PROCEDURAL SYNTHESIS OF SYNTHESIZABLE MOLECULES

Michael Sun[1], **Alston Lo**[1], **Minghao Guo**[1], **Jie Chen**[2], **Connor Coley**[3], **Wojciech Matusik**[1]
[1]MIT CSAIL [2]MIT-IBM Watson AI Lab, IBM Research [3]MIT Chemical Engineering
{msun415,alston,wojciech}@csail.mit.edu, chenjie@us.ibm.com,
ccoley@mit.edu

ABSTRACT

Designing synthetically accessible molecules and recommending analogs to unsynthesizable molecules are important problems for accelerating molecular discovery. We reconceptualize both problems using ideas from program synthesis. Drawing inspiration from syntax-guided synthesis approaches, we decouple the syntactic skeleton from the semantics of a synthetic tree to create a bi-level framework for reasoning about the combinatorial space of synthesis pathways. Given a molecule we aim to generate analogs for, we iteratively refine its skeletal characteristics via Markov Chain Monte Carlo simulations over the space of syntactic skeletons. Given a black-box oracle to optimize, we formulate a joint design space over syntactic templates and molecular descriptors and introduce evolutionary algorithms that optimize both syntactic and semantic dimensions synergistically. Our key insight is that once the syntactic skeleton is set, we can amortize over the search complexity of deriving the program's semantics by training policies to fully utilize the fixed horizon Markov decision process imposed by the syntactic template. We demonstrate performance advantages of our bi-level framework for synthesizable analog generation and synthesizable molecule design. Notably, our approach offers the user explicit control over the resources required to perform synthesis and biases the design space towards simpler solutions, which is particularly promising for autonomous synthesis platforms. Supporting code is at https://github.com/shiningsunnyday/SynthesisNet.

1 INTRODUCTION

The discovery of new molecular entities is central to advancements in fields such as pharmaceuticals (Zhavoronkov et al., 2019; Lyu et al., 2019), materials science (Hachmann et al., 2011; Janet et al., 2020), and environmental engineering (Zimmerman et al., 2020; Yao et al., 2021). Traditional make-design-test workflows for molecular design typically rely on labor-intensive methods that involve a high degree of trial and error (Sanchez-Lengeling & Aspuru-Guzik, 2018). Systematic and data-efficient approaches that minimize costly experimental trials are the key to accelerating these processes (Coley et al., 2020a;b; Gao et al., 2022). In recent years, a large number of molecular generative models has been proposed (De Cao & Kipf, 2018; Ma et al., 2018; Simonovsky & Komodakis, 2018; You et al., 2018; Li et al., 2018; Samanta et al., 2020; Liu et al., 2018; Jin et al., 2018; 2020; Guo et al., 2022; Sun et al., 2024). However, few of their outputs are feasible to make and proceed to experimental testing due to their lack of consideration for synthesizability (Gao & Coley, 2020). This has motivated recent methods that integrate design and synthesis into a single workflow, aiming to optimize both processes simultaneously (Button et al., 2019; Bradshaw et al., 2019; 2020; Gao et al., 2021; Swanson et al., 2024) which significantly closes the gap between the design and make steps, reducing cycle time significantly (Koscher et al., 2023; Volkamer et al., 2023; McCorkindale, 2023). This development is spurred by the curation of a small but robust collection of expert reaction templates that are inspired by real-world reactions and defined in close collaboration with chemists Hartenfeller et al. (2011). This workflow facilitates de novo applications by imposing synthetic accessibility by design, recasting molecular design as navigating the space of possible synthetic procedures over a set of building blocks and forward reaction steps, as defined in Vinkers et al. (2003). However, these methods still face computational challenges, particularly in navigating the combinatorial explosion of potential synthetic procedures (Smith, 1997).

Inspired by techniques in program synthesis, particularly syntax-guided synthesis (Alur et al., 2013), our method decouples the syntactical template of a synthetic procedure (the *skeleton*) from their chemical semantics (the *substance*). This bifurcation allows for a more granular optimization process, wherein the syntactical and semantic aspects of reaction pathways can be optimized independently yet synergistically. Our methodology employs a bi-level optimization strategy. The upper level optimizes the syntactic template of the synthetic pathway, and the lower level fine-tunes the molecular descriptors within that given structural framework. This dual-layered approach is facilitated by a surrogate policy, implemented by a graph neural network, that propagates embeddings top-down following the topological order of the syntactical skeleton. This ensures that each step in the synthetic pathway is optimized in context, respecting the overarching structural strategy while refining the molecular details. We address the combinatorial explosion in the number of programs using tailored strategies for fixed horizon Markov decision process (MDP) environments. This algorithm amortizes the complexity of the search space through predictive modeling and simulation of Markov Chain Monte Carlo (MCMC) processes (Metropolis et al., 1953; Hastings, 1970; Gilks et al., 1995), focusing on the generation and evaluation of syntactical skeletons. By leveraging the inductive biases from retrosynthetic analysis without resorting to retrosynthesis search, our approach combines accuracy and efficiency in “synthesizing" synthetic pathways. In summary, the contributions of this work are:

1. We reconceptualize molecule design and synthesis as a conditional program synthesis problem, establishing common terminology for bridging the two fields.
2. We propose a bi-level framework that decouples the syntactical skeleton of a synthetic tree from the program semantics. Then, we introduce amortized algorithms within our framework for the tasks of synthesizable analog recommendation and synthesizable molecule design.
3. We demonstrate improvements across multiple dimensions of performance for both tasks, and include in-depth visualizations and analyses for understanding the source of our method’s efficacy as well as its limitations.

2 Related Works

2.1 Synthesizable Analog Generation

The problem of synthesizable analog generation aims to find molecules close to the target molecule that are *synthesizable*. Closely related but distinct from this problem is computer-assisted retrosynthetic analysis, which has developed through the decades (Corey et al., 1985) in tandem with computers, and is now known as retrosynthetic planning due to its resemblance to more classical tests of AI based around planning. As retrosynthetic planning is done by working backwards (top-down), partial success is not straightforward to define. In other domains, procedural modeling is a bottom-up generation process that generates analogs by design (Merrell & Manocha, 2010; Merrell et al., 2011; Müller et al., 2006; Merrell, 2023). Thus, synthesizable analog generation warrants more specialized methods. Prior works such as Dolfus et al. (2022); Levin et al. (2023) address this by performing alterations of existing retrosynthesis routes, but this constrained approach severely limits the diversity of analogs. Instead, we neither start from a search route nor constrain the search route, but instead, extract analogs via the iterative refinement of the program’s syntactical skeleton with inner loop decoding of the program semantics in a bi-level setup. We implement the iterative refinement phase using an MCMC sampler with a stationary distribution governed by similarity to the target being conditioned on. This is a common technique used to search over procedural models of buildings, shapes, and furniture arrangements (Merrell et al., 2011; Talton et al., 2011; Yu et al., 2011), and we showcase its efficacy for the new application domain of molecules.

2.2 Synthesizable Molecule Design

The problem of synthesizable molecule design is to design the synthetic pathway, or *program*, whose output molecule optimizes some property oracle function. Note that unlike generic molecular optimization approaches, the design space is reformulated to guarantee synthesizability by construction. The early works to follow this formulation (Vinkers et al., 2003; Hartenfeller et al., 2011; Button et al., 2019) use machine learning to assemble molecules by iteratively selecting building blocks and virtual reaction templates to enumerate a library, with recent works such as Swanson et al. (2024) obtaining experimental validation. The key computational challenge these methods must address is how to best navigate the combinatorial search space of synthetic pathways. Prior efforts using bottom-up generation (Bradshaw et al., 2019; 2020) probabilistically model synthetic trees as sequences of actions. These adopt an encoder-decoder approach to map between a continuous latent space and a complex combinatorial space. This results in low reconstruction accuracy and hinders the method on the task of conditional generation. Instead, SynNet (Gao et al., 2021) admits a unified framework for solving analog generation and molecule design by formulating the problem as an infinite-horizon MDP and doing amortized tree generation conditioned on Morgan fingerprints. We show improvements to both tasks through our novel formulation.

2.3 Program Synthesis

Program synthesis is the problem of synthesizing a function f from a set of primitives and operators to meet some correctness specification. A program synthesis problem entails: (1) a background theory T that is the vocabulary for constructing formulas, (2) a correctness specification, i.e., a logical formula involving the output of f and T, and (3) a set of expressions L that f can take on, described by a context-free grammar G_L. In molecular synthesis, we can formulate T as containing operators for chemical reactions, constants for reagents, molecular graph isomorphism checking, and so forth. The correctness specification for finding a synthesis route for a molecule M is simply $f(\mathcal{B}) = M$, where $\mathcal{B}$ is a set of building blocks, and we seek to find an implementation f from L to meet the specification. A coarser specification is to match the molecule's fingerprint, $\text{FP}(f(\mathcal{B})) = \text{FP}(M)$, and as shown by Gao et al. (2021), this relaxed formulation enables both analog generation and fingerprint-based molecule optimization. Our key innovation takes inspiration from the line of work surrounding syntax-guided synthesis (Alur et al., 2013; Schkufza et al., 2013). Syntax guidance explicitly constrains G_L, which reduces the set of implementations f can take on (Alur et al., 2013) and facilitates more accurate amortized inference. Further discussion on the connections between program synthesis and molecular synthesis and the similarities to the literature of retrosynthesis are given in App. D.

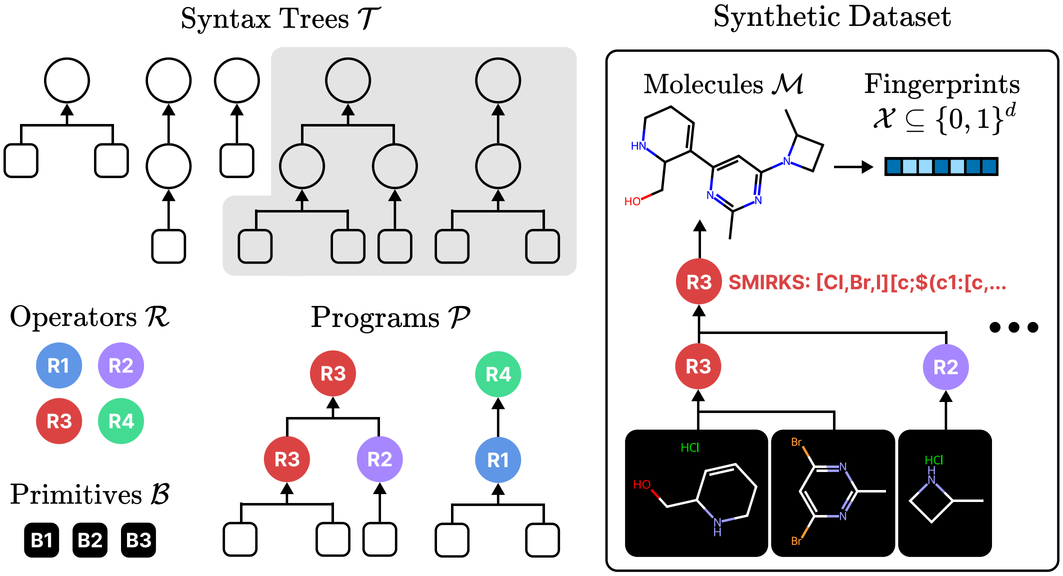


Figure 1: Program synthesis terminology for modeling synthesis pathways.

3 Methodology

3.1 Problem Definition

For clarity, we recapitulate the problems of interest discussed in Section 2:

Synthesizable analog generation is the inverse task of inferring the program and inputs that best reconstructs a target molecule. Denoting the set of reactions as $\mathcal{R}$ and the set of building blocks as $\mathcal{B}$, we obtain a compact yet expressive design space $\mathcal{P}$: all non-trivial, attributed binary trees where each internal node corresponds to a reaction. Drawing parallels to the program synthesis literature, we call $\mathcal{P}$ the *program* space. The problem can now be formalized: given a space of molecules $\mathcal{M}$, learn a mapping $F\colon \mathcal{M} \to \mathcal{P} \times \mathcal{B}^*, M \mapsto (P, B)$ such that B can be assigned to the leaf nodes of P and running the reaction(s) in P in a bottom-up manner (by recursively feeding the product of a node's reaction to its parent) produces a molecule M_* with minimal "distance" to M. Using program execution notation, the objective is stated as: $\arg\min_{(P,B)\in\mathcal{P}\times\mathcal{B}^*} \text{dist}(P(B), M)$.

