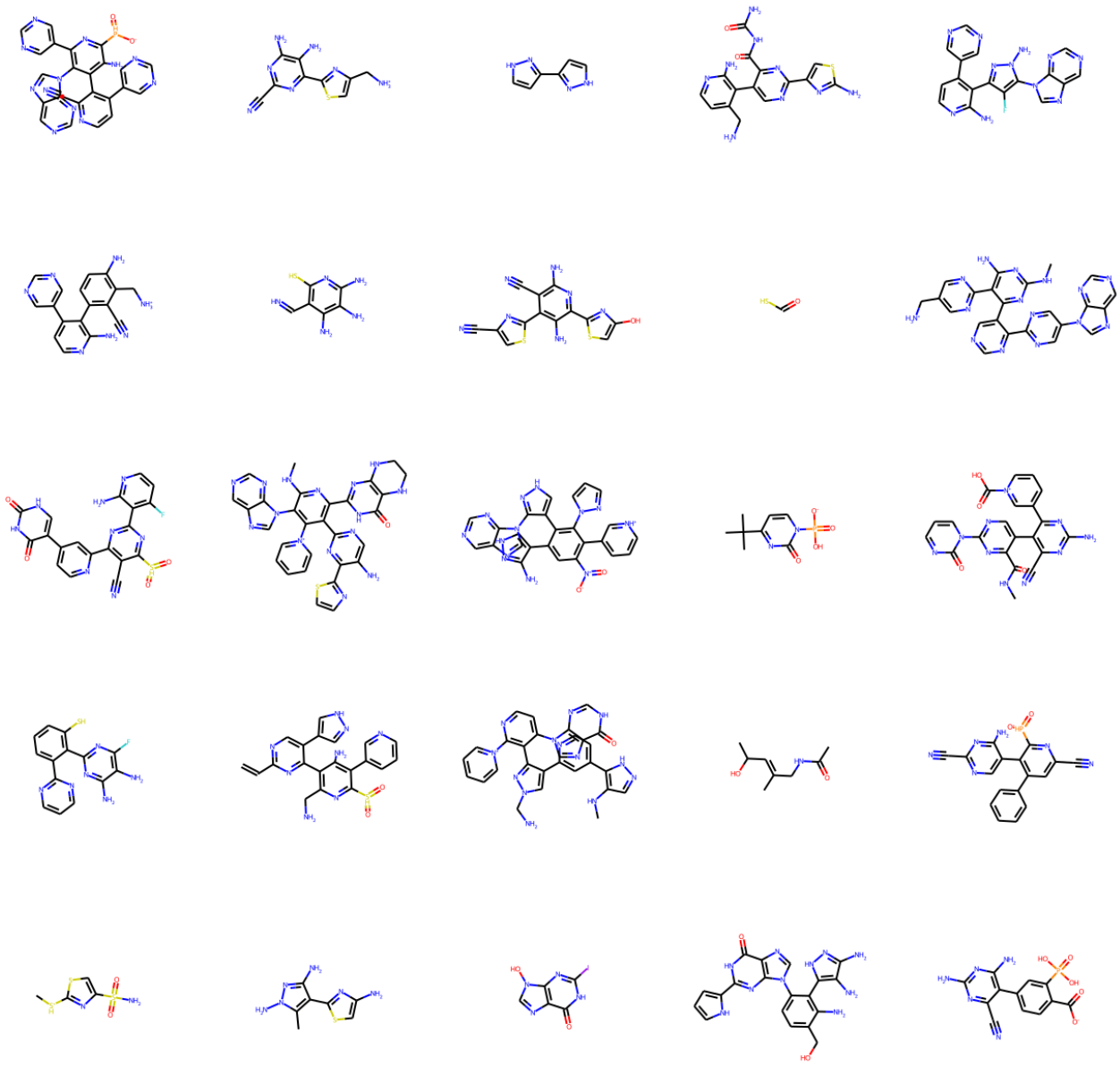

FIGURE 5.19: Sample from the Pareto-optimal molecules generated by GFN using TB with reward exponentiation factor β=96 for the base DrugEx V2 task.

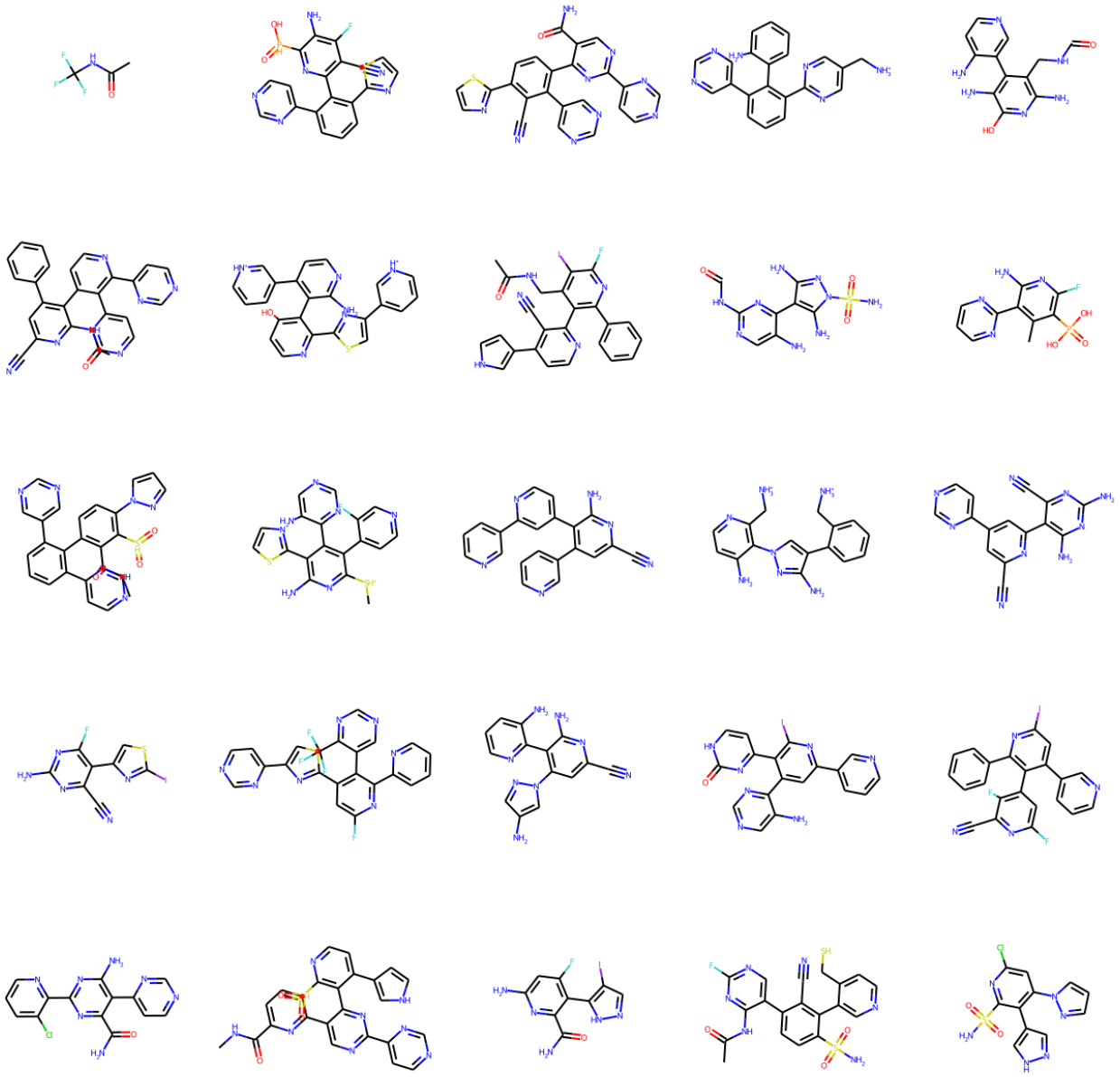

FIGURE 5.20: Sample from the Pareto-optimal molecules generated by GFN using TB with reward exponentiation factor β=96 for the extended (+RA Score) DrugEx V2 task.

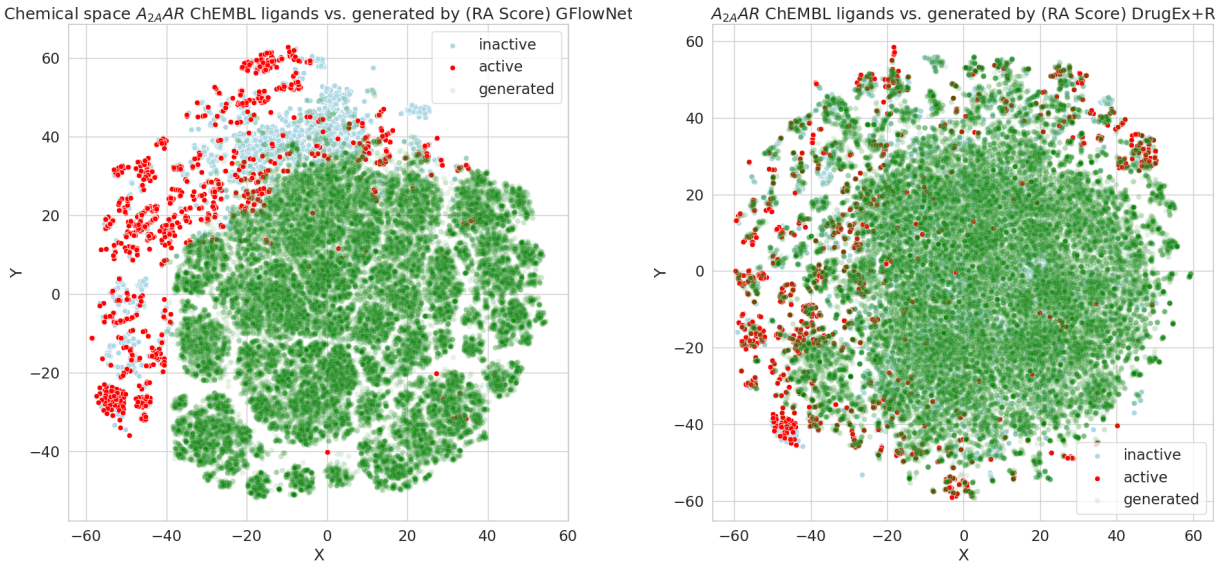

FIGURE 5.21: Visualization of the chemical space of GFN-sampled molecules (left) and DrugEx-sampled molecules (right) against known ChEMBL $A_{2A}AR$ ligands for the DrugEx V2 task extended with RA Score.