Synthesizable molecule design is the forward task of finding the program and inputs whose output optimizes a property oracle function. The oracle can represent property predictors, simulation, experimental validation, etc. but are black-box in nature. The objective is $\arg\min_{(P,B)} \text{Oracle}(P(B))$.

Moving forwards, we identify (P, B) with its output M_* when it simplifies notation. Stripping the semantics from a program leaves behind a syntactic skeleton, which lies in the space $\mathcal{T}$ of non-trivial binary trees. Figure 1 illustrates the discussed terminologies[1].

3.2 Solution Overview

Herein, we use expert-defined reaction templates, a popular abstraction codifying deterministic graph transformation patterns in the SMIRKS formal language. SMIRKS assumes the reactants are ordered for defining how atoms and bonds are transformed from reactants to products. Since templates are designed to encompass the most common and robust chemical transformations, ours are restricted to uni-molecular (e.g., isomerization, ring closure, or fragmentation reactions) or bi-molecular (e.g., additions, substitutions, or coupling reactions) reactions. In practice, we featurize molecules using Morgan fingerprints with radius 2 and $d = 2048$ bits, which is a common molecular representation in both predictive and design tasks. This means that F is now technically a map over fingerprint space $\mathcal{X} \subseteq \{0, 1\}^d$. It is then natural to use the Tanimoto distance between fingerprints as our notion of molecular distance.

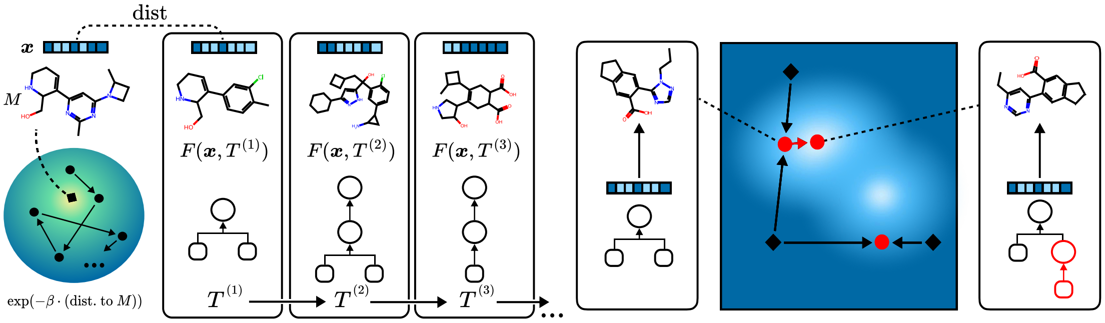

dist
$\boldsymbol{x}$
M
$F(\boldsymbol{x}, T^{(1)})$
$F(\boldsymbol{x}, T^{(2)})$
$F(\boldsymbol{x}, T^{(3)})$
$\exp(-\beta \cdot (\text{dist. to } M))$
$T^{(1)}$
$T^{(2)}$
$T^{(3)}$
...

Figure 2: (Left) Our Metropolis-Hastings algorithm in Section 3.3 iteratively refines the syntax tree skeleton towards the stationary distribution which is proportional to the inverse distance to our target molecule M. (Right) Our genetic algorithm over the joint design space $\mathcal{X} \times \mathcal{T}$ in Section 3.4 combines the strategies of semantic crossover (→) and syntactical mutation (→) to encourage both global improvement and local exploration.

3.3 Bilevel Syntax-Semantic Synthesizable Analog Generation

Given a molecule $M \in \mathcal{M}$, we aim to find a program and inputs (P, B) whose output M_* is most similar to M. Suppose first that the syntax T of P was given. Then, we would use two learned policies $\pi_\mathcal{R}$ and $\pi_\mathcal{B}$ that iteratively attribute reactions and building blocks to the nodes of T in topological order (Section 3.3.2). Recalling Section 2.3, this can be seen as parameterizing the derivation process of a context-free grammar $G_\mathcal{P}$. We give further details in App. E and discuss how these derivations are constrained through syntax.

[1] Please refer to Figure 3 of Gao et al. (2021) for example chemical illustrations of the two tasks.

The application of our policies essentially specifies a syntax-conditioned program synthesis map $F\colon \mathcal{M} \times \mathcal{T} \to \mathcal{P} \times \mathcal{B}^*$. However, T is not known during inference, so to remove this dependence, we consider two further strategies. One is to learn a classifier $\tau\colon \mathcal{M} \to \mathcal{T}$ to predict the most likely syntax tree of the program that produces M. We implement τ with a multi-layer perceptron (MLP) trained under a standard classification task.[2] However, a single prediction may be inadequate, since there can be multiple skeletons corresponding to the same molecule. Instead, our second strategy uses a bi-level setup, where the outer loop explores syntax space $\mathcal{T}$ through invocations of the inner loop F, which explores the program's semantics. This approach is further made efficient by amortizing the training of the inner loop.

3.3.1 Outer loop: Syntax Tree Structure Optimization

We simulate a Markov process on $\mathcal{T}$ for discovering skeletons whose decoding will maximize the similarity between M and the program's output (Figure 2 (Left)). The details for how we bootstrap and apply $\mathcal{T}$ is in App. A. We adopt the Metropolis-Hastings algorithm with proposal distribution $q(T \mid T_0) \propto \exp(-\lambda\, d_{\mathcal{T}}(T, T_0))$ and scoring function $f(T) \propto \exp(-\beta\, d_{\mathcal{M}}(M, F(M, T)))$, where λ and β are parameters that trade-off exploration with exploitation, $d_{\mathcal{T}}$ is the tree edit distance, and $d_{\mathcal{M}}$ is the Tanimoto distance. In other words, we use the inner loop to score candidates in $\mathcal{T}$.

3.3.2 Inner loop: Inference of Tree Semantics

We now formulate syntax-conditioned derivations under $G_{\mathcal{P}}$ as a finite-horizon MDP.

State space: To bridge $\mathcal{T}$ and $\mathcal{P} \times \mathcal{B}^*$, we introduce an intermediate state space of partial programs $\partial\mathcal{P}$ consisting of all possible partial programs arising from the following modifications to any syntax tree $T \in \mathcal{T}$: (1) prepend a new root node of T, and attribute it with a fingerprint from $\mathcal{X}$, or (2) attribute some internal (resp. leaf) nodes of T with elements of $\mathcal{R}$ (resp. $\mathcal{B}$). We further require that if a node in T is filled, then so is its parent. Intuitively, $\partial\mathcal{P}$ comprises all partially filled-in trees in $\mathcal{T}$ obeying topological order (with an added root node attached to a molecular fingerprint).

Action space: At a state $S \in \partial\mathcal{P}$, the actions are to attribute any frontier node, i.e., unfilled nodes whose parents are filled, with an item from $\mathcal{R}$ (resp. $\mathcal{B}$) if the node is internal (resp. a leaf).

Policy network: We parameterize the policies $\pi_{\mathcal{R}}\colon \partial\mathcal{P} \to \mathcal{R}^*$ and $\pi_{\mathcal{B}}\colon \partial\mathcal{P} \to \mathcal{B}^*$ with separate graph neural networks. Given a partial program S as input, the former predicts the reactions that should be attributed to internal frontier nodes as an $|\mathcal{R}|$-label classification problem, while the latter does so for building blocks and leaf nodes. However, $|\mathcal{B}| \gg |\mathcal{R}|$ is very large, so $\pi_{\mathcal{B}}$ instead predicts a 256-dimensional embedding at each nodes which implicitly specifies the building block whose 256-bit Morgan fingerprint is closest.

Training: We train both policies using supervised policy learning. Key to this approach is the dataset used for training constructed using Algorithm 1. See App. F for further details.

Algorithm 1 Construction of training dataset.

```
Require: A synthetic dataset $\mathcal{D}_0 \subseteq \mathcal{P} \times \mathcal{B}^*$ of programs (Section 4.1.1).
$\mathcal{D} \leftarrow \varnothing$
for each $(P, B) \in \mathcal{D}_0$ do
    Turn $(P, B)$ into a fully-filled program $T \in \partial\mathcal{P}$ whose root is attributed with FP$(P, B)$.
    for each $\Lambda \in 2^T$ containing the root and closed under parent$(\cdot)$ do
        Frontier$(\Lambda) \leftarrow \{i \in T \mid i \notin \Lambda$ and parent$(i) \in \Lambda\}$
        Populate node features $\boldsymbol{H}$ and labels $\boldsymbol{Y}$ based on $P$ and $B$ (App. F)
        for $i \in T - \Lambda$ do
            Mask the feature in $\boldsymbol{H}$ corresponding to node $i$
        for $i \in T -$ Frontier$(\Lambda)$ do
            Mask the label in $\boldsymbol{Y}$ corresponding to node $i$
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{(T, \boldsymbol{H}, \boldsymbol{Y})\}$
return $\mathcal{D}$
```

[2] In practice, we restrict to a finite subset of $\mathcal{T}$ by imposing a maximum number of reaction nodes.

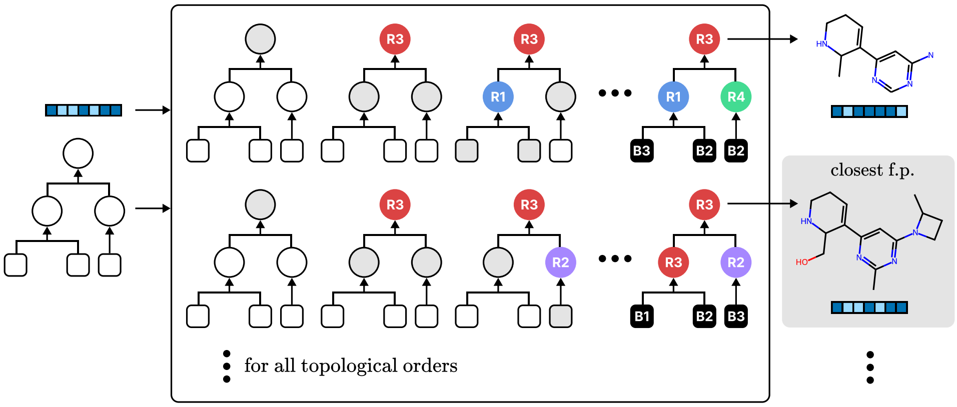


Figure 3: Illustration of our decoding scheme F: (Left) The input is a Morgan fingerprint $\boldsymbol{x}$ and syntax skeleton T; (Middle) Decode once for every topological ordering of the tree, tracking all partial programs with a stack; (Right) Execute all decoded programs, then returning the closest analog which minimizes distance to $\boldsymbol{x}$.

3.4 Bi-level Syntax-Semantic Synthesizable Molecular Design

The task of synthesizable molecule design is to find a program P and building blocks B whose M_* maximizes a property of interest. Given the learned policies from synthetic planning, we apply the inner loop procedure $F\colon \mathcal{X} \times \mathcal{T} \rightarrow \mathcal{P} \times \mathcal{B}^*$ as a surrogate, casting the optimization problem as one over the joint design space $\mathcal{X} \times \mathcal{T}$. We approach this problem with a genetic algorithm (GA) over fingerprint space $\mathcal{X}$ that leverages this extra dimension of syntax $\mathcal{T}$ (Figure 2 (Right)). The seed population is obtained by sampling random bit vectors. To generate a child $\boldsymbol{x}$ from two parents $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, we combine *semantic* crossover with *syntactical* mutation, reminiscent of our bi-level approach for analog generation:

Semantic crossover: We first generate $\boldsymbol{x}_*$ by combining bits from both $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ and possibly mutating a small number of bits of the result.

Syntactic mutation: We set $T = \tau(\boldsymbol{x}_*)$ and apply random edit(s) to obtain a perturbed tree T'. Together, $(\boldsymbol{x}_*, T)$ and $(\boldsymbol{x}_*, T')$ form a sibling pool. Applying the surrogate F to each child gives two sibling SMILES that are turned into two sibling fingerprints. We fit a Gaussian process on past individuals and select the sibling with the highest expected improvement as the favoured child $\boldsymbol{x}$.

Intuitively, semantic crossover optimizes for chemical semantics by combining existing ones from the mating pool, while syntactic mutation explores syntactic analogs of individuals of interest. Each child is then given a fitness score under the property oracle, and the top scoring unique individuals are retained into the next generation (i.e., elitist selection). Further details and hyperparameters are given in App. G.

4 Experiments

4.1 Experiment Setup

4.1.1 Data Generation

We use 91 reaction templates from Hartenfeller et al. (2011); Button et al. (2019) representative of common synthetic reactions. They consist primarily of ring formation and fusing reactions but also peripheral modifications. Starting from 147,505 purchaseable compounds from Enamine Building Blocks, we follow the same steps as Gao et al. (2021) to generate 600,000 synthetic trees. Filtering by QED > 0.5 of the product molecules leaves 227,808 synthetic trees (136,684 for training, 45,563 for validation, and 45,561 for testing), which are then preprocessed into programs to construct our final datasets. We bootstrap our set of syntactic templates based on those observed in the training set, resulting in 1117 syntactic skeleton classes. Additional statistics on these syntactic templates and insights on their coverage are given in App. A. Further analyses of the structure-property relationship and a detailed case study are given in App. C.

4.1.2 Baselines

We evaluate against all 25 molecular optimization methods benchmarked in the large-scale study by Gao et al. (2022). Most of these are divided into three categories based on the molecular representation used: string, graph, or synthesis trees. Synthesis methods restrict the design space to only products of robust template-based reactions, so for fair comparison, we also consider intra-category rankings. We report the synthetic accessibility (SA) score (Ertl & Schuffenhauer, 2009) of the optimized molecules to cross-verify the synthesizability of synthesis-based methods as well a investigate the performance trade-off imposed by constraining for template-compatible synthesizability.

4.1.3 Computational Efficiency

Constructing $\mathcal{D}$. Alg. 1 suggests $|\mathcal{D}| = O(2^{\max_{\mathcal{T}} |T|}|\mathcal{D}_0|)$ because we compute all the (closed under parent$(\cdot)$) masks per $P \in \mathcal{D}_0$. However, we don't need to explicitly store $\mathcal{D}$. We only need to precompute the (closed under parent$(\cdot)$) masks for each $T \in \hat{\mathcal{T}}_k$ (k fixed and small), so the running time is $O(\sum_{T \in \hat{\mathcal{T}}_k} 2^{|T|})$, independent of $\mathcal{D}_0$. Besides, the actual number of masks is far smaller than $2^{|T|}$, and high $|T|$ is less represented in $\mathcal{D}_0$, since large programs are less likely to be sampled, so in practice $|\mathcal{D}|$ is much smaller than $2^{\max_{\mathcal{T}} |T|}|\mathcal{D}_0|$. Detailed statistics are given in App A.

Training with $\mathcal{D}$. During training, we flat-index into $\mathcal{D}_0$ and their precomputed masks to perform a pass over $\mathcal{D}$. Despite the larger dataset, we find the total training steps to actually be comparable with Gao et al. (2021). To scale to larger k, we propose a stratified sampling strategy that does only a $O(\mathcal{D}_0)$ pass per epoch, has positive support over $\mathcal{D}$, and represents each $(P, B) \in \mathcal{D}_0$ equally. The idea is to sample a constant number of masks C per $P, B \in \mathcal{D}_0$, and re-sample each epoch. We show this has complexity linear to the *size* of $\mathcal{D}_0$ per epoch but empirically converges in much fewer training steps than SynNet for $C = 1$. Details are in App F.7. We further include an ablation study on the downstream task performance of this simplified training strategy in App F.5.

4.1.4 Evaluation Metrics

Synthesizable analog generation. We evaluate the ability to generate a diverse set of structural analogs to a given input molecule using the Recovery Rate (RR, whether the most similar analog is exactly the target), Average Similarity (as measured by Tanimoto distance to the input), SA score, and Internal Diversity (average pairwise Tanimoto distance).

Synthesizable molecule design. We evaluate our method's ability to optimize 15 oracle functions (Huang et al., 2022) relevant to real-world drug discovery:

1. **Bioactivity predictors** (GSK3β, JNK3, DRD2) that estimate responses against targets related to the pathogenesis of various diseases such as Alzheimer's disease (Koch et al., 2015) based on experimental data (Sun et al., 2017), and whose inhibitors are the basis for many antipsychotics and have shown promise for treating diseases like Parkinson's schizophrenia and bipolar disorder (Madras, 2013).
2. **Structural profiles** (Median1, Median2, Rediscovery) that primarily focus on maximizing structural similarity to multiple molecules, which is useful for designing molecules fitting a more multifaceted structural profile (Brown et al., 2004). The rediscovery oracle focuses on hit expansion around a specific drug.
3. **Multi-property objectives** (Osimertinib, 6 others) that use real drugs as a basis for optimizing additional pharmacological properties, mimicking real-world drug discovery.
4. **Docking Simulations** (M^{pro}, DRD3) against M^{pro}, the main protease of SARS-Cov-2, and DRD3, which has its own leaderboard with a particular focus on sample efficiency.

In addition to the average score of the top k molecules, we particularly focus on sample efficiency, i.e., the top-k AUC as described in Gao et al. (2022).

4.2 Results on Synthesizable Analog Generation

In Table 1, we see our method outperforming SynNet across both dimensions of similarity (how "analog" compounds are) and diversity (how different the compounds are). Additionally, our method

Table 1: We generate 5 unique analog molecules conditioned on an input molecule M and sort them by decreasing similarity to M. For SynNet, we follow their beam search strategy and produce analogs using the top 5 beams. For Ours (τ), we sample the top 5 syntactic templates from τ. Ours (τ, rev) is the same except we use a bottom-up decoding process, and it is included as an ablation for Section 4.4.1. Then, we evaluate how similar, diverse, and structurally simple the first k molecules are. The best method is bolded. For three-way comparisons, the second best method is underlined.

			Avg. Sim. ↑			SA ↓			Diversity ↑	
Dataset	Method	RR ↑	Top-1	Top-3	Top-5	Top-1	Top-3	Top-5	Top-3	Top-5
Train Set	Ours (τ, rev)	79.3 %	0.923	0.632	0.569	**3.072**	**2.795**	**2.716**	**0.615**	**0.657**
	Ours (τ)	**88.1%**	**0.958**	**0.704**	**0.626**	3.099	2.928	2.852	0.532	0.615
Test Set	SynNet	46.3%	0.766	**0.622**	**0.566**	3.108	3.057	3.035	0.525	0.584
	Ours (τ, rev)	40.8 %	0.749	0.548	0.487	**2.970**	**2.743**	**2.659**	**0.640**	**0.685**
	Ours (τ)	**52.3%**	**0.799**	0.588	0.548	3.075	2.895	2.856	0.609	0.653
ChEMBL	SynNet	4.9%	0.499,	0.436,	0.394	2.669,	2.685,	2.697	0.644,	0.693
	Ours (τ)	7.6%	0.531,	0.443,	0.396	2.544,	2.510,	2.460	0.675,	0.727
	Ours (MCMC)	**9.2%**	**0.532,**	**0.486,**	**0.432**	**2.364,**	**2.310,**	**2.263**	**0.765,**	**0.759**

achieves lower SA Score, which is a proxy for synthetic accessibility that rewards simpler molecules. Guided by a set of simple yet expressive syntactic templates, our model simultaneously produces more diverse and structurally *simple* molecules without sacrificing one for the other. Additionally, our policy network is well-suited to navigate these simple yet horizon structures, enabling a 6% higher reconstruction accuracy after training on the same dataset. Combining these three dimensions, we can conclude our method is the superior one for the task of synthesizable analog generation. To better understand which design choices are responsible for the performance, we provided a comprehensive analysis of the policy network in App. F. We begin in App. F.3 by elaborating on the main distinction of our method vs existing works, highlight the novelty of our formulation, and motivate an auxiliary training task that takes inspiration from cutting-edge ideas in inductive program synthesis. We then perform several key ablations in App. F.4, using concrete examples to highlight success and failure cases. Lastly, we perform a step-by-step walkthrough of our decoding algorithm in App. F.8, visualizing the evolution of attention weights to showcase the full-horizon awareness of our surrogate and the dynamic incorporation of new information. These analyses shed insights into why our surrogate works, and points to future extensions to make it even better.

Table 2: Our model's average performance across 13 TDC oracles compared to baselines compiled in Gao et al. (2022). We limit to 1000 oracle calls each run and normalize oracle outputs to $[0, 1]$. We report to mean score, AUC, and SA scores (Ertl & Schuffenhauer, 2009) of the top 10 molecules. Methods are categorized by Gao et al. (2022) and, for brevity, we display only the top three algorithms within each category with respect their AUC. The best method per column is bolded and the best synthesis-based method is underlined. See App. H for the full results and experiment details.

		Score ↑		AUC ↑		SA ↓	
Category	Method	Value	Rank	Value	Rank	Value	Rank
Screening	Screening	0.426	20	0.377	20	3.097	8
	MolPAL	0.472	16	0.444	15	3.018	4
String	REINVENT	0.697	2	0.607	2	3.415	9
	REINVENT-SELFIES	0.682	3	0.578	4	3.791	15
	STONED	0.609	8	0.555	6	5.550	24
	7 rows omitted · · ·						
Graph	Graph-GA	**0.701**	1	0.601	3	3.982	17
	GPBO	0.642	6	0.570	5	3.954	16
	DST	0.555	10	0.479	11	4.146	20
	7 rows omitted · · ·						
Synthesis	SynNet	0.578	9	0.545	7	3.075	6
	DoG-Gen	0.634	7	0.511	9	2.793	2
	DoG-AE	0.460	18	0.450	14	2.857	3
	Ours	0.670	4	**0.608**	1	**2.739**	1

Table 3: AutoDock Vina scores against DRD3 and $\mathrm{M^{pro}}$, limited to 5000 oracle calls. For ZINC (Screening), we use numbers from TDC's DRD3 Leaderboard, and for SynNet, we report both their paper's numbers (*) and our reproduced results. We also report the top 3 binders for $\mathrm{M^{pro}}$ for the real-world case study in App. I.

			Score ↑					AUC ↑	SA ↓
Target	Method	#Calls	1st	2nd	3rd	Top-10	Top-100	Top-10	Top-10
DRD3	ZINC	–	12.8	–	–	12.59	12.08	–	–
	SynNet*	5000	12.3	–	–	12.02	11.13	–	–
	SynNet	5000	10.8	10.4	10.3	10.30	9.20	9.55	2.59
	Ours	5000	**13.7**	**13.1**	**13.1**	**13.01**	**12.13**	**11.91**	**2.13**
$\mathrm{M^{pro}}$	SynNet	5000	8.3	8.3	8.2	8.09	7.46	7.60	**2.27**
	Ours	5000	**9.9**	**9.7**	**9.7**	**9.54**	**9.02**	**9.01**	2.59
	SynNet*	–	10.5	9.3	9.3	–	–	–	–
	Ours	10000	**10.8**	**10.7**	**10.6**	–	–	–	–

Table 4: (Left) Ablation of sibling pool generation strategies on JNK3: (edits) mutates the syntax, (τ) uses the top skeletons predicted from τ, and (flips) doesn't consider skeleton and instead flips random bits in the fingerprint. (Right) Ablation of SynNet with our Bayesian optimization (BO) acquisition over a sibling pool generated by beam search. **Seeds** and **All** are the average scores of the initial and final populations.

	1st	2nd	3rd	Top-10	Top-100	Diversity
Ours (edits)	**0.88**	**0.88**	**0.87**	**0.86**	**0.8**	**0.61**
Ours (τ)	**0.88**	**0.88**	**0.87**	0.83	0.74	0.55
Ours (flips)	0.87	0.87	0.86	0.84	0.77	0.49

Oracle	Method	Top-1	Top-10	Top-100	Seeds	All
GSK3β	SynNet	0.94	0.907	0.815	0.050 ± 0.051	0.803 ± 0.041
	SynNet + BO	0.85	0.684	0.471	0.013 ± 0.024	0.447 ± 0.090
	Ours	**0.98**	**0.967**	**0.944**	0.074 ± 0.055	$\mathbf{0.941 \pm 0.012}$
JNK3	SynNet	0.80	0.758	0.719	0.032 ± 0.025	0.715 ± 0.017
	SynNet + BO	0.31	0.241	0.143	0.006 ± 0.012	0.134 ± 0.039
	Ours	**0.88**	**0.862**	**0.800**	0.059 ± 0.053	$\mathbf{0.792 \pm 0.030}$
DRD2	SynNet	**1.000**	**1.000**	0.998	0.007 ± 0.018	0.996 ± 0.003
	SynNet + BO	0.982	0.963	0.722	0.005 ± 0.018	0.672 ± 0.147
	Ours	**1.000**	**1.000**	**1.000**	0.024 ± 0.056	$\mathbf{1.000 \pm 0.000}$

4.3 Results on Synthesizable Molecule Design

In Table 2, we see our method outperforming *all* synthesis-based methods on average across the 13 TDC oracles for all considered metrics – average score, AUC, and SA score. Surprisingly, our method stays competitive with the state-of-the-art string and graph methods in terms of average score (ranking 4th) but being considerably more sample-efficient at finding the top molecules (ranking 1st for top-1/10 AUC). We see evidence that a synthesizability-constrained design space does not sacrifice end performance when reaping benefits of enhanced synthetic accessibility and sample efficiency.