5.5 Exploring (retro)synthetically feasible Adenosine A2A Receptor agonists

To further explore the capabilities of the GFN, we explored what would happen if we focused on a single biological target along with accessibility by maximizing both RA Score and SA Score simultaneously. We observed again that the maximization of RA Score was successful for almost the entire batch of molecules produced by the GFN after convergence, as shown in Figure 5.22. The SA Score also shifts to the right after convergence from having the highest density around 0.7 to 0.8. The $A_{2A}AR$ score peaked at approximately the same score as it was for the previous experiments with multiple different biological targets. The bottom (or top) diagonals visualize the relationships between the objectives, where we see that none of the objectives compete with each other for maximization.

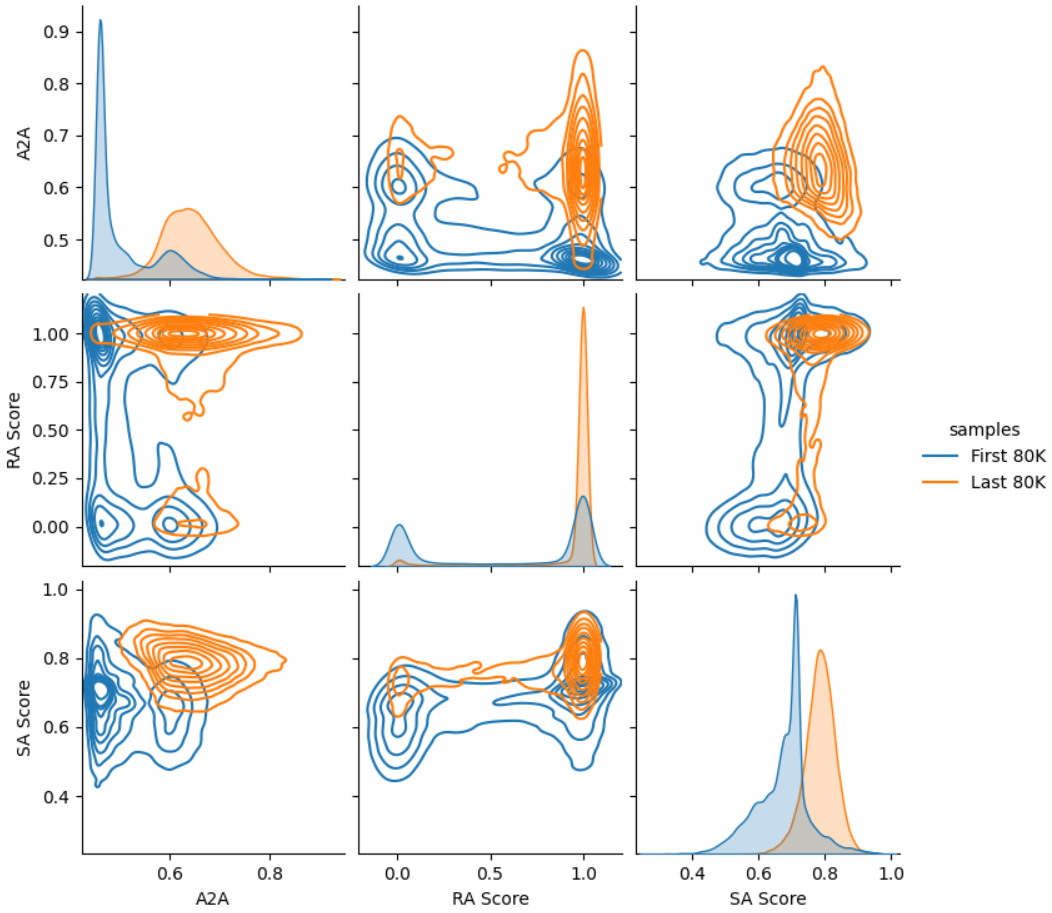

FIGURE 5.22: Comparisons between the properties of the first and last 80 thousand molecules sampled by the fragment-based GFN using TB for this task. The diagonals show the distribution shift between the early sampling (blue) and the sampling after the GFN has converged during training (orange). The off-diagonal plots show the relation between the different objectives.

We subsequently visualized the chemical space taken up by molecules sampled from this GFN against the known $A_{2A}AR$ ligands in Figure 5.23. We observed slightly increased overlap in comparison to the multiple target case with RA Score (see Figure 5.21), especially with active ligands. However, the same clustering can be seen here as well. Examples of molecules generated by this GFN can be seen in Figure 5.24

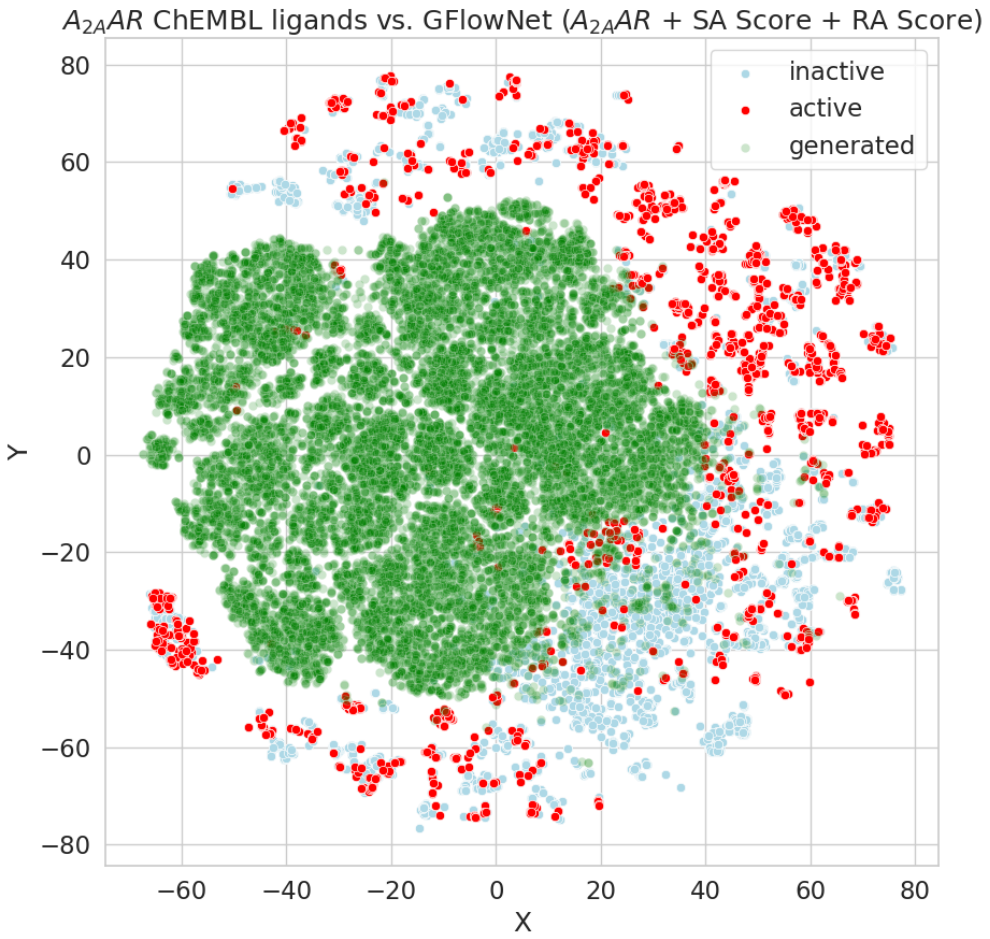

FIGURE 5.23: The chemical space covered by known $A_{2A}AR$ ligands compared to GFN generated molecules optimized solely for $A_{2A}AR$, SA Score and RA Score.