The AutoDock Vina scores reflect our method's strength in real-world ligand design tasks. Our best binders against $\mathrm{M^{pro}}$ in Table 3 are significantly better than nearly all known inhibitors from virtual screening or literature (Ghahremanpour et al., 2020; Zhang et al., 2021). For example, Zhang et al. (2021) report a best score of -8.5. Our best binders against DRD3 also rank us 3rd on the TDC Leaderboard (as of Aug. 2024). We present additional analysis of the best binders for our two docking targets in App. I.

4.4 Ablation Studies

In Tables 1 and 4, we analyze findings from four carefully designed ablation studies (1 in 4.4.1, 2 & 3 in 4.4.2, 4 in 4.4.3) to justify the key design decisions that differentiate our method from the predecessor SynNet as well as other synthesis-based methods that have similar modules.

4.4.1 Top-down vs Bottom-up Decoding.

Should we decode top-down or bottom-up? This separation exists between retrosynthesis vs molecular design. Baselines that serialize the construction of synthesis trees (Bradshaw et al., 2019; 2020; Gao et al., 2021) adopt the latter. Given only knowledge of the target molecule, the model has to predict the first building block, the first reaction, and so on. Our empirical findings confirm observations by Gao et al. (2021) that the first few steps are difficult, due to inherent ambiguity – multiple valid choices exist. Our method resolves this bottleneck by reformulating the (Markov) state as partial syntax trees, where holes are reactions and building blocks left to predict. This state captures the horizon structure, so we can learn tailored policies for the fixed horizon. We reintroduce the inductive bias of retrosynthetic analysis to procedural synthesis. Conditioned on the syntax (skeleton), we show a top-down filling order outperforms bottom-up, with two intuitive explanations: (1) there are orders of magnitude less reactions than building blocks (91 vs. 147,505) and (2) it is easier to reason backwards from the specification (target molecule) which reactions lead to the product. To demonstrate these factors compensate for any gains from bottom-up pruning of applicable reactions, we perform an additional ablation in Table 1. Instead of the MDP enforcing we fill in a skeleton top-down, we fill the skeleton from the bottom-up. We retrain the model by pretraining on inverted masks, and decode by following every possible topological order of the skeleton with edges reversed. The results show this cannot reconstruct the training data as well and struggles on generalization.

4.4.2 How Analog Generation Capabilities Translate to Design Capabilities.

Why does our superior analog generation capability translate to better performance when used as an offspring generator within molecule optimization? An ideal surrogate takes as input a fingerprint and outputs multiple synthesizable analogs, creating diverse offspring(s) that balance local neighborhood exploration with global exploitation. SynNet does the former by mutating the fingerprint directly, whereas the key insight of our syntax-guided method is to mutate the syntactic skeleton instead, doing so via editing mutations. Table 4 (Left) shows edit-based mutations are superior to the top recognition strategy used for analog generation ((Skeleton) in Table 1) and the trivial strategy of ignoring the skeleton and flipping individual bits to obtain siblings. This suggests edits to the skeleton better preserves the locality bias within the GA. Higher population diversity and average scores for $k \in \{10, 100\}$ suggest the same symbiotic relationship between diversity and similarity in analog generation is also the key enabler to better GA optimization. Our GA benefits from an inner loop sibling acquisition within the crossover operation, acquiring the highest expected improvement sibling to expend an oracle call on. It can be argued this extra mechanism is why our method gets better results and makes for an unfair comparison with SynNet, or that this is a method-agnostic hack to improve GA performance. In Table 4 (Right), we show SynNet endowed with a similar mechanism in its crossover operation (generate an offspring pool using the top beams then apply a BO acquisition step on top) didn't improve, but actually downgraded the performance. We hypothesize that SynNet's optimization trajectory is *derailed* by the additional variation to its sibling pool, reducing local movements within the output space that a syntactic editing approach naturally preserves. Thus, we believe the performance gains of this mechanism is *unlocked* by our syntax-driven approach.

4.4.3 Extrapolation to Unseen Templates.

How dependent is our framework on the initial set of syntactic templates? Can SynthesisNet generalize to more templates? Our ablation study in App. B evaluate whether the model can extrapolate to new template classes. We investigate: 1) how well SynthesisNet extrapolates to programs whose structural template was unseen during training, 2) how well the framework can incorporate new templates at test time, and 3) how robust overall task performance is to them. To answer these questions, we hold out $\approx 25\%$ of $\hat{\mathcal{T}}$ to be the test set, remove all programs belonging to those templates, retrain, and analyze changes in task performances, both holistic results and results specific to the held-out templates. Our results reveal minor performance drop, and in some instances, *improved* results along some metrics. Our analysis reveals the surrogate model is robust and unsensitive to missing templates, and the framework can also incorporate unseen templates in the online phases of the downstream tasks. We encourage future works to study the exact scaling laws between templates and performance, and we hope our initial findings unlock new directions for scaling up our solutions to achieve greater coverage and impact.

5 Discussion & Conclusion

We reconceptualize synthesis pathway design using a program synthesis approach, introducing a bi-level framework that separates the syntactical skeleton of a synthetic tree from its chemical semantics. Our learning algorithms leverage the tree horizon structure, improving performance on key metrics of analog generation and de novo design. By decoupling syntax and semantics, we effectively navigate a rich design space, integrating design and synthesis into a single workflow that reduces discovery cycle time. Our framework offers control over synthesis resources and biases towards simpler solutions, with the exciting prospect of integration with autonomous synthesis platforms (Coley et al., 2019).

REFERENCES

Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis. In *2013 Formal Methods in Computer-Aided Design*, pp. 1–8, 2013. doi: 10.1109/FMCAD.2013.6679385.

Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow network based generative models for non-iterative diverse candidate generation. *Advances in Neural Information Processing Systems*, 34:27381–27394, 2021.

Yoshua Bengio, Salem Lahlou, Tristan Deleu, Edward J Hu, Mo Tiwari, and Emmanuel Bengio. Gflownet foundations. *The Journal of Machine Learning Research*, 24(1):10006–10060, 2023.

John Bradshaw, Brooks Paige, Matt J Kusner, Marwin Segler, and José Miguel Hernández-Lobato. A model to search for synthesizable molecules. *Advances in Neural Information Processing Systems*, 32, 2019.

John Bradshaw, Brooks Paige, Matt J Kusner, Marwin Segler, and José Miguel Hernández-Lobato. Barking up the right tree: an approach to search over molecule synthesis dags. *Advances in neural information processing systems*, 33:6852–6866, 2020.

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

Rudy Bunel, Matthew Hausknecht, Jacob Devlin, Rishabh Singh, and Pushmeet Kohli. Leveraging grammar and reinforcement learning for neural program synthesis. *arXiv preprint arXiv:1805.04276*, 2018.

Alexander Button, Daniel Merk, Jan A Hiss, and Gisbert Schneider. Automated de novo molecular design by hybrid machine intelligence and rule-driven chemical synthesis. *Nature machine intelligence*, 1(7):307–315, 2019.

Binghong Chen, Chengtao Li, Hanjun Dai, and Le Song. Retro*: learning retrosynthetic planning with neural guided a* search. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

Xinyun Chen, Chang Liu, and Dawn Song. Execution-guided neural program synthesis. In *International Conference on Learning Representations*, 2018.

Connor W. Coley, Dale A. Thomas, Justin A. M. Lummiss, Jonathan N. Jaworski, Christopher P. Breen, Victor Schultz, Travis Hart, Joshua S. Fishman, Luke Rogers, Hanyu Gao, Robert W. Hicklin, Pieter P. Plehiers, Joshua Byington, John S. Piotti, William H. Green, A. John Hart, Timothy F. Jamison, and Klavs F. Jensen. A robotic platform for flow synthesis of organic compounds informed by ai planning. *Science*, 365(6453):eaax1566, 2019. doi: 10.1126/science.aax1566. URL https://www.science.org/doi/abs/10.1126/science.aax1566.

Connor W Coley, Natalie S Eyke, and Klavs F Jensen. Autonomous discovery in the chemical sciences part i: Progress. *Angewandte Chemie International Edition*, 59(51):22858–22893, 2020a.

Connor W Coley, Natalie S Eyke, and Klavs F Jensen. Autonomous discovery in the chemical sciences part ii: outlook. *Angewandte Chemie International Edition*, 59(52):23414–23436, 2020b.

Elias James Corey, Alan K Long, and Stewart D Rubenstein. Computer-assisted analysis in organic synthesis. *Science*, 228(4698):408–418, 1985.

Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018.

Uschi Dolfus, Hans Briem, and Matthias Rarey. Synthesis-aware generation of structural analogues. *Journal of Chemical Information and Modeling*, 62(15):3565–3576, 2022.

Kevin Ellis, Maxwell Nye, Yewen Pu, Felix Sosa, Josh Tenenbaum, and Armando Solar-Lezama. Write, execute, assess: Program synthesis with a repl. *Advances in Neural Information Processing Systems*, 32, 2019.

Peter Ertl and Ansgar Schuffenhauer. Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. *Journal of cheminformatics*, 1:1–11, 2009.

Alison B Flynn. How do students work through organic synthesis learning activities? *Chemistry Education Research and Practice*, 15(4):747–762, 2014.

Wenhao Gao and Connor W Coley. The synthesizability of molecules proposed by generative models. *Journal of chemical information and modeling*, 60(12):5714–5723, 2020.

Wenhao Gao, Rocío Mercado, and Connor W Coley. Amortized tree generation for bottom-up synthesis planning and synthesizable molecular design. *arXiv preprint arXiv:2110.06389*, 2021.

Wenhao Gao, Tianfan Fu, Jimeng Sun, and Connor Coley. Sample efficiency matters: a benchmark for practical molecular optimization. *Advances in neural information processing systems*, 35: 21342–21357, 2022.

Wenhao Gao, Shitong Luo, and Connor W Coley. Generative artificial intelligence for navigating synthesizable chemical space. *arXiv preprint arXiv:2410.03494*, 2024.

Mohammad M Ghahremanpour, Julian Tirado-Rives, Maya Deshmukh, Joseph A Ippolito, Chun-Hui Zhang, Israel Cabeza de Vaca, Maria-Elena Liosi, Karen S Anderson, and William L Jorgensen. Identification of 14 known drugs as inhibitors of the main protease of sars-cov-2. *ACS medicinal chemistry letters*, 11(12):2526–2533, 2020.

Wally R Gilks, Nicky G Best, and Keith KC Tan. Adaptive rejection metropolis sampling within gibbs sampling. *Journal of the Royal Statistical Society Series C: Applied Statistics*, 44(4):455–472, 1995.

Jiasheng Guo, Chenning Yu, Kenan Li, Yijian Zhang, Guoqiang Wang, Shuhua Li, and Hao Dong. Retrosynthesis zero: Self-improving global synthesis planning using reinforcement learning. *Journal of Chemical Theory and Computation*, 2024.

Minghao Guo, Veronika Thost, Beichen Li, Payel Das, Jie Chen, and Wojciech Matusik. Data-efficient graph grammar learning for molecular generation. *arXiv preprint arXiv:2203.08031*, 2022.

Johannes Hachmann, Roberto Olivares-Amaya, Sule Atahan-Evrenk, Carlos Amador-Bedolla, Roel S Sánchez-Carrera, Aryeh Gold-Parker, Leslie Vogt, Anna M Brockway, and Alán Aspuru-Guzik. The harvard clean energy project: large-scale computational screening and design of organic photovoltaics on the world community grid. *The Journal of Physical Chemistry Letters*, 2(17): 2241–2251, 2011.

Markus Hartenfeller, Martin Eberle, Peter Meier, Cristina Nieto-Oberhuber, Karl-Heinz Altmann, Gisbert Schneider, Edgar Jacoby, and Steffen Renner. A collection of robust organic synthesis reactions for in silico molecule design. *Journal of chemical information and modeling*, 51(12): 3093–3098, 2011.

W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. 1970.

Kexin Huang, Tianfan Fu, Wenhao Gao, Yue Zhao, Yusuf Roohani, Jure Leskovec, Connor W Coley, Cao Xiao, Jimeng Sun, and Marinka Zitnik. Artificial intelligence foundation for therapeutic science. *Nature chemical biology*, 18(10):1033–1036, 2022.

Jon Paul Janet, Sahasrajit Ramesh, Chenru Duan, and Heather J Kulik. Accurate multiobjective design in a space of millions of transition metal complexes with neural-network-driven efficient global optimization. *ACS central science*, 6(4):513–524, 2020.

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

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Hierarchical generation of molecular graphs using structural motifs. In *International conference on machine learning*, pp. 4839–4848. PMLR, 2020.

Junsu Kim, Sungsoo Ahn, Hankook Lee, and Jinwoo Shin. Self-improved retrosynthetic planning. In *International Conference on Machine Learning*, pp. 5486–5495. PMLR, 2021.

Akihiro Kishimoto, Beat Buesser, Bei Chen, and Adi Botea. Depth-first proof-number search with heuristic edge cost and application to chemical synthesis planning. *Advances in Neural Information Processing Systems*, 32, 2019.

Pierre Koch, Matthias Gehringer, and Stefan A Laufer. Inhibitors of c-jun n-terminal kinases: an update. *Journal of medicinal chemistry*, 58(1):72–95, 2015.

Brent A Koscher, Richard B Canty, Matthew A McDonald, Kevin P Greenman, Charles J McGill, Camille L Bilodeau, Wengong Jin, Haoyang Wu, Florence H Vermeire, Brooke Jin, et al. Autonomous, multiproperty-driven molecular discovery: From predictions to measurements and back. *Science*, 382(6677):eadi1407, 2023.

Itai Levin, Michael E Fortunato, Kian L Tan, and Connor W Coley. Computer-aided evaluation and exploration of chemical spaces constrained by reaction pathways. *AIChE journal*, 69(12):e18234, 2023.

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

Guoqing Liu, Di Xue, Shufang Xie, Yingce Xia, Austin Tripp, Krzysztof Maziarz, Marwin Segler, Tao Qin, Zongzhang Zhang, and Tie-Yan Liu. Retrosynthetic planning with dual value networks. In *International Conference on Machine Learning*, pp. 22266–22276. PMLR, 2023.

Qi Liu, Miltiadis Allamanis, Marc Brockschmidt, and Alexander Gaunt. Constrained graph variational autoencoders for molecule design. *Advances in neural information processing systems*, 31, 2018.

Shitong Luo, Wenhao Gao, Zuofan Wu, Jian Peng, Connor W Coley, and Jianzhu Ma. Projecting molecules into synthesizable chemical spaces. *arXiv preprint arXiv:2406.04628*, 2024.

Jiankun Lyu, Sheng Wang, Trent E Balius, Isha Singh, Anat Levit, Yurii S Moroz, Matthew J O'Meara, Tao Che, Enkhjargal Algaa, Kateryna Tolmachova, et al. Ultra-large library docking for discovering new chemotypes. *Nature*, 566(7743):224–229, 2019.

Tengfei Ma, Jie Chen, and Cao Xiao. Constrained generation of semantically valid graphs via regularizing variational autoencoders. *Advances in Neural Information Processing Systems*, 31, 2018.

Bertha K Madras. History of the discovery of the antipsychotic dopamine d2 receptor: a basis for the dopamine hypothesis of schizophrenia. *Journal of the History of the Neurosciences*, 22(1):62–78, 2013.

William McCorkindale. *Accelerating the Design-Make-Test cycle of Drug Discovery with Machine Learning*. PhD thesis, 2023.

Paul Merrell. Example-based procedural modeling using graph grammars. *ACM Transactions on Graphics (TOG)*, 42(4):1–16, 2023.

Paul Merrell and Dinesh Manocha. Model synthesis: A general procedural modeling algorithm. *IEEE transactions on visualization and computer graphics*, 17(6):715–728, 2010.

Paul Merrell, Eric Schkufza, Zeyang Li, Maneesh Agrawala, and Vladlen Koltun. Interactive furniture layout using interior design guidelines. In *ACM SIGGRAPH 2011 Papers*, SIGGRAPH '11, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450309431. doi: 10.1145/1964921.1964982. URL https://doi.org/10.1145/1964921.1964982.

Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. Procedural modeling of buildings. In *ACM SIGGRAPH 2006 Papers*, pp. 614–623. 2006.

Nadia Polikarpova, Ivan Kuraj, and Armando Solar-Lezama. Program synthesis from polymorphic refinement types. *ACM SIGPLAN Notices*, 51(6):522–538, 2016.

Bidisha Samanta, Abir De, Gourhari Jana, Vicenç Gómez, Pratim Chattaraj, Niloy Ganguly, and Manuel Gomez-Rodriguez. Nevae: A deep generative model for molecular graphs. *Journal of machine learning research*, 21(114):1–33, 2020.

Benjamin Sanchez-Lengeling and Alán Aspuru-Guzik. Inverse molecular design using machine learning: Generative models for matter engineering. *Science*, 361(6400):360–365, 2018. doi: 10.1126/science.aat2663. URL https://www.science.org/doi/abs/10.1126/science.aat2663.

Eric Schkufza, Rahul Sharma, and Alex Aiken. Stochastic superoptimization. *SIGPLAN Not.*, 48(4):305–316, mar 2013. ISSN 0362-1340. doi: 10.1145/2499368.2451150. URL https://doi.org/10.1145/2499368.2451150.

Marwin HS Segler and Mark P Waller. Neural-symbolic machine learning for retrosynthesis and reaction prediction. *Chemistry–A European Journal*, 23(25):5966–5971, 2017.

Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*, 2020.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I 27*, pp. 412–422. Springer, 2018.

Warren D Smith. Computational complexity of synthetic chemistry–basic facts. Technical report, Citeseer, 1997.

Armando Solar-Lezama, Rodric Rabbah, Rastislav Bodík, and Kemal Ebcioğlu. Programming by sketching for bit-streaming programs. In *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, pp. 281–294, 2005.

Jiangming Sun, Nina Jeliazkova, Vladimir Chupakhin, Jose-Felipe Golib-Dzib, Ola Engkvist, Lars Carlsson, Jörg Wegner, Hugo Ceulemans, Ivan Georgiev, Vedrin Jeliazkov, et al. Excape-db: an integrated large scale dataset facilitating big data analysis in chemogenomics. *Journal of cheminformatics*, 9:1–9, 2017.

Michael Sun, Minghao Guo, Weize Yuan, Veronika Thost, Crystal Elaine Owens, Aristotle Franklin Grosz, Sharvaa Selvan, Katelyn Zhou, Hassan Mohiuddin, Benjamin J Pedretti, et al. Representing molecules as random walks over interpretable grammars. *arXiv preprint arXiv:2403.08147*, 2024.

Kyle Swanson, Gary Liu, Denise B Catacutan, Autumn Arnold, James Zou, and Jonathan M Stokes. Generative ai for designing and validating easily synthesizable and structurally novel antibiotics. *Nature Machine Intelligence*, 6(3):338–353, 2024.

Jerry O Talton, Yu Lou, Steve Lesser, Jared Duke, Radomír Mech, and Vladlen Koltun. Metropolis procedural modeling. *ACM Trans. Graph.*, 30(2):11–1, 2011.

Paula Torren-Peraire, Alan Kai Hassen, Samuel Genheden, Jonas Verhoeven, Djork-Arné Clevert, Mike Preuss, and Igor V Tetko. Models matter: the impact of single-step retrosynthesis on synthesis planning. *Digital Discovery*, 3(3):558–572, 2024.

Hongyu Tu, Shantam Shorewala, Tengfei Ma, and Veronika Thost. Retrosynthesis prediction revisited. In *NeurIPS 2022 AI for Science: Progress and Promises*, 2022.

H Maarten Vinkers, Marc R de Jonge, Frederik FD Daeyaert, Jan Heeres, Lucien MH Koymans, Joop H van Lenthe, Paul J Lewi, Henk Timmerman, Koen Van Aken, and Paul AJ Janssen. Synopsis: synthesize and optimize system in silico. *Journal of medicinal chemistry*, 46(13): 2765–2773, 2003.

Andrea Volkamer, Sereina Riniker, Eva Nittinger, Jessica Lanini, Francesca Grisoni, Emma Evertsson, Raquel Rodríguez-Pérez, and Nadine Schneider. Machine learning for small molecule drug discovery in academia and industry. *Artificial Intelligence in the Life Sciences*, 3:100056, 2023.

Zhenpeng Yao, Benjamín Sánchez-Lengeling, N Scott Bobbitt, Benjamin J Bucior, Sai Govind Hari Kumar, Sean P Collins, Thomas Burns, Tom K Woo, Omar K Farha, Randall Q Snurr, et al. Inverse design of nanoporous crystalline reticular materials with deep generative models. *Nature Machine Intelligence*, 3(1):76–86, 2021.

Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. *Advances in neural information processing systems*, 31, 2018.

Lap-Fai Yu, Sai-Kit Yeung, Chi-Keung Tang, Demetri Terzopoulos, Tony F. Chan, and Stanley J. Osher. Make it home: automatic optimization of furniture arrangement. *ACM Trans. Graph.*, 30 (4), jul 2011. ISSN 0730-0301. doi: 10.1145/2010324.1964981. URL https://doi.org/10.1145/2010324.1964981.

Chun-Hui Zhang, Elizabeth A Stone, Maya Deshmukh, Joseph A Ippolito, Mohammad M Ghahremanpour, Julian Tirado-Rives, Krasimir A Spasov, Shuo Zhang, Yuka Takeo, Shalley N Kudalkar, et al. Potent noncovalent inhibitors of the main protease of sars-cov-2 from molecular sculpting of the drug perampanel guided by free energy perturbation calculations. *ACS central science*, 7(3): 467–475, 2021.

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

Julie B Zimmerman, Paul T Anastas, Hanno C Erythropel, and Walter Leitner. Designing for a green chemistry future. *Science*, 367(6476):397–400, 2020.

A SYNTACTIC TEMPLATES

Syntactic templates form the essential ingredients for syntax-guided synthesis, as they significantly reduce the number of possible programs. In practice, syntactical templates are provided by users who operate with real-world constraints or experts who can help narrow the search space to *desirable* templates. The exact criteria for selecting templates are problem-dependent. To prove our concept in a more generalizable workflow, we bootstrap our set of syntactic templates $\hat{\mathcal{T}}$ in a data-driven way by obtaining the syntactic templates present in the training set. We then simulate real-world constraints by setting $\hat{\mathcal{T}}_k \leftarrow \{T \in \hat{\mathcal{T}} \mid T \text{ has at most } k \text{ internal nodes}\}$ and optimize within the induced design space $\partial\hat{\mathcal{P}}_k$ (Section 3.3.2). We tabulate summary statistics in Table 5 for the number of unique syntactic templates and the number of topological orders. We see that the empirical distribution is biased towards *simpler* syntactic templates, which reflects real-world constraints and is a key enabler of our amortized approach. We train the parameters $(\Theta, \Phi, \Omega)_k$ of our policies $(\tau, \pi_{\mathcal{R}}, \pi_{\mathcal{B}})$, respectively, for $k = 3, 4, 5, 6$ on our (pre)training dataset $\mathcal{D}$. For samples in $\mathcal{D}$ with more than $k > 6$ reactions, we snap it to the closest $T \in \hat{\mathcal{T}}_k$ according to the tree edit distance. We find $k = 3$ using full topological decoding (illustration in Figure 3) is best for Synthesizable Analog Generation and $k = 4$ with random sampling of the decoding beams is a good compromise between accuracy and efficiency for Synthesizable Molecule Design. We also note that the number of unique templates grows sub-exponentially, and in fact the number of templates for a fixed number of reactions starts diminishing for $k > 6$. To make sure this does not cause issues, we ensured there is still sufficient coverage to formulate a Markov Chain on $\hat{\mathcal{T}}_k$, which is crucial for our bilevel algorithms. For example, Figure 4 visualizes the empirical proposal distribution $J(T_1, T_2), \forall T_1, T_2 \in \hat{\mathcal{T}}_4 \times \hat{\mathcal{T}}_4$. Importantly, key hyperparameters like β and n_{edits} enable control over exploration vs exploitation.