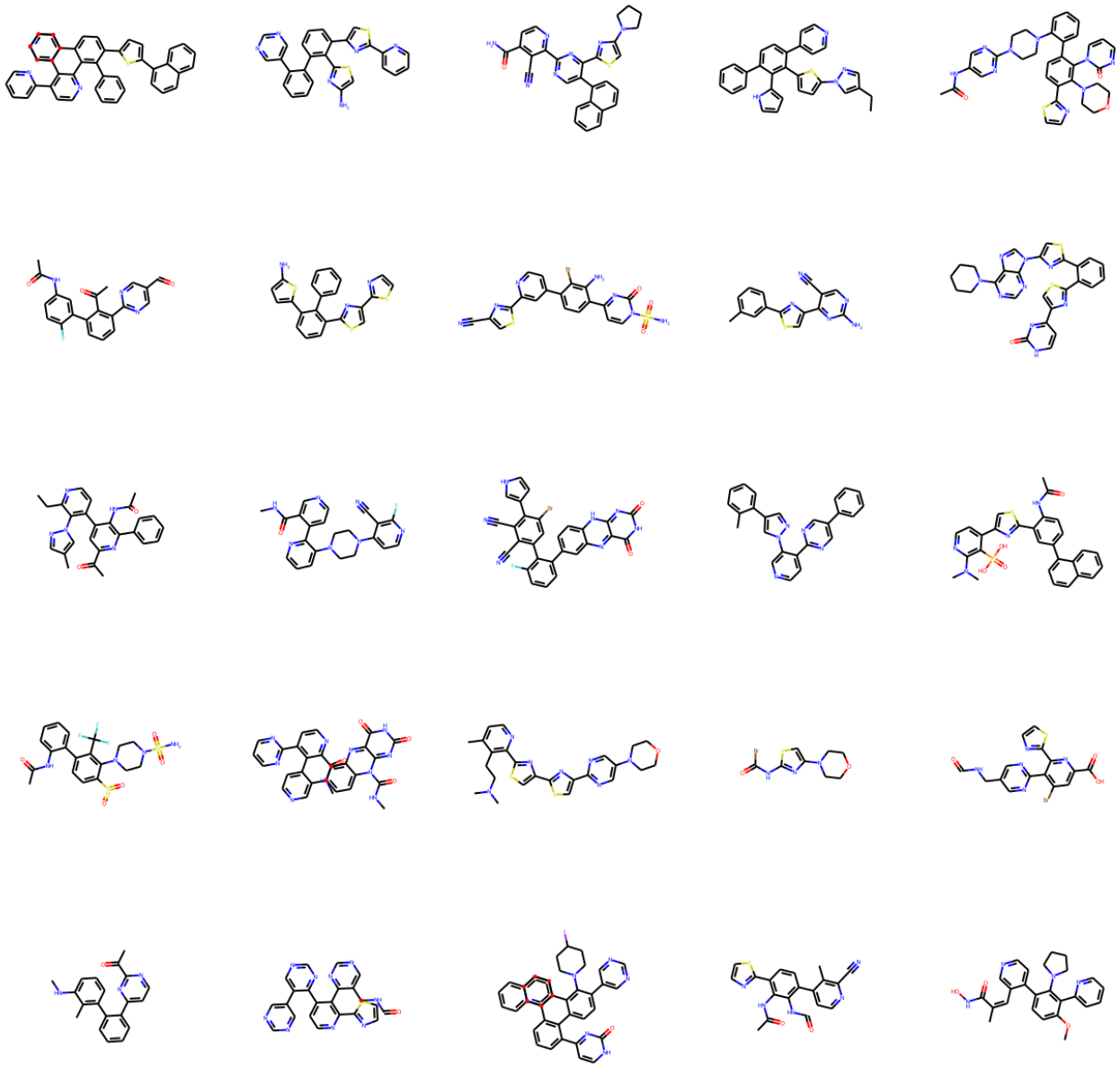

FIGURE 5.24: Sample from the molecules generated by GFN using TB with reward exponentiation factor β=96 for the synthesizable $A_{2A}AR$ task.

5.6 Training GFlowNets with 5 objectives

To further challenge the capabilities of GFNs, we experimented with the additional objective of maximizing the SA Score (taking into account that in the GFN setting, maximizing is interpreted as increasing the probability of sampling higher rewards proportionally, for some objective). This additional objectives was added to capture as much of the complexity of synthesizability as possible without overcomplicating the reward landscape, as we expected that maximizing the SA Score would, at the very least, not hinder RA Score maximization.

To assess if there was any improvement, we visualized the score distribution at the beginning and end of the training of the GFN by evaluating the first and last 80,000

molecules it generated as seen in Figure 5.25. The addition of SA Score as the fifth objective was not observed to have a significant impact on the maximization of the other objectives, as similar score distributions were achieved after convergence here as well. The SA Score itself was also improved after convergence, as the peak shifted right. It was also noteworthy that the first 80,000 molecules already showed score distributions close to the ones that would be achieved after convergence. A sample of the molecules produced by this GFN can be seen in Figure 5.26

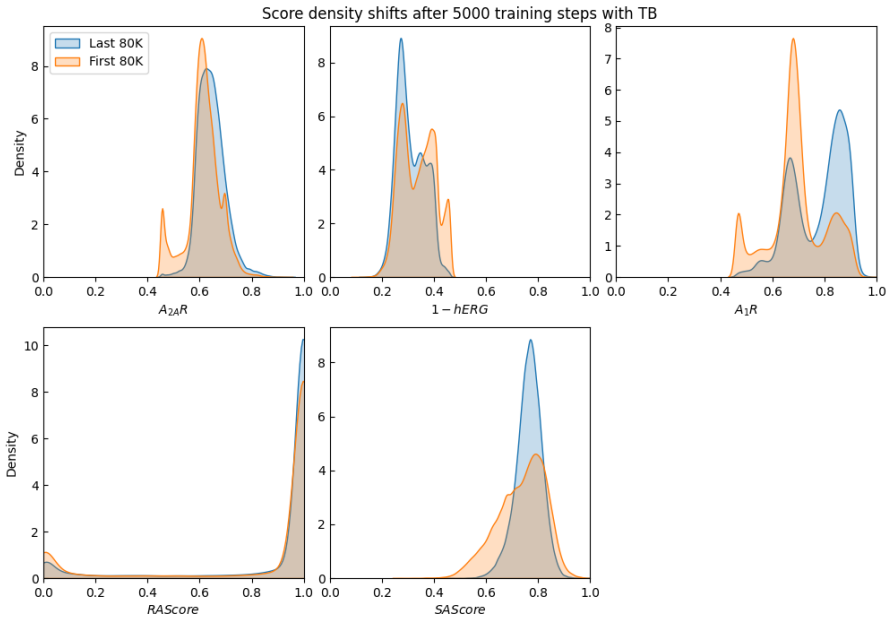

FIGURE 5.25: Score distribution shift of molecules at the start of training (orange) vs. the end of training (blue) for GFN using TB as training objective in the context of the 5-objective task.

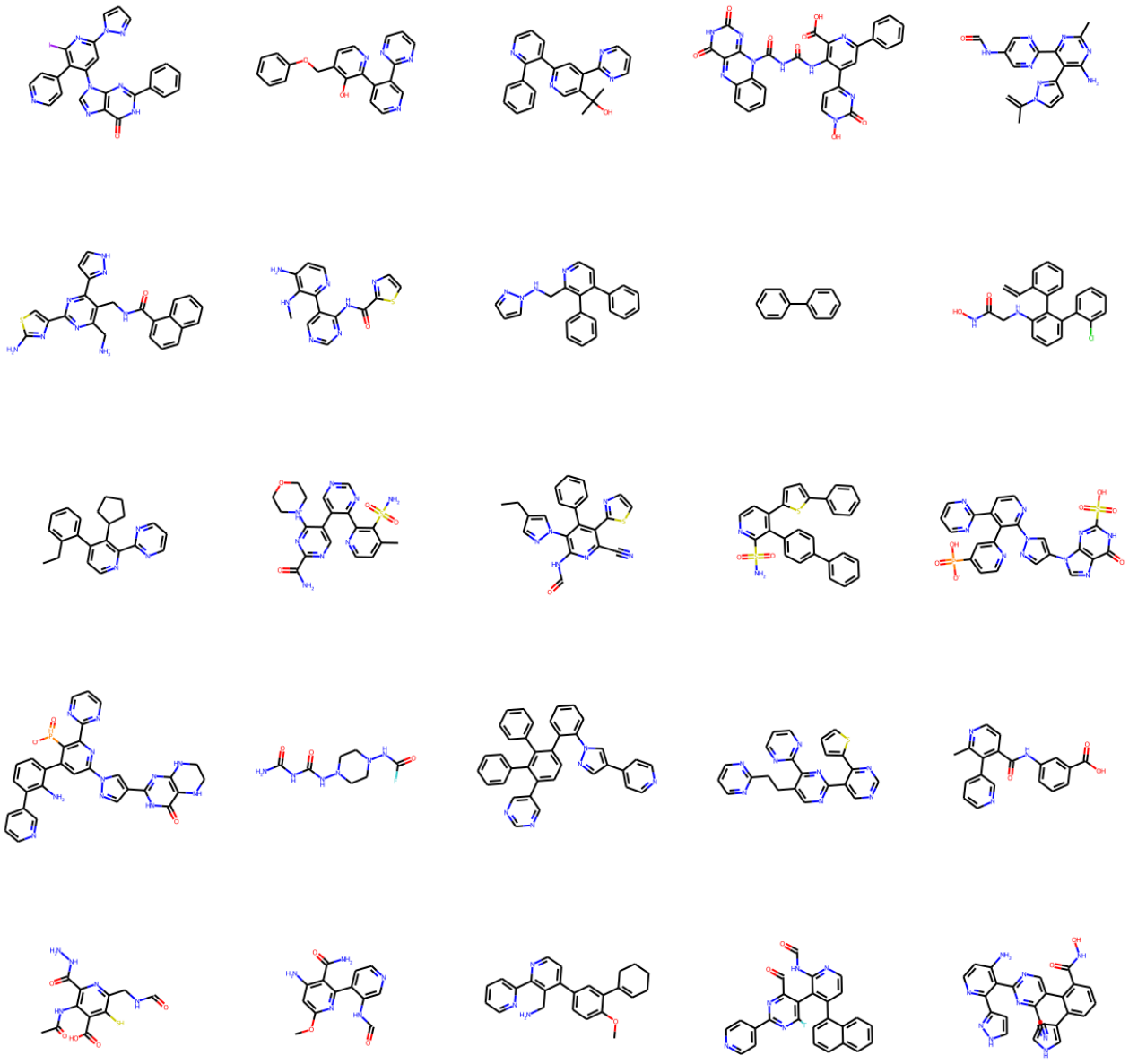

FIGURE 5.26: Sample from the molecules generated by GFN using TB with reward exponentiation factor β=96 for the 5-objective task.

5.7 Preference conditioning can be used to change sampling weights post-training

An important property of the PC-MOGFN that we wanted to test was the ability to change the focus (weight) of the target that our model samples for *after* it had already been trained. We had a GFN trained on the base task produce 10,000 molecules with uniformly varying preferences to assess if we would see the scores increase proportionally to their associated preference value, as displayed in Figure 5.27 We see that generally, as the preferences (y-axis) increased towards 1, the majority of scores shifted to the right accordingly. This effect was especially pronounced for the $(1 - hERG)$ objective.

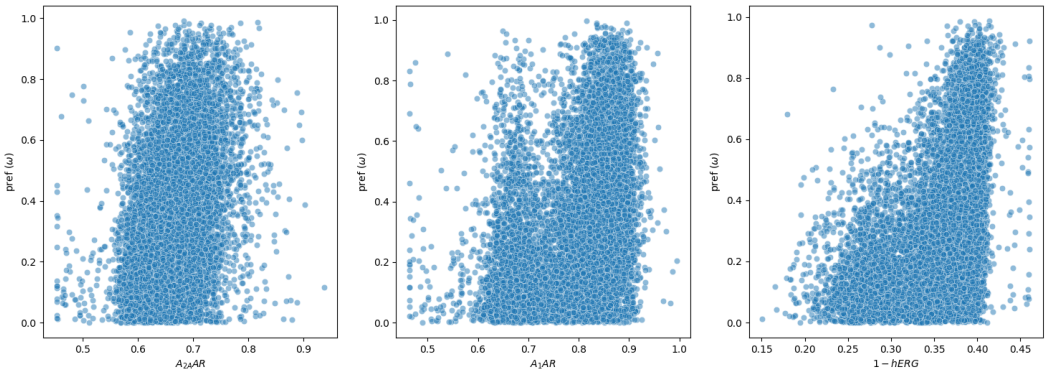

FIGURE 5.27: The relation between an increased preference (ω) value and the score distribution.

Chapter 6

Discussion

6.1 GFlowNets discover modes that are entirely outside known chemical space

Based on the results shown in section 5.4, we can observe that the molecules generated by the GFNs consistently form clusters that seldom overlap with the known ligands. This was expected to some degree, as the GFNs were trained purely on online data – that is, other than the reward signal coming from a model trained on the known ligands, there was no direct exposure to these known examples. Furthermore, the fragments that we used may not lead to structures similar to the known ligands as easily. It would be of interest to retry these experiments using fragments extracted from the same dataset as the target molecules to assess the impact of fragment choice. However, we do note that adding RA Score as an additional objective resulted in a slight increase in overlap with known ligands in the chemical space, as can be seen in, e.g., Figure 5.21. Interestingly, this shift is especially visible with the active ligands for $A_{2A}AR$, suggesting that the constraint of maintaining high retrosynthetic accessibility indeed results in more "realistic" molecules being suggested. This effect is further highlighted by comparing the samples from the two GFNs that were trained, both without (Figure 5.19) and with (Figure 5.20) maximizing RA Score as an objective. There is a noticeable decrease in size and complexity, when RA Score is considered, which is in line with expectations given that the score is based on known reaction templates.

Furthermore, we noticed that the GFN models were particularly adept to maximizing the A_1AR score. This is consistent across all tasks and settings, which could be a result of this objective being trivial. However, as DrugEx is not able to reproduce these scores, it is more likely that the GFN actually has an edge for producing predicted active A_1AR ligands. One could explain this through, e.g., the fragments that were used conveniently leading to similar structures to known ligands. This is not likely, as we saw in Figure 5.16 that the overlap in chemical space between the molecules produced by the GFN is negligible. It could also be that the QSAR model for the A_1AR can be exploited by the GFN, but that remains to be investigated and is currently outside the scope of this research.

6.2 GFlowNets are unlikely to sample duplicate molecules

While the GFNs were shown to sample from a chemical space that is dissimilar to that of the known ligands, we still found that overall, the set of molecules generated by Drugex were more internally diverse – that is to say, given the opportunity to produce the exact same amount of **unique** molecules, DrugEx has a slightly broader

sampling range than our current GFNs. However, when taking into account exactly how many molecules either of these models have to generate before reaching the same number of unique molecules, the difference is vastly in favour of GFNs, as shown in Table 5.1. We believe this is explainable by two factors:

1. The GFNs used in the experiment were trained entirely offline, while DrugEx was pre-trained on a dataset including known ligands. This results in the GFN naturally having to explore more, thereby finding more modes
2. The intrinsic modelling of diversity through GFN sampling proportionally to the reward as opposed to maximizing a single mode is especially pronounced here. DrugEx uses the latter, as it is RL-based.

Especially the second point is particularly important to explain this result. Sampling proportionally to the reward intuitively results in sampling different molecules each time, as it is not fixated on a particular mode, but rather samples around the node.

6.2.1 DrugEx samples more unique molecules, but has difficulty maximizing objectives, with RA Score as additional constraint

A related noteworthy observation here was that including RA Score as an additional objective drastically improved the rate of unique molecule sampling for DrugEx. This effect was not as pronounced for GFN, and they still retained a $> 99\%$ unique molecule sampling rate in this case as well. This could indicate that the additional objective forced the model to find more varied modes, as it had to account for an additional constraint. This could also be the result of increased randomness due to degradation of the modes (too many objectives). This is further backed up when we compare the score distributions of DrugEx with and without the additional RA Score objective, shown in Figure 5.11 and Figure 5.8, respectively. All objectives barring $(1 - hERG)$ shift to the left (decrease in score) when including RA Score as an objective, suggesting that it is more challenging for the underlying RL algorithm to handle.

6.3 RA Score significantly impacts molecules generated by GFlowNets

As mentioned in section 6.1, the candidates generated by GFNs trained with RA Score have more overlap with the chemical space of the known ligands. We also observed the molecules becoming less complex as the RA Score is maximized. Furthermore, it is of note that before including RA Score as an objective, DrugEx generally produces molecules that are predicted to be solvable, while GFN seems to have underperformed in that regard (see Figure 5.8). In contrast, we see that in the case where RA Score is directly optimized for, the GFN is able to maximize this effectively as shown in Figure 5.11. On the other hand, the RA Score distribution for DrugEx does not seem to have changed much compared to the base case, which is likely the result of not being able to readily find modes maximizing all four objectives as discussed above in subsection 6.2.1. While the score distribution in DrugEx generally shifted to the left in the extended setting, GFN was able to consistently find similar modes while maximizing RA Score, with the notable exception of the $(1 - hERG)$ objective. This suggests that the approach used by GFN is more resilient to complex multi-objective settings with many parameters. This is further bolstered by the results seen in the case of maximizing five objectives simultaneously, as shown in Figure 5.25, where the GFN did not have any of its scores shifted left despite the

addition of the additional objective. It can be argued, however, that SA Score was not a difficult objective to maximize. Especially in the case where we are already maximizing RA Score, which would intuitively be in agreement with a higher SA Score. However, we were not able to observe any explicit "synergistic" relationship between the RA Score and SA Score, as the molecules generated in settings where RA Score was not a direct objective would have high SA Score values even when the RA Score indicated that most of the molecules would not be retrosynthetically accessible.

6.4 Synthesizability constraints increase overlap in chemical space with known compounds

We observed in Figure 5.18 and Figure 5.21 that the molecules sampled by both GFN and DrugEx models increased in similarity to each other and the known ChEMBL ligands after including RA Score as an additional objective to optimize. Additionally, when we trained the GFN for the synthesizable $A_{2A}AR$ task, we also noted that there was even more overlap with the known ligand space than the first example (Figure 5.23), while retaining a similar score distribution for the target. This was interesting for us, as the GFN is never exposed to any of the known ligands, and is purely reliant on data it generates by exploring "blindly", unlike DrugEx, which is expected to cover much of said known ligand chemical space due to its exposure to it during pre-training – although we see an increase in overlap for DrugEx as well, further strengthening the argument that these constraints bring us closer to the known chemical space. This may also simply be explained by RA Score being trained on results of a tool that was itself trained using known reaction templates.

6.5 RA Score as an objective improves the synthetic accessibility of downstream GFlowNet-generated chemical libraries

Finally, we also observed that regardless of the model, using RA Score as an objective led to molecules with improved state scores when evaluated in AiZynthFinder (see Figure 5.14). This is an expected but important observation, as it supports the arguments for why we should use approaches such as RA Score as part of the virtual pre-screening pipeline in de-novo drug design moving forward – especially if we need to train models that are dependent on large amounts of data. As CASP tools become better and more prominent in filtering out unfeasible candidates in the drug discovery process, this approach becomes increasingly valuable in practice.

Chapter 7

Conclusion

Based on our results, we observed that a GFN using the TB training objective can be adapted to the multi-objective setting and produce predicted active ligands even without being exposed to any offline data. We also showed that it was capable of managing many objectives at the same time, and although we only demonstrated this with a maximum of 5 objectives, it was more resilient towards degradation of its scores compared to the alternative DrugEx model that was using an RL-based approach. While optimistic, being able to propose potential ligands that are entirely dissimilar to known ligands can also lead to the discovery of interesting novel compounds with entirely different binding mechanisms. Another observation in favour of GFNs was that unique molecules could consistently be sampled, which means that it would generally take much less time to generate a large library of relatively diverse candidates compared to other approaches.

Furthermore, we also showed that incorporating RA Score into our model training indeed led to higher state scores when evaluated directly using AiZynthFinder. This makes tools such as RA Score a valuable asset – given that running large amounts of molecules through an actual CASP tool is expensive, pre-filtering which molecules should be even be taken into consideration further streamlines the in-silico drug discovery pipeline.

Future Research

Our current GFN implementation has several straight-forward areas where it can be improved. Firstly, we reuse the same set of fragments introduced in the original GFN paper. This set was created from a subset of the ZINC database, and reduced to a small set of blocks. Despite 105 (72 unique) blocks being enough to sample a rather large number of unique molecules given enough graph building steps, it still introduces some level of rigidity in the space the model can explore. It would be interesting to expand the set of available blocks, or to experiment with more stable atom-by-atom generation approaches. Secondly, we trained our GFNs exclusively using offline data. We could observe that exposing models to some known examples of desirable compounds can aid in the discovery of modes in the DrugEx model, and other papers have shown the benefit of a balanced proportion of online training GFNs as well[43]. The challenge here for fragment-based models would of course be to efficiently generate trajectories for the online molecules using the available fragments. On the theoretical side, there are more training objectives to be explored to achieve flow balance.

Many GFN papers that have been published rely purely on empirical evidence, and more experimentation is necessary to solidify the framework. We've shown here

that SubTB(λ) also works for the Multi-Objective setting with the property of faster convergence – however the trade-off for this convergence speed is that once convergence is achieved, the sampling (during training) is much slower than with TB, which can be detrimental in the case where you want to continue sampling in this setting. Improving SubTB(λ) to compute sub-trajectory losses batch-wise would also be a solid contribution. Additionally, in settings where longer trajectories are desired (i.e., larger molecules or atom-wise generation), SubTB is likely to perform better. This is also worth investigating in-depth.

Finally, as mentioned before, RA Score partially inherits limitations of the CASP tool that it's trained on, and the fact that it learns a more flexible mapping can be considered a double-edged sword. Experimenting with other CASP tools, or ensuring the training examples come from a CASP tool that has more coverage (i.e., trained using larger – possibly proprietary– databases than USPTO) could result in more accurate mappings and, by extension, have an even stronger positive effect on the down-stream (retro)synthetic accessibility of our generated molecular library.