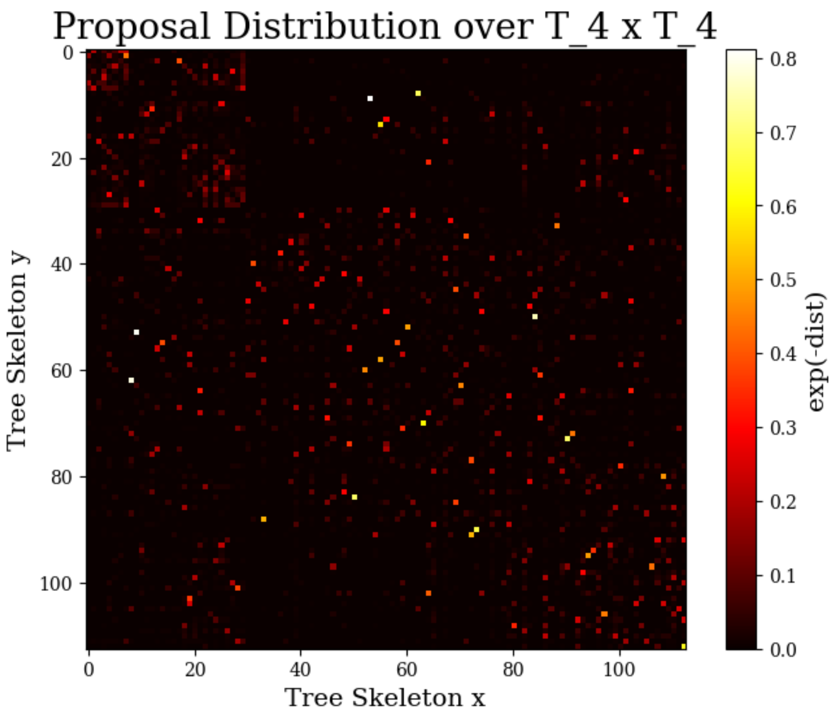


Figure 4: We adopt the tree edit distance as the dist function. We see that $\hat{\mathcal{T}}_4$ has sufficient transition coverage for bootstrapping our space of syntactic templates.

# Rxns	$\lvert\hat{\mathcal{T}}_{k\setminus k-1}\rvert$ ($\lvert\mathcal{T}_{k\setminus k-1}\rvert$)	# Topo. Orders (Max, Mean, Std)$_{\hat{\mathcal{T}}_{k\setminus k-1}}$
1	2 (2)	2, 1.5, 0.5
2	6 (6)	8, 4.17, 2.79
3	22 (22)	80, 19.59, 20.55
4	83 (90)	896, 152.02, 215.53
5	209 (394)	19200, 2506.25, 3705.77

(a)

# Rxns	$\lvert\hat{\mathcal{T}}_{k\setminus k-1}\rvert$
6	298
7	243
8	112
9	63
10	42
11	22
12	11
13	4
14	2

(b)

# Rxns	# Topo. Masks ()$_{\hat{\mathcal{T}}_{k\setminus k-1}}$	# Topo. Masks ()$_{\mathcal{T}_{k\setminus k-1}}$
1	5, 4, 1	5, 4, 1
2	11, 7.67, 2.56	11, 7.67, 2.56
3	26, 14.36, 5.86	26, 14.36, 5.86
4	56, 27.99, 12.47	56, 26.73, 12.78
5	131, 65.07, 26.36	131, 49.74, 27.09
6	287, 165.12, 61.43	287, 92.67, 56.29

(c)

Figure 5: (a) Summary statistics of the number of syntactic templates (both empirical and theoretically possible) and possible topological decoding node orders for $k = 1, 2, \ldots, 5$; (b) Summary statistics for only the number of syntactic templates since enumerating all topological sorts becomes intractable; (c) Summary statistics for the number of topological masks (subset of nodes closed under parent(.))

B Extrapolation to Unseen Templates

B.1 Data Preparation

In this section, we investigate whether SynthesisNet can extrapolate to unseen templates, and effectively incorporate them for synthesizable analog generation and molecular design. We setup an ablation study as follows:

1. Reverse sort the templates by frequency using our dataset $\mathcal{D}_0$.
2. Collect every *fourth* template into a hold-out set $\mathcal{T}_{\text{test}}$ for $k = 4$.
3. Construct $\mathcal{D}_0' := \{(P, B) \in \mathcal{D}_0 \mid T_{P,B} \in \mathcal{T}_4 \setminus \mathcal{T}_{\text{test}}\}$ where $T_{P,B} \in \mathcal{T}$ is the syntactical template of (P, B).
4. Run Algo. 1 on $\mathcal{D}_0'$ to obtain $\mathcal{D}'$.
5. Train ablation policy networks $\{\pi_{\mathcal{B}}', \pi_{\mathcal{R}}'\}$ using $\mathcal{D}'$.
6. Evaluate task performances with same τ as before.

We select hold-out templates in a frequency-stratified manner, ensuring the frequency distribution of $\hat{\mathcal{T}}_{\text{test}}$ is similar to that of $\hat{\mathcal{T}}$. Since smaller templates appear more frequently, the sizes of of templates are also indirectly stratified this way. Since we choose the least frequent from each consecutive group of 4 (Step 2), we note on average templates in $\hat{\mathcal{T}}_{\text{test}}$ tend to be slightly larger than $\hat{\mathcal{T}}$, so results for test templates may be lower in Table 5.

B.2 Results on Synthesizable Analog Generation

We evaluate the synthesizable analog task performances using ablation networks. We want to test whether the ablation networks integrate effectively with templates outside its structural support. Thus, we use the same τ as before for the τ experiments in Table 5, allowing access to the full template set but forcing $\{\pi_{\mathcal{B}}', \pi_{\mathcal{R}}'\}$ to extrapolate when performing inference over $\hat{\mathcal{T}}_{\text{test}}$.

Table 5: Apart from swapping out the policy networks, we use the same experimental setup as Table 1. For fair comparison, we also retrained Ours with $k = 4$ templates (whereas Table 1 used $k = 3$).

			Avg. Sim. ↑			SA ↓			Diversity ↑	
Dataset	Method	RR ↑	Top-1	Top-3	Top-5	Top-1	Top-3	Top-5	Top-3	Top-5
Test Set	Ours:EP (τ)	52%	0.815	0.616	0.548	3.140	2.964	2.892	0.585	0.646
	Ours (τ)	56%	0.827	0.633	0.555	3.100	3.019	2.918	0.543	0.628
	Ours:EP $\mathcal{T}_{\text{test}}$ (τ)	20%	0.636	0.539	0.473	2.844	3.023	2.987	0.564	0.675
	Ours $\mathcal{T}_{\text{test}}$ (τ)	20%	0.626	0.552	0.493	3.360	3.135	3.070	0.542	0.634
ChEMBL	Ours (τ)	7.6%	0.531	0.443	0.396	2.544	2.510	2.460	0.675	0.727
	Ours (MCMC)	9.2%	0.532	0.486	0.432	2.364	2.310	2.263	0.765	0.759
	Ours:EP (MCMC)	8.5%	0.519	0.421	0.367	2.644	2.420	2.382	0.618	0.640

The performance takes only a minor dip on the Test Set, compensating for slightly lower Avg. Sim. with higher Diversity and comparable SA. We further zoom in on the subset of $\mathcal{D}_0$ with structure among the held-out template classes. We emphasize this is very difficult for $\{\pi'_{\mathcal{B}}, \pi'_{\mathcal{R}}\}$ to do, which has not seen any examples from those structural classes. It actually appears Ours:EP has the slight edge over Ours on the held-out template set, with slightly better SA and greater diversity. We attribute this to a regularization effect induced by removing (slightly, due to Step 2.) more complex program structures. There are still enough complex templates left that this does not harm performance, highlighting the robustness of our model in this setting. We believe the fact the ablation model can maintain comparable performance implies the following:

- **Structural Extrapolation**: It is capable of inference of programs outside the structural support of its training distribution in this case.
- **Template Set Robustness**: Our model is not very sensitive to the default size of the template set, since using only 75% of it already brings it close to diminishing returns.

The dip in performance is more noticeable for the predominantly unsynthesizable dataset ChEMBL, with lower metrics across the board. We suspect the reason is due to ChEMBL containing more complex molecules that require longer synthetic routes. This is also apparent from the lower analog diversity, suggesting training on more template variety helps.

We believe the difficulty of the task (ratio of synthesizable vs unsynthesizable molecules) can inform whether the method is sensitive to the template set it sees during training. Similar ablation studies to this one can highlight when additional resources should be allocated to expanding the training set and when it is sufficient to simply incorporate more templates at test time.

B.3 Results on Synthesizable Molecular Design

Figure 6: We select the first Oracle from each Table in App. H to compare Ours with Ours (EP). Aside from the ablation networks, we use the same experimental settings as Table 2.

(a)

		GSK3β									
				Top 1			Top 10			Top 100	
	category	Oracle Calls	Score	SA	AUC	Score	SA	AUC	Score	SA	AUC
Ours (EP)	synthesis	6921	**0.99**	**1.975**	0.872	0.965	2.321	0.818	0.94	**2.237**	0.744
Ours	synthesis	4886	0.98	2.045	**0.891**	**0.967**	**2.302**	**0.848**	**0.944**	2.27	**0.778**

		Median 1									
				Top 1			Top 10			Top 100	
	category	Oracle Calls	Score	SA	AUC	Score	SA	AUC	Score	SA	AUC
Ours (EP)	synthesis	9050	**0.4**	4.12	0.358	**0.344**	4.434	**0.305**	**0.301**	4.394	**0.257**
Ours	synthesis	8303	**0.4**	**3.353**	**0.371**	0.342	**4.161**	**0.305**	0.298	**4.256**	0.252

(b)

		Osimertinib MPO									
				Top 1			Top 10			Top 100	
	category	Oracle Calls	Score	SA	AUC	Score	SA	AUC	Score	SA	AUC
Ours (EP)	synthesis	10000	0.852	2.322	**0.831**	**0.849**	2.475	**0.823**	**0.839**	2.484	**0.802**
Ours	synthesis	10000	**0.859**	**2.263**	0.826	0.847	**2.21**	0.81	0.832	**2.249**	0.769

		Perindopril MPO									
				Top 1			Top 10			Top 100	
	category	Oracle Calls	Score	SA	AUC	Score	SA	AUC	Score	SA	AUC
Ours (EP)	synthesis	10000	**0.626**	**3.382**	**0.562**	**0.598**	**3.375**	**0.54**	0.56	3.349	**0.509**
Ours	synthesis	10000	0.622	3.338	0.547	0.591	3.378	0.524	**0.558**	**3.137**	0.485

We also evaluate the synthesizable molecular design performances using ablation networks. We want to evaluate whether the ablation models can guide the optimization trajectory as a surrogate generator of synthesizable analogs. We see comparable performances across all four Oracles in Table 14, and for each Ours (EP) having the edge on some metrics while Ours having the edge on other metrics. This suggests both are capable enough to serve as the inner subroutine of our bilevel genetic framework, although the models may have different biases on the kind of analogs it generates which affects the optimization trajectories. Since Ours (EP) may generate less structurally diverse analogs, it can converge in fewer Oracle calls, resulting in less Oracle calls and slightly higher AUCs. Meanwhile, Ours produce more diverse analogs, which enables the acquisition of higher confidence analogs. We see Ours to have the edge on SA across Median 1 (Top 1) and the MPO (Top 100) Oracles. This may be because higher confidence regions tend to be where the simpler molecules are, resulting in simpler analogs hence lower SA. Overall, the results show the robustness of our model.