Bibliography

[1] Ana Sofia Pina, Abid Hussain, and Ana Cecília A. Roque. "An Historical Overview of Drug Discovery". In: *Methods in molecular biology (Clifton, N.J.)* 572 (2010), pp. 3–12. DOI: 10.1007/978-1-60761-244-5{_}1. URL: https://link-springer-com.ezproxy.leidenuniv.nl/protocol/10.1007/978-1-60761-244-5_1.

[2] Jürgen Drews. "Drug Discovery: A Historical Perspective". In: (). URL: http://science.sciencemag.org/.

[3] *High-throughput screening (HTS) | BMG LABTECH*. URL: https://www.bmglabtech.com/high-throughput-screening/.

[4] Christopher M. Dobson. "Chemical space and biology". In: *Nature* 432.7019 (Dec. 2004), pp. 824–828. DOI: 10.1038/NATURE03192.

[5] Gisbert Schneider and Uli Fechner. "Computer-based de novo design of drug-like molecules". In: *Nature Reviews Drug Discovery 2005 4:8* 4.8 (Aug. 2005), pp. 649–663. ISSN: 1474-1784. DOI: 10.1038/NRD1799. URL: https://www-nature-com.ezproxy.leidenuniv.nl/articles/nrd1799.

[6] Gisbert Schneider. *De novo molecular design*. Weinheim Germany: Wiley-VCH Verlag GmbH & Co. KGaA, 2014, p. 553. ISBN: 9783527334612.

[7] Gisbert Schneider and David E. Clark. "Automated De Novo Drug Design: Are We Nearly There Yet?" In: *Angewandte Chemie International Edition* 58.32 (Aug. 2019), pp. 10792–10803. ISSN: 1521-3773. DOI: 10.1002/ANIE.201814681. URL: https://onlinelibrary-wiley-com.ezproxy.leidenuniv.nl/doi/full/10.1002/anie.201814681https://onlinelibrary-wiley-com.ezproxy.leidenuniv.nl/doi/abs/10.1002/anie.201814681https://onlinelibrary-wiley-com.ezproxy.leidenuniv.nl/doi/10.1002/anie.201814681.

[8] Mariya Popova, Olexandr Isayev, and Alexander Tropsha. "Deep reinforcement learning for de novo drug design". In: *Science Advances* 4.7 (July 2018), eaap7885. ISSN: 2375-2548. DOI: 10.1126/SCIADV.AAP7885. URL: https://advances.sciencemag.org/content/4/7/eaap7885https://advances.sciencemag.org/content/4/7/eaap7885.abstract.

[9] Xuhan Liu et al. "DrugEx v2 : De Novo Design of Drug Molecule by Pareto-based Multi-Objective Reinforcement Learning in Polypharmacology". In: 1 ().

[10] David Silver et al. "Mastering chess and shogi by self-play with a general reinforcement learning algorithm". In: *arXiv preprint arXiv:1712.01815* (2017).

[11] John Jumper et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596.7873 (2021), pp. 583–589.

[12] Alhussein Fawzi et al. "Discovering faster matrix multiplication algorithms with reinforcement learning". In: *Nature* 610.7930 (2022), pp. 47–53.

[13] Marcus Olivecrona et al. "Molecular de-novo design through deep reinforcement learning". In: *Journal of cheminformatics* 9.1 (2017), pp. 1–14.

[14] Wenhao Gao and Connor W Coley. "The synthesizability of molecules proposed by generative models". In: *Journal of chemical information and modeling* 60.12 (2020), pp. 5714–5723.

[15] Peter Ertl and Ansgar Schuffenhauer. "Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions". In: *Journal of cheminformatics* 1.1 (2009), pp. 1–11.

[16] Connor W Coley et al. "SCScore: synthetic complexity learned from a reaction corpus". In: *Journal of chemical information and modeling* 58.2 (2018), pp. 252–261.

[17] Milan Voršilák et al. "SYBA: Bayesian estimation of synthetic accessibility of organic compounds". In: *Journal of cheminformatics* 12.1 (2020), pp. 1–13.

[18] Connor W Coley et al. "A robotic platform for flow synthesis of organic compounds informed by AI planning". In: *Science* 365.6453 (2019), eaax1566.

[19] Samuel Genheden et al. "AiZynthFinder: a fast, robust and flexible open-source software for retrosynthetic planning". In: *Journal of Cheminformatics* 12.1 (Dec. 2020), p. 70. ISSN: 17582946. DOI: 10.1186/s13321-020-00472-1. URL: https://jcheminf.biomedcentral.com/articles/10.1186/s13321-020-00472-1.

[20] Thomas Blaschke et al. "REINVENT 2.0: an AI tool for de novo drug design". In: *Journal of chemical information and modeling* 60.12 (2020), pp. 5918–5922.

[21] Xuhan Liu et al. "An exploration strategy improves the diversity of de novo ligands using deep reinforcement learning: a case for the adenosine A2A receptor". In: *Journal of cheminformatics* 11.1 (2019), pp. 1–16.

[22] David Weininger. "SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules". In: *Journal of chemical information and computer sciences* 28.1 (1988), pp. 31–36.

[23] Mario Krenn et al. "Self-referencing embedded strings (SELFIES): A 100% robust molecular string representation". In: *Machine Learning: Science and Technology* 1.4 (2020), p. 045024.

[24] Mario Krenn et al. "SELFIES and the future of molecular string representations". In: *arXiv preprint arXiv:2204.00056* (2022).

[25] Noel O'Boyle and Andrew Dalke. "DeepSMILES: an adaptation of SMILES for use in machine-learning of chemical structures". In: (2018).

[26] Laurianne David et al. "Molecular representations in AI-driven drug discovery: a review and practical guide". In: *Journal of Cheminformatics* 12.1 (2020), pp. 1–22.

[27] Marc G Bellemare, Will Dabney, and Rémi Munos. "A distributional perspective on reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 449–458.

[28] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. "Junction tree variational autoencoder for molecular graph generation". In: *International conference on machine learning*. PMLR. 2018, pp. 2323–2332.

[29] David Rogers and Mathew Hahn. "Extended-connectivity fingerprints". In: *Journal of chemical information and modeling* 50.5 (2010), pp. 742–754.

[30] Emmanuel Bengio et al. "Flow network based generative models for non-iterative diverse candidate generation". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 27381–27394.

[31] Yoshua Bengio et al. "Gflownet foundations". In: *arXiv preprint arXiv:2111.09266* (2021).

[32] Nikolay Malkin et al. "Trajectory Balance: Improved Credit Assignment in GFlowNets". In: *arXiv preprint arXiv:2201.13259* (2022).

[33] Kanika Madan et al. "Learning GFlowNets from partial episodes for improved convergence and stability". In: *arXiv preprint arXiv:2209.12782* (2022).

[34] Xuhan Liu et al. "An exploration strategy improves the diversity of de novo ligands using deep reinforcement learning: a case for the adenosine A2A receptor". In: *Journal of cheminformatics* 11.1 (2019), pp. 1–16.

[35] John J Irwin and Brian K Shoichet. "ZINC- a free database of commercially available compounds for virtual screening". In: *Journal of chemical information and modeling* 45.1 (2005), pp. 177–182.

[36] Anna Gaulton et al. "The ChEMBL database in 2017". In: *Nucleic acids research* 45.D1 (2017), pp. D945–D954.

[37] Andrew R Solow and Stephen Polasky. "Measuring biological diversity". In: *Environmental and Ecological Statistics* 1.2 (1994), pp. 95–103.

[38] Marwin H.S. Segler, Mike Preuss, and Mark P. Waller. *Planning chemical syntheses with deep neural networks and symbolic AI*. Mar. 2018. DOI: 10.1038/nature25978. URL: https://www-nature-com.ezproxy.leidenuniv.nl/articles/nature25978.

[39] Amol Thakkar et al. "Retrosynthetic accessibility score (RAscore)–rapid machine learned synthesizability classification from AI driven retrosynthetic planning". In: *Chemical science* 12.9 (2021), pp. 3339–3349.

[40] Moksh Jain et al. "Multi-Objective GFlowNets". In: *arXiv preprint arXiv:2210.12765* (2022).

[41] Jacob Buckman et al. "Thermometer encoding: One hot way to resist adversarial examples". In: *International Conference on Learning Representations*. 2018.

[42] Amol Thakkar et al. "Retrosynthetic accessibility score (RAscore)-rapid machine learned synthesizability classification from AI driven retrosynthetic planning". In: *Chemical Science* 12.9 (2021), pp. 3339–3349. ISSN: 20416539. DOI: 10.1039/d0sc05401a. URL: https://github.com/.

[43] Moksh Jain et al. "Biological sequence design with gflownets". In: *International Conference on Machine Learning*. PMLR. 2022, pp. 9786–9801.

Appendix A

Additional Results

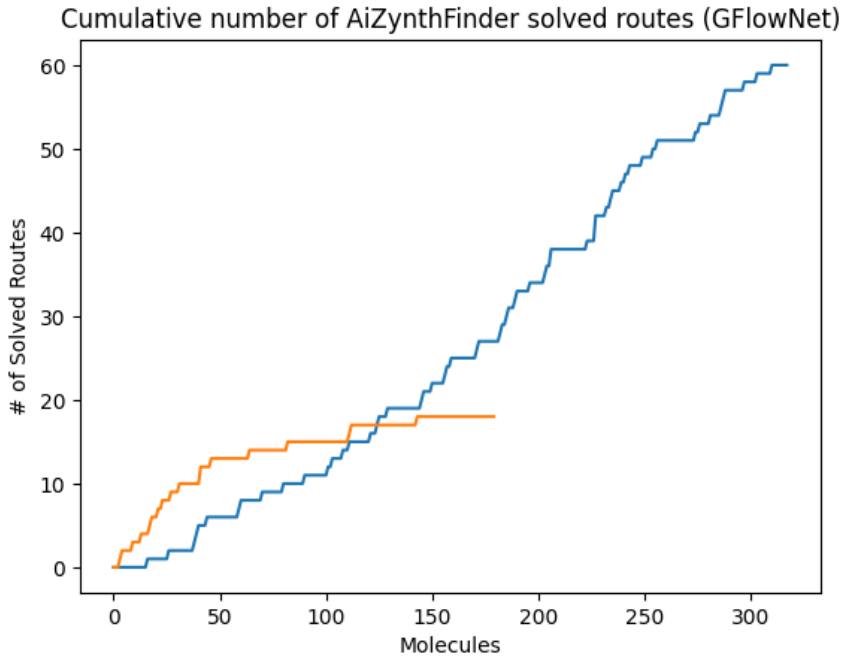

FIGURE A.1: The trend of number of AiZynthFinder solved routes for Pareto-optimal molecules generated by the GFlowNet trained with RA Score (blue) and only the base task (orange). The lines do not match in length, as the RA Score trained GFlowNet produced more Pareto-optimal candidates.

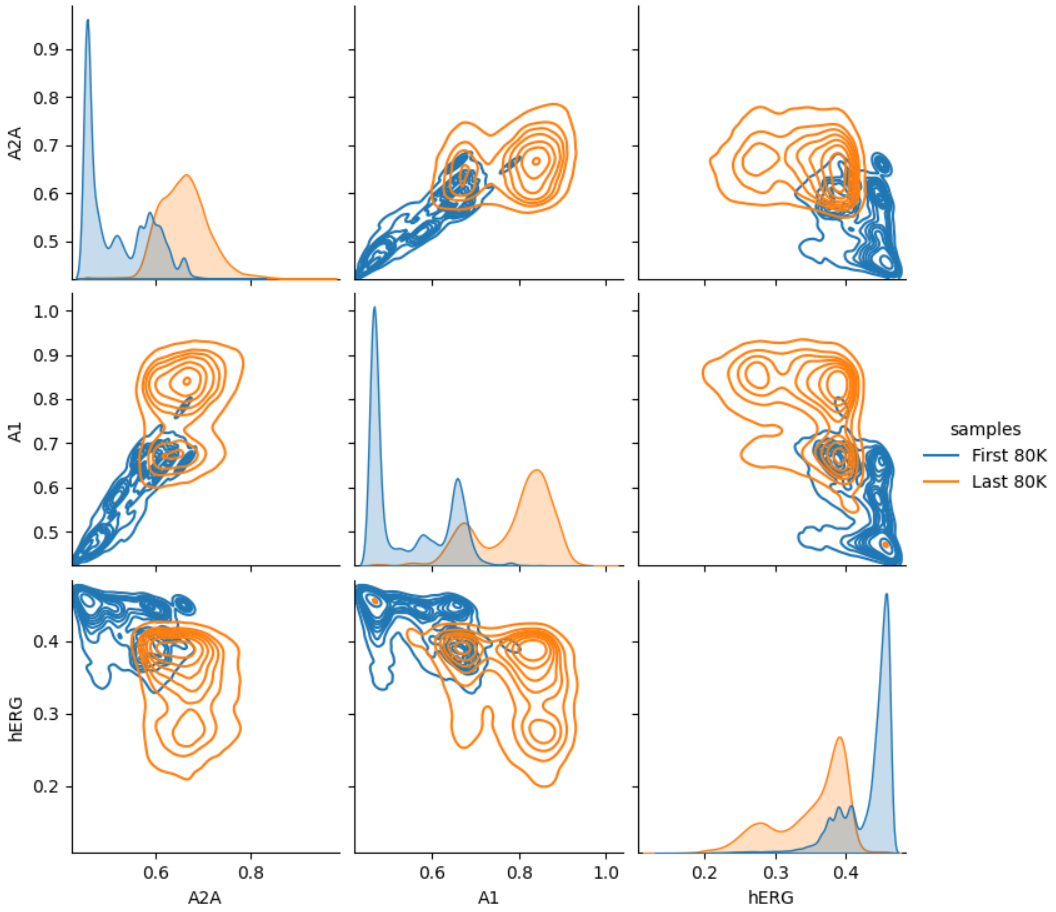

FIGURE A.2: The shift in score distribution and relationship between objectives, before (blue) and after convergence (orange) of the molecules generated by the GFlowNet using Trajectory Balance.

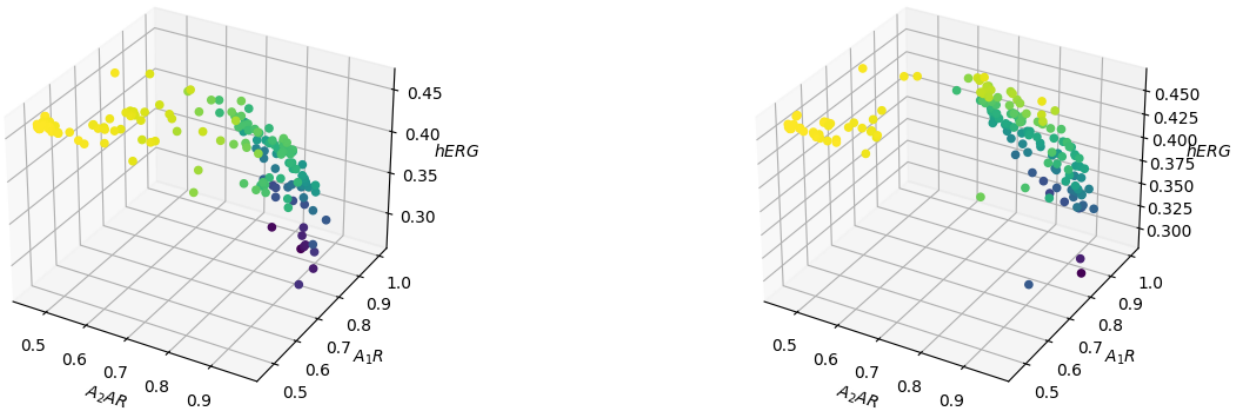

FIGURE A.3: The Pareto fronts found for the DrugEx V2 task of simultaneously optimizing pChEMBL values for $A_{2A}AR$, A_1AR and $1 - hERG$ using the fragment-based GFN with (left) the reward exponentiation factor β set to 32 and (right) β set to 96.

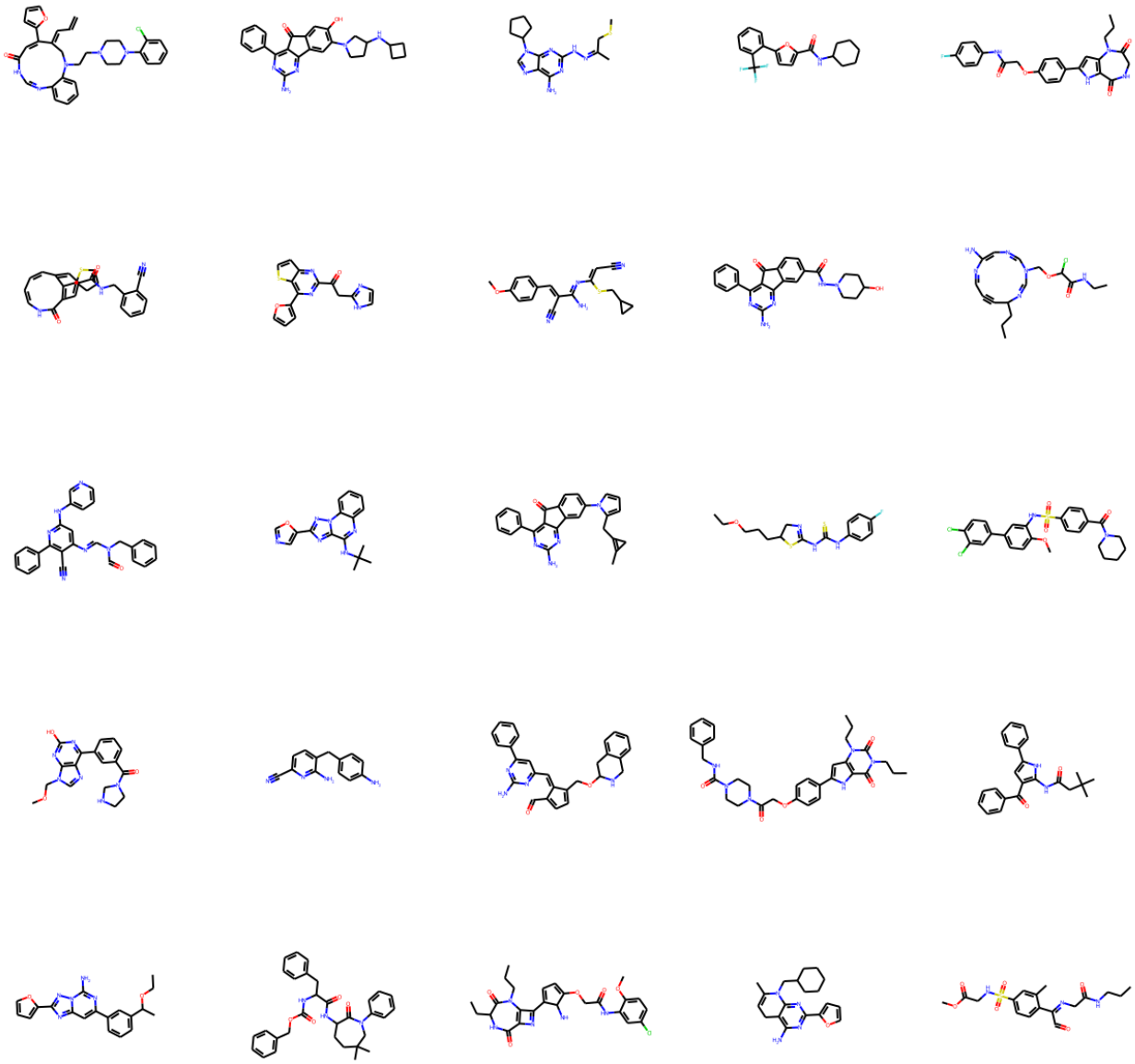

FIGURE A.4: Sample from the molecules produced by DrugEx(SELFIES) in the context of the base DrugEx V2 task.