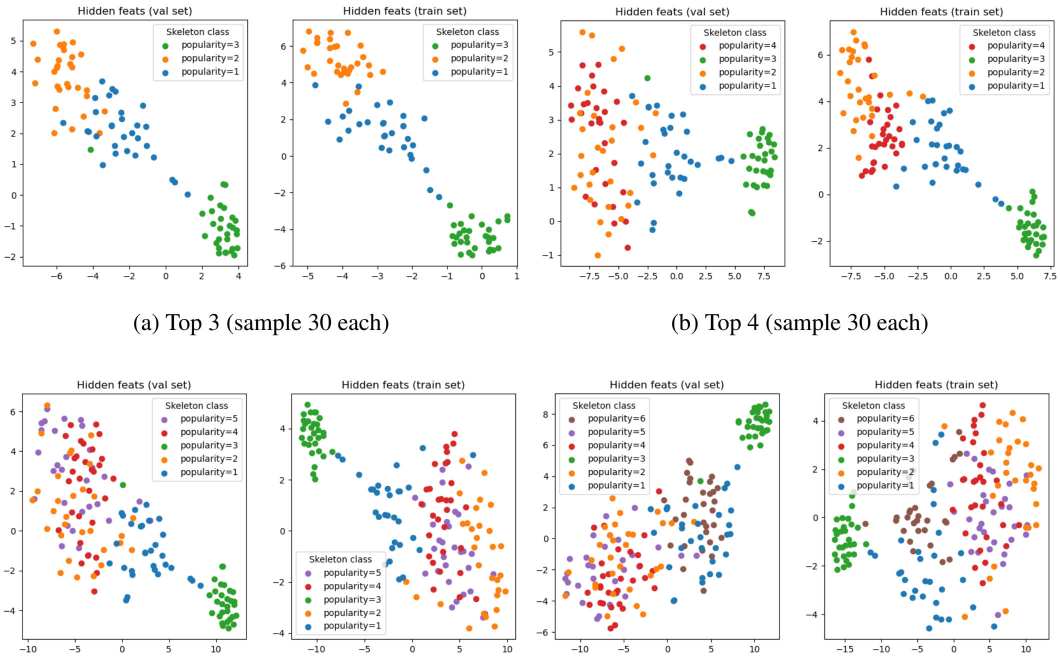
C Syntax Tree Recognition

In this section, we answer key questions like: (1) How does the relationship change with the addition of $\mathcal{T}$? (2) How strong is the correlation between $\mathcal{X}$ and $\mathcal{T}$? (3) How justified are the most confident predictions made by τ? We investigate the relationship between $\mathcal{M}$ and $\mathcal{T}$. We seek to understand the extent to which the true mapping $\mathcal{M} \to \mathcal{T}$ is well-defined. The first part is quantitative analysis, and the second part is a qualitative study.

C.1 t-SNE and MDS Plots

We use the t-distributed stochastic neighbor embedding (t-SNE) on the final layer hidden representations of our MLP τ to visualize how our recognition model discriminates between molecules of different syntax tree classes. From Figure 7, we see the MLP is able to discriminate amongst the top 3 or 4 most popular skeleton classes, visually partitioning the representation space. However, beyond that the representations on the validation set begin to coalesce, i.e., the model begins overfitting.

Figure 7: t-SNE on molecules in top (3,4,5,6) skeleton classes



(a) Top 3 (sample 30 each)

(b) Top 4 (sample 30 each)

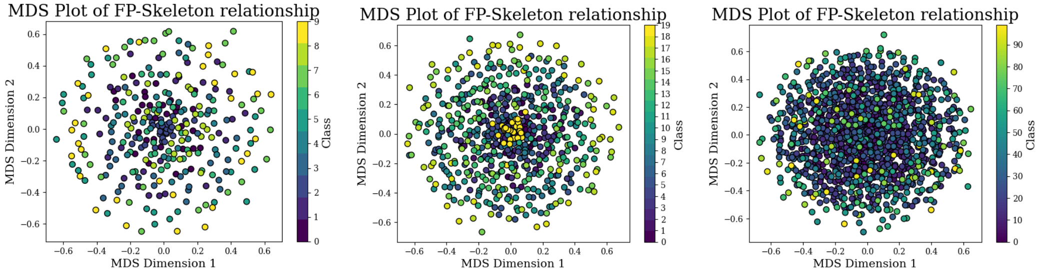
(c) Top 5 (sample 30 each)

(d) Top 6 (sample 30 each)

Since gradient descent is stochastic, we also use multi-dimensional scaling (MDS) using the Morgan Fingerprint Manhattan distance on a subset of our dataset to visualize the relative positioning between molecules of different syntax tree classes (sorted based on popularity). From the plots in Figure 9, we observe some interesting trends:

- **Similarly positioned points tend to have similar colors.**

Figure 8: MDS on molecules in top (10,20,100) skeleton classes



(a) Top 10 (sample 30 each)

(b) Top 20 (sample 30 each)

(c) Top 100 (sample 30 each)

- The darker end of the spectrum corresponding to the most popular classes generally cluster together in the middle.
- The classes do not form disjoint partitions in space. As the ranked popularity increases, the points tend to disperse outwards. There are exception classes, e.g., the yellow set of points in Figure 8b that cluster in the center.

Based on these findings, it's reasonable to conclude a recognition classifier by itself is overly naive. However, the useful inductive bias that similar molecules are more likely to share the same syntactic template indicates the **localness** property still holds. Our method is designed with this property in mind: we encourage iterative refinement of the syntactic template when doing analog generation.

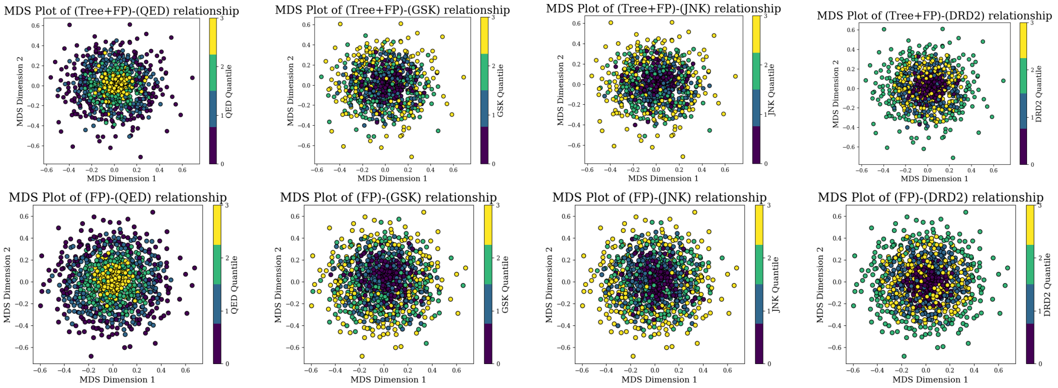


Figure 9: We visualize the structure-property relationship as a scatterplot of 2D structures vs property values. (Top) Structure is $\mathcal{X} \times \mathcal{T}$. We use MDS with the dissimilarity $d_{\mathcal{X}\times\mathcal{T}}((\boldsymbol{x}_1, T_1), (\boldsymbol{x}_2, T_2)) = ||\boldsymbol{x}_1 - \boldsymbol{x}_2||_1 + \text{Tree-Edit-Distance}(T_1, T_2)$. (Bottom) Structure is only $\mathcal{X}$.

We also use MDS to investigate the structure-property relationship to understand the joint effect $\mathcal{T}$ and $\mathcal{X}$ has on different properties of interest. As shown in Figure 9, we see overall, the functional landscape varies significantly from property to property, but the general trend is that decoupling $\mathcal{T}$ from $\mathcal{X}$ does not change the structure-property relationship much. Whereas analog generation requires a more granular understanding of the synergy between $\mathcal{X}$ and $\mathcal{T}$, molecular optimization does not. Instead, the evolutionary strategy should be kept fairly consistent between the original design space ($\mathcal{X}$) and ($\mathcal{X} \times \mathcal{T}$). However, the top row exhibits lower entropy, with the empirical distribution looking "less Gaussian". To capture this nuance, the evolutionary algorithm should combine both global and local optimization steps. We meet this observation with a bilevel optimization strategy that combines semantic crossover with syntactic mutation.

C.2 Expert Case Study

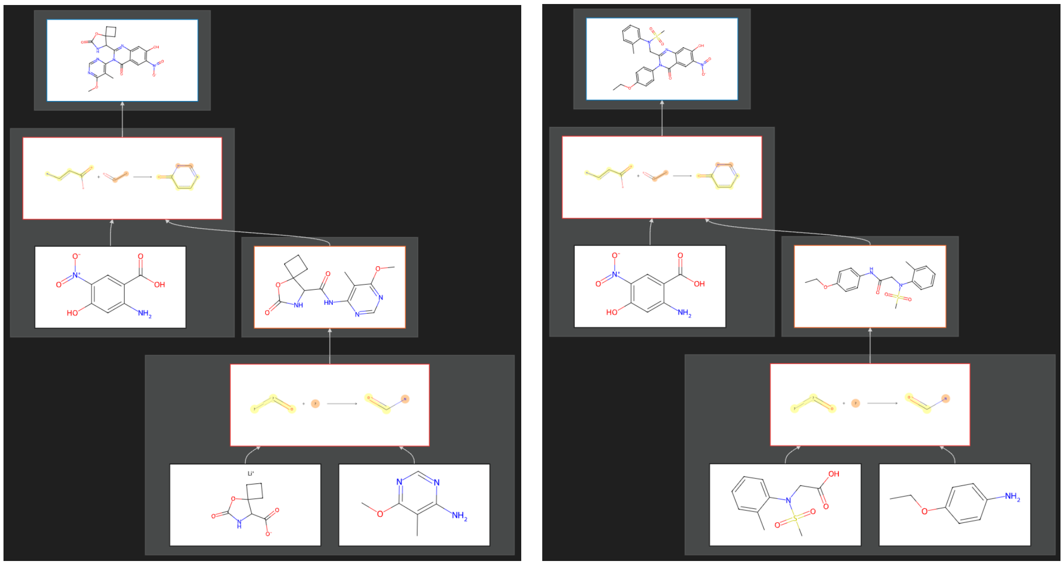
In this section, we enter the perspective of the recognition model learning the mapping from molecules to syntax tree skeletons. The core difference between this exercise and a common organic chemistry exam question (Flynn, 2014) is the option to abstract out the specific chemistry. Since the syntax only determines the skeletal nature of the molecule, the specific low-level dynamics don't matter. As long as the model can pick up on skeletal similarities between molecules, it will be confident in its prediction. We did the following exercise to understand if cases where the recognition model is most confident on unseen molecules can be attributed to training examples. We took the following steps:

For each true skeleton class T

1. Inference the recognition model on 10 random validation set molecules belonging to T.
2. Pick the top 2 molecules the model was most confident belongs to T.
3. For each molecule M.
    1. Find the 2 nearest neighbors to M belonging to T in the training set.

Shown in Figures Figure 10 and Figure 11 is the output of these steps for a common skeleton class which requires two reaction steps.

Figure 10: COc1ncnc(N2C(=O)c3cc([N+](=O)[O-])c(O)cc3N=C2C2NC(=O)OC23CCC3)c1C which recognition model predicts is in its true class with 87.5% probability



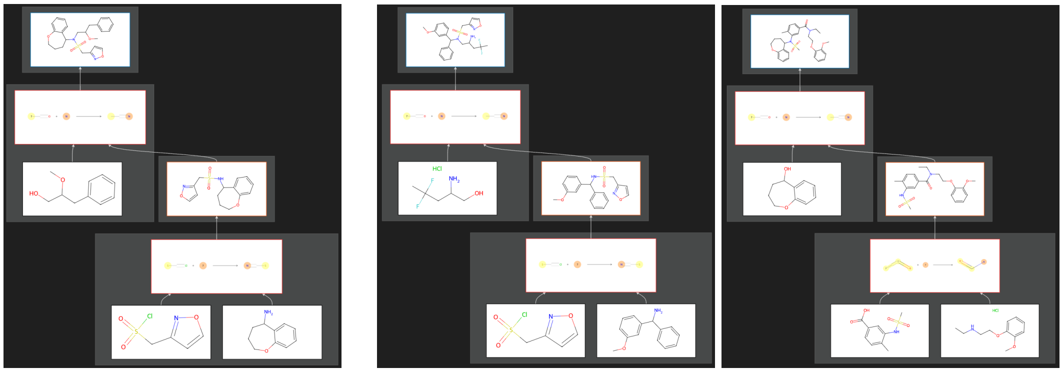
(a) Query molecule

(b) Nearest neighbor in training set

In Figure 10, we see that the query molecule's nearest neighbor is an output from the *same program* but different building blocks. Both feature the same core fused ring system involving a nitrogen. Given that the model has seen Figure 10b (and other similar instances), it should associate this core feature with a ring formation reaction step. Taking a step deeper, the respective precursors also share the commonality of having an amide linkage in the middle. Amides are key structural elements that the recognition model can identify. Both precursors underwent the same amide linkage formation step, despite the building blocks being different. Thus, the model's high confidence on the query molecule can be attributed directly to Figure 10b.