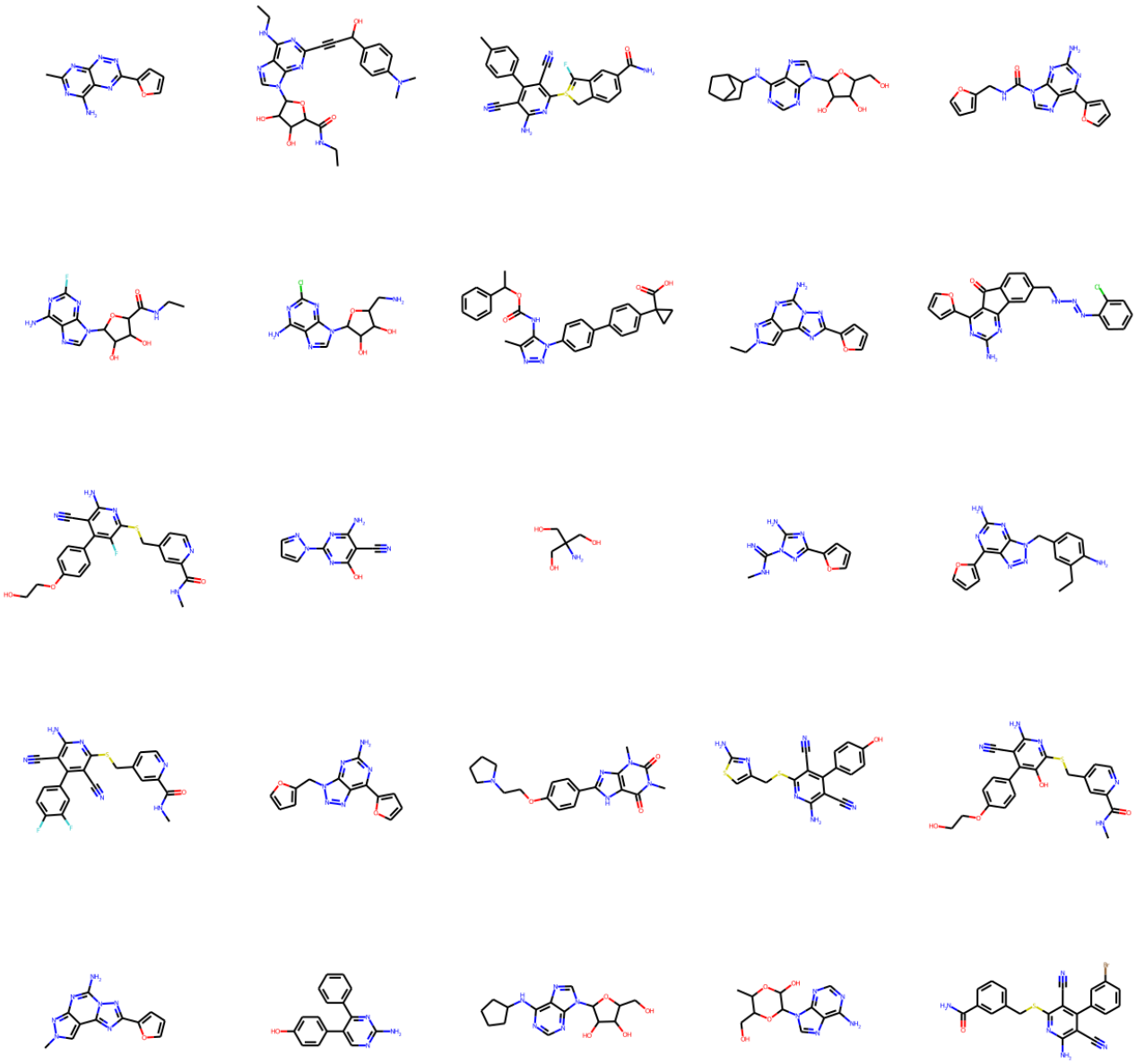

FIGURE A.5: Sample from the Pareto-optimal molecules produced by DrugEx(SELFIES) in the context of the DrugEx V2 task extended with RA Score.

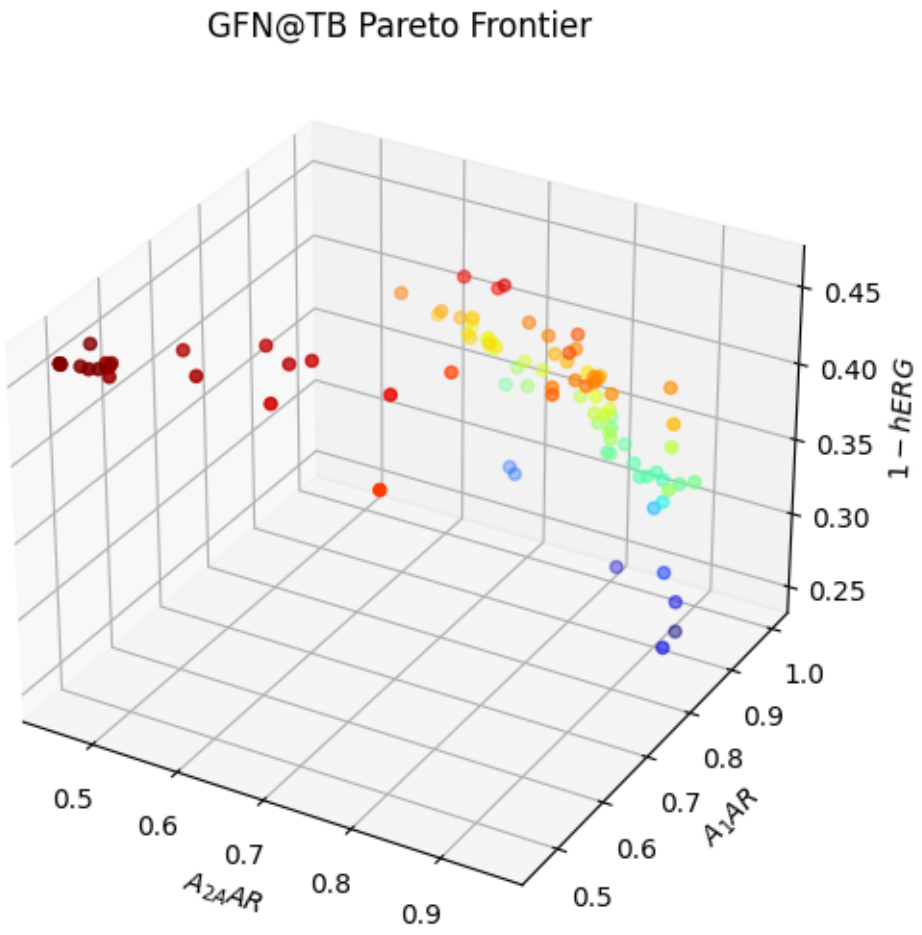
GFN@TB Pareto Frontier
0.45
0.40
0.35
0.30
0.25
1 − hERG
1.0
0.9
0.8
0.7
0.6
0.5
A1AR
0.5
0.6
0.7
0.8
0.9
A2AAR

FIGURE A.6

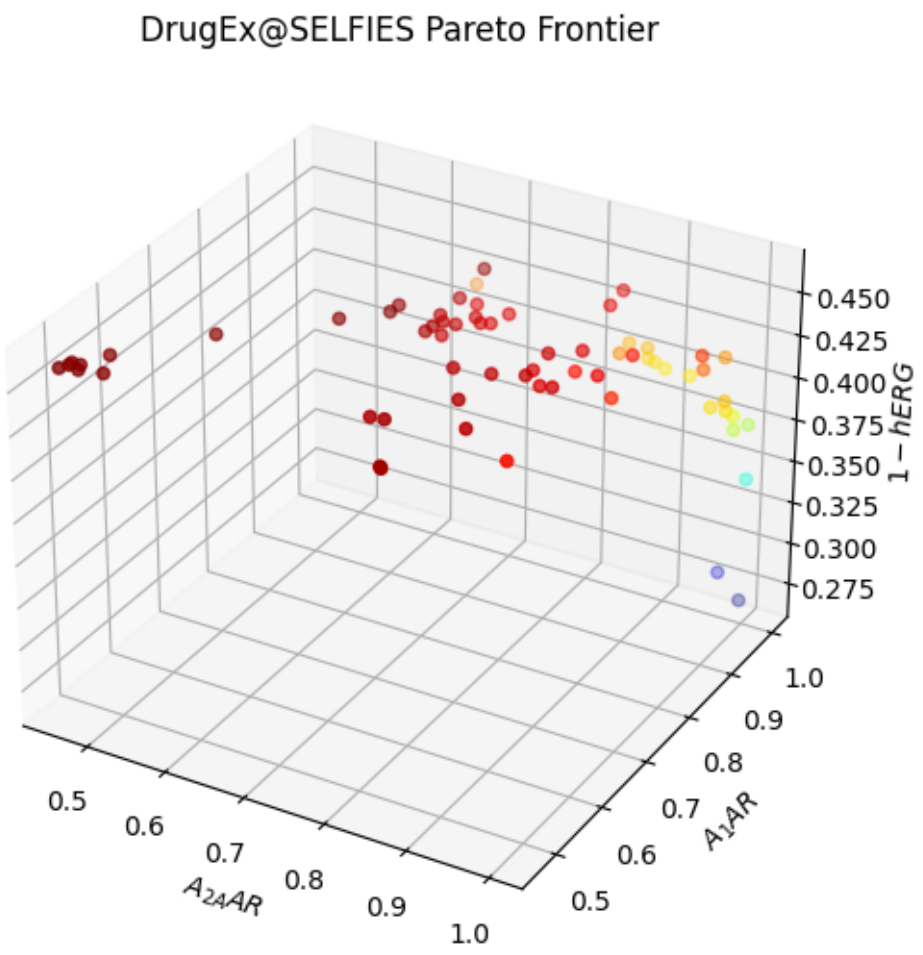
DrugEx@SELFIES Pareto Frontier
0.450
0.425
0.400
0.375
0.350
0.325
0.300
0.275
1 − hERG
1.0
0.9
0.8
0.7
0.6
0.5
A1AR
0.5
0.6
0.7
0.8
0.9
1.0
A2AAR

FIGURE A.7

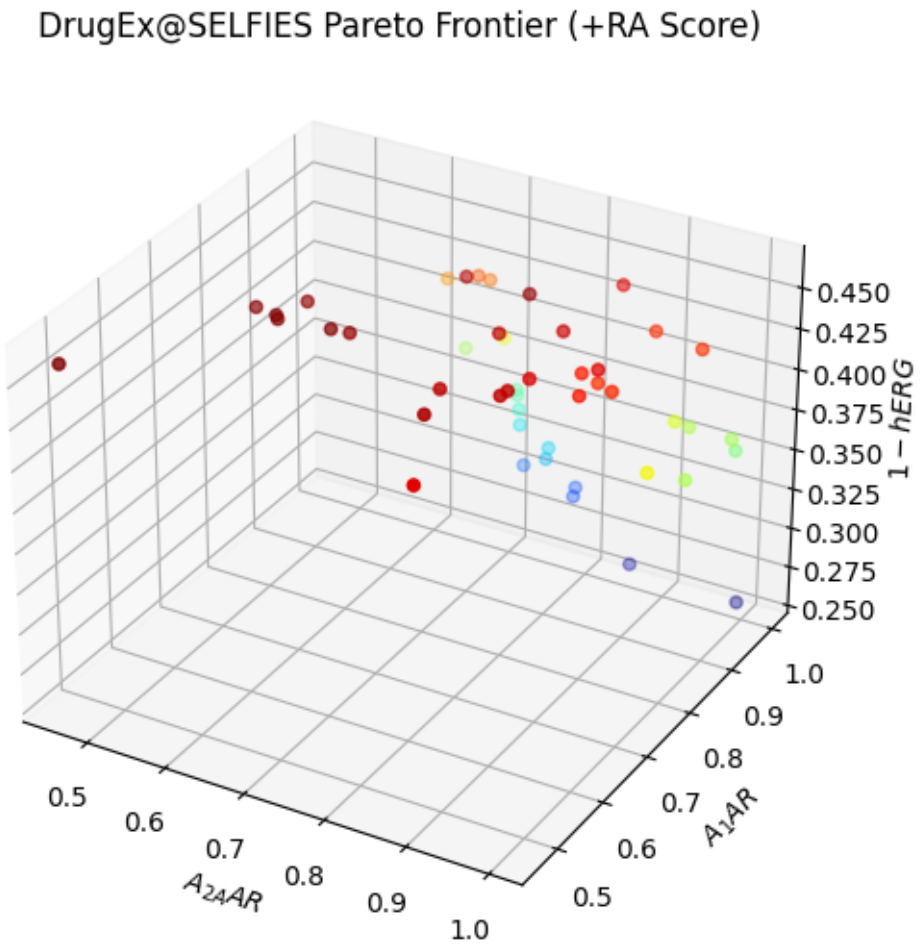
DrugEx@SELFIES Pareto Frontier (+RA Score)
0.450
0.425
0.400
0.375
0.350
0.325
0.300
0.275
0.250
1 − hERG
0.5
0.6
0.7
0.8
0.9
1.0
A2AAR
0.5
0.6
0.7
0.8
0.9
1.0
A1AR

FIGURE A.8

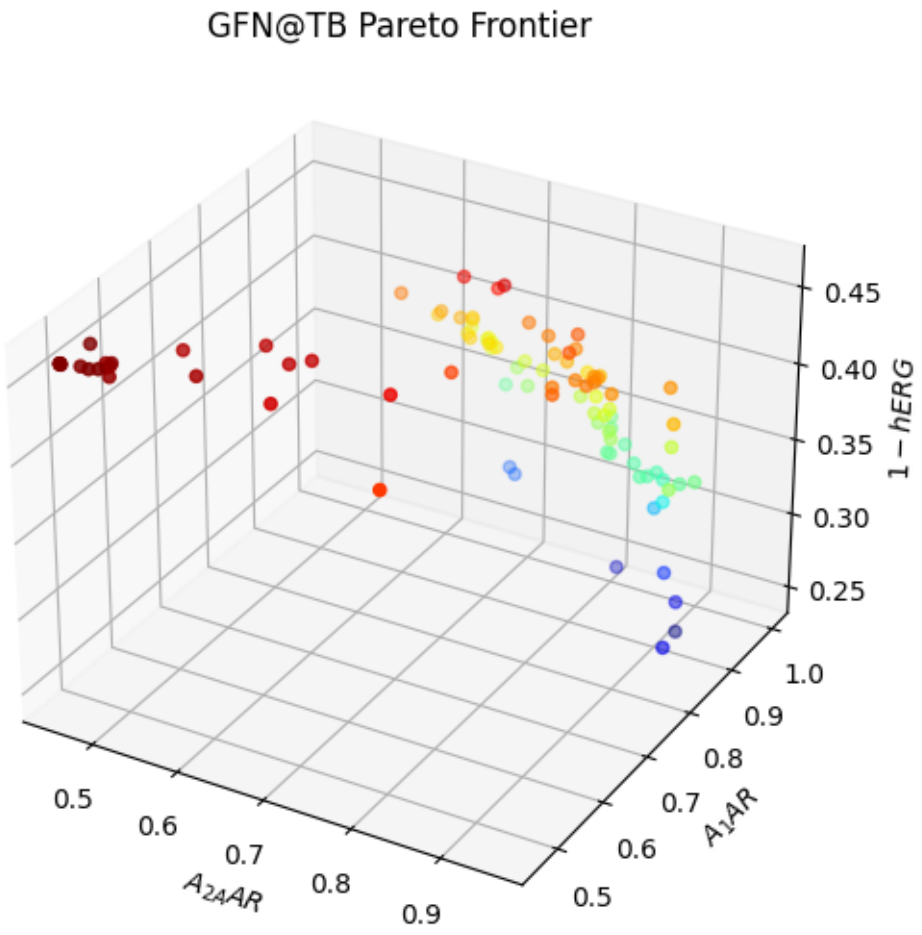
GFN@TB Pareto Frontier
0.45
0.40
0.35
0.30
0.25
1 − hERG
1.0
0.9
0.8
0.7
0.6
0.5
A1AR
0.5
0.6
0.7
0.8
0.9
A2AAR

FIGURE A.9

Appendix B

Additional information

Linear Layer Dims	Activation Fn.	Dropout Pr.
(2048, 512)	-	-
(512, 512)	ReLU	0.45834579304621176
(512, 128)	-	0.20214636121010582
(128, 512)	ELU	0.13847113009081813
(512, 256)	-	0.21312873496871235
(256, 128)	ReLU	0.33530504087548707
(128, 128)	-	0.11559123444807062
(128, 128)	ReLU	0.2618908919792556
(128, 512)	ReLU	0.3587291059530903
(512, 512)	SELU	0.43377277017943133
(512, 1)	Sigmoid	-

TABLE B.1: Exact structure and hyperparameters used for RA Score.

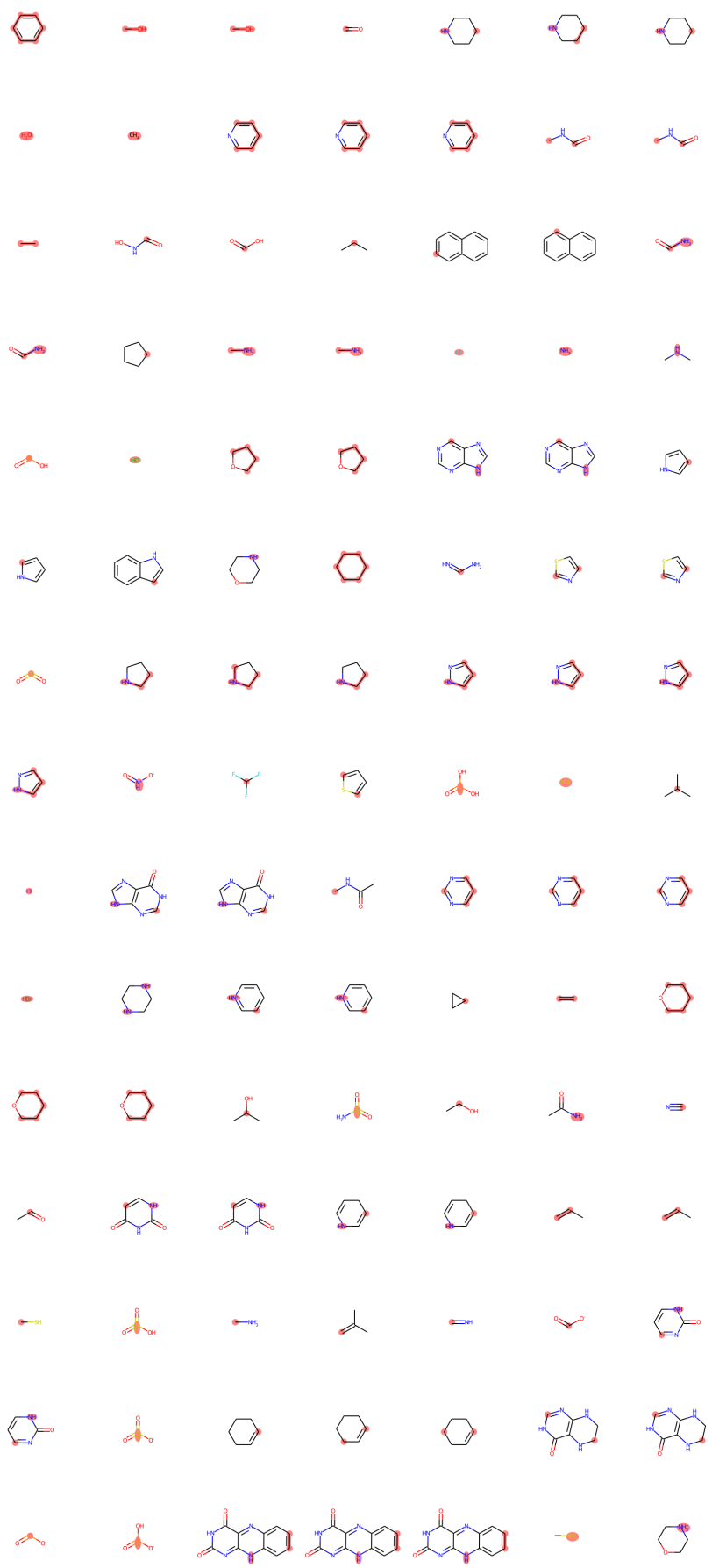

FIGURE B.1: All 105 fragments available as used in the original GFlowNet paper, along with the atoms that can be used as attachment points (stems) highlighted. Some fragments appear more than once as they can have different stems, or their stems are ordered differently.