In Figure 11, there is more "depth" to the matter. We see a skeletal similarity across all three molecules: a nitrogen in the center with three substituents. Although it's noteworthy that the nitrogen participates in a sulfonamide group in all three cases, using this fact to inform the syntax tree would be a mistake. This is because in Figure 11a and Figure 11b, the sulfonamide group is the result of an explicit sulfonamide formation reaction, where a sulfonyl chloride reacts with an amine. However, in Figure 11c the sulfonamide group is already present in a building block. Thus, we see where the recognition model taking as input the circular fingerprint of this molecule could overfit. Nonetheless, the nitrogen with three substituents necessitates at least one reaction is required. The necessity for a

Figure 11: COC(Cc1ccccc1)CN(C1CCCOc2ccccc21)S(=O)(=O)Cc1ccon1 which recognition model predicts is in its true class with 86.2% probability



(a) Query molecule (b) Nearest neighbor in training set (c) 2nd Nearest neighbor

second reaction can be attributed to the ether linkage present in both Figure 11a and Figure 11c. The recognizer would be able to justify an additional reaction after it has seen the bicyclic ring structure joined with the sulfonamide group sufficiently many times before. In summary, the model will often be presented multiple complex motifs, but only a subset of them may be responsible for reaction steps. The exact number of reactions needed can only be determined via actually doing the search, but high-level indicators (such as the nitrogen with three substituents) allow the recognition model to abstract out the semantic details and construe a “first guess" of what the syntax tree is.

D Expanded Related Works

D.1 Background on Program Synthesis

Program synthesis is the problem of synthesizing a function f from a set of primitives and operators to meet some correctness specification. For example, if we want to synthesize a program to find the max of two numbers, the correctness specification $\phi_{\text{max}} := f(x,y) \geq x \wedge f(x,y) \geq y \wedge (f(x,y) = x \vee f(x,y) = y)$. As our approach is inspired from ideas in program synthesis, we briefly cover some basic background. A program synthesis problem entails three things:

1. Background theory T which is the vocabulary for constructing formulas, a typical example being linear integer arithmetic: which has boolean and integer variables, boolean and integer constants, connectives ($\wedge$, $\vee$, $\neg$, $\rightarrow$), operators ($+$), comparisons ($\leq$), conditional (If-Then-Else)
2. Correctness specification: a logical formula involving the output of f and T
3. Set of expressions L that f can take on described by a context-free grammar G_L.

Program synthesis is often formulated as deducing a constructive proof to the statement: for all inputs, there exists an output such that ϕ holds. The constructive proof itself is then the program. At the low-level, program synthesis methods repeatedly calls a SAT solver with the logical formula $\neg\phi$. If UNSAT is returned, this means f is valid. Syntax-guided synthesis (Alur et al., 2013; Schkufza et al., 2013) (SyGuS) is a framework for meeting the correctness specification with a syntactic template. Syntactic templates explicitly constrains G_L, significantly reducing the number of implementations f can take on. Sketching is an example application where programmers can sketch the skeletal outline of a program for synthesizers to fill in the rest (Solar-Lezama et al., 2005). More directly related to our problem's formulation is inductive synthesis, which seeks to generate f to match input/output examples. The problem of synthesis planning for a molecule M is a special case of the programming-by-example paradigm, where we seek to synthesize a program consistent with a single input/output pair: $(\{B\}, M)$. Inductive synthesis search algorithms have been developed to search through the combinatorial space of derivations of G_L. In particular, stochastic inductive synthesis use techniques like MCMC to tackle complex synthesis problems where enumerative methods do not scale to. MCMC has been used to optimize for the opcodes in a program (Schkufza et al., 2013) or for the abstract syntax tree directly (Alur et al., 2013). In our case, the space of possible program semantics is so large that we decouple the syntax from the semantics, performing stochastic synthesis over only the syntax trees. We also borrow ideas from functional program synthesis, where top-down strategies are preferred over bottom-up ones to better leverage the connection between a high-level specification and a concrete implementation (Polikarpova et al., 2016). Similar to how top-down synthesis enables aggressive pruning of the search space via type checking, retrosynthesis algorithms leverages the target molecule M to prune the search space via template compatability checks.

D.2 Execution-guided Program Synthesis

We would like to note the distinction between our program synthesis formulation and other formulations. Retrosynthesis is essentially already guided by the execution state at every step. Each expansion in the search tree executes a deterministic reaction template to obtain the new intermediate molecule. Planners based on single-step models (Chen et al., 2020), for example, assume the Markov Property by training models to directly predict a template given *only* the intermediate (Torren-Peraire et al., 2024; Tu et al., 2022). In program synthesis, meanwhile, the state space is a set of partial programs with actions corresponding to growing the program. The execution of the program (or verification against the specification) does not happen until a complete program is obtained. In recent years, neural program synthesis methods found using auxiliary information in the form of the *execution state* of a program can help indirectly inform the search (Bunel et al., 2018; Chen et al., 2018; Ellis et al., 2019) since it gives a sense on what the program can compute so far. This insight does not apply to retrosynthesis, since retrosynthesis already executes on the fly. It also does not apply for the methods introduced in Section 2.2 that construct a synthetic tree in a bottom-up manner, for the same reason (the only difference is they use forward reaction templates, with a much smaller set of robust reaction templates) to obtain the execution state each step. However, as described in Section 3.3.1, our approach combines the computational advantages of restricting to a small set of forward reaction templates with the inductive bias of retrosynthetic analysis. Our policy is to predict *forward* reaction templates in a *top-down* manner. This formulation is common in top-down program synthesis, where an action corresponds to selecting a hole in the program. Similarly, our execution of the program does not happen until the tree is filled in. However, we leverage the insight that the execution state helps in an innovative way, as discussed in F.3.

D.3 Inspirations from Retrosynthesis and Alternate Formulations

We begin by elaborating the distinctions between retrosynthesis methods and methods for synthesizable molecular design. Then, we identify a few recent works from retrosynthesis that can inspire cross-pollination of ideas. Finally, we end with alternate formulations of the problem that are also valuable to consider for future cross-examination.

Intended Use Case

Retrosynthesis aims to find a synthetic route for a given target molecule, without reference to how the target molecule is obtained or further optimizations on the target molecule. The target molecule is a compound that may serve any application or use case that we will not get into here, but importantly it *is* the problem to solve. We refer readers to Gao & Coley (2020) for further descriptions.

Synthesizable molecular design aims to be a standalone molecular optimization workflow that explicitly constrains the design space to be synthetically accessible building blocks and reactions Vinkers et al. (2003). This is often coupled with property oracles that evaluate the designs, which guides the optimization towards parts of the design space with higher fitness Gao et al. (2022).

MDP Formulation

Retrosynthesis can be formulated as a tree-shaped MDP, where each state is a molecule (initial state being the target, terminal states being building blocks) and each action is a reversed (“retro") reaction. The tree shape of the MDP is due to the fact the retro reaction (action) produces a set of reactants (states) Liu et al. (2023). Retrosynthetic planners often tackle the MDP by combining a single-step model (predicting retro reactions) and a multi-step planner (e.g. A* search Liu et al. (2018), MCTS Segler & Waller (2017), depth-first search Kishimoto et al. (2019)). A solution to the MDP is a *tree* of actions, i.e. a synthetic tree, where all sequences of actions in this tree lead to terminal states.

Synthesizable molecular design feature a broader set of methods but can be defined as a discrete optimization problem over synthetic trees directly: $\arg\max_x f(F(x))$ where x is a synthetic tree, F the root molecule of x and f is the fitness function. As the design space is intractably large, prior approaches discussed in 2.3 formulate the problem as a serial MDP, where each state is a synthetic tree and each action an edit operation (add, merge, etc.) to the synthetic tree. Though simple, we argue such a formulation is ill-advised, for reasons we discussed and demonstrated in the main text. We use an alternate formulation, inspired by program synthesis, that considers each state as a partially sketched program and each action as completing a hole of the program.

Learning Goal

Retrosynthesis methods aim to learn a policy $\pi(a|s)$, where s is a molecule and a a retro reaction, which can be template-based or template-free (we won't go into that here). Traditional works learn the policy from public datasets of synthetic routes (e.g. USPTO), but recent works have explored novel strategies for learning π by combining offline and online training. The offline training is usually done on a reaction dataset to initialize a policy network and/or reaction model. The online training iteratively adapts the policy network by acquiring more data using a planning algorithm, possibly guided by the current policy network. More specifically, Guo et al. (2024) uses MCTS to acquire data, inferring policy and value targets based on the node visit counts. Kim et al. (2021) uses a self-improvement strategy, reminiscent of AlphaGo Zero Silver et al. (2017), that trains independent reference forward and backward reaction models to control the quality and diversity of new reaction pathways acquired by the planner. Liu et al. (2023) follows a similar strategy, but decouples the synthesizability and cost of the value function. They also architect a two-branch policy network that uses a trainable single-step network to optimize the probabilities over reactions from the frozen reference network to better model real-world synthesis considerations. Like Guo et al. (2024), they use MCTS guided by the current policy network to acquire more data, creating a synergistic feedback loop that results in a holistic, trained policy network which can be plugged into multi-step planners.

Synthesizable molecular design methods, meanwhile, are relatively more spread out, with research going into problem formulation and algorithmic frameworks for tackling this more open-ended problem. In SynthesisNet, the policy learning is done entirely offline (as described in Sections 3.3.1 and 4.1.1) to amortize for the cost of searching during the actual online phases (MCMC and GA), but above techniques from retrosynthesis can also facilitate self-improving the surrogate network. One potential idea for generating more experiences is to take partial program examples from the existing data, use guided planning to complete those examples, then retrain the surrogate network with the augmented dataset. We leave the details for exciting future extensions of our work.

Alternate Formulations

Towards synthesizable molecular design, alternate formulations from recent developments can also be considered. Decision Transformer Chen et al. (2021) is a recent work that re-imagines offline RL as a conditional sequence modeling task. Notably, the model conditions on reward-to-go and the history for generating the next action. The Transformer architecture enables long-term modeling of the environment's dynamics, enabling credit assignment and relational modeling of its history. The offline setting tackled by Decision Transformer naturally aligns with our method, where we do (self-)supervised policy training from programs generated offline. Although the program structure carries important hierarchical information about the relations of building blocks and reactions to one another, it's worth considering whether the synthesis tree construction serialization protocol used by prior works Bradshaw et al. (2020); Gao et al. (2021); Luo et al. (2024); Gao et al. (2024) can be used to formulate a conditional sequence modeling problem. GFlowNet Bengio et al. (2021; 2023) is also a recent work that formulates a flow network over a tree-based MDP, where the incoming and outgoing flow are proportional the probability of subsequent actions and learned using flow matching objectives. The goal of this model is to learn amortized samplers for reward functions, producing both diverse and high-quality samples constructed in a step-by-step manner following an MDP environment. The learning occurs on offline trajectories with observed rewards. The formulation using reward functions can be a direction of future work for our framework, which currently only considers rewards in the online phases. However, if we had considered rewards to be provided upfront along with the data generation procedure, we can adopt GFlowNet to amortize the expensive work done by MCMC, and directly sample programs. We leave this study for future works. We believe cross-examining alternate formulations and recent methodologies to be essential for finding future inspirations for extending the innovation horizon of methods used to tackle synthesizable molecular design.

E Derivation of Grammar

We now define the grammar $G_{\mathcal{P}}$ describing the set of implementations our program can take on. A context-free grammar is a tuple $G_{\mathcal{P}} := (\mathcal{N}, \Sigma, \mathcal{P}, \mathcal{X})$ that contains a set $\mathcal{N}$ of non-terminal symbols, a set Σ of terminal symbols, a starting node $\mathcal{X}$, and a set of production rules which define how to expand non-terminal symbols. Recall we are given a set of reaction templates $\mathcal{R}$ and building blocks $\mathcal{B}$. Templates are either uni-molecular ($:= \mathcal{R}_1$) or bi-molecular ($:= \mathcal{R}_2$), such that $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$. In the original grammar, these take on the following:

1. **Starting symbol**: T
2. **Non-terminal symbols**: R_1, R_2, B
3. **Terminal symbols**:
   - $\{R \in \mathcal{R}_1\}$: Uni-molecular templates
   - $\{R \in \mathcal{R}_2\}$: Bi-molecular templates
   - $\{BB \in \mathcal{B}\}$: Building blocks
4. **Production rules**:
   1. $T \to R_1$
   2. $T \to R_2$
   3. $R_1 \to R(B)$ $(\forall R \in \mathcal{R}_1)$
   4. $R_1 \to R(R_1)$ $(\forall R \in \mathcal{R}_1)$
   5. $R_1 \to R(R_2)$ $(\forall R \in \mathcal{R}_1)$
   6. $\forall (X_1, X_2) \in \{``R_1", ``R_2", ``B"\} \times \{``R_1", ``R_2", ``B"\}$
      - $R_2 \to R(X_1, X_2)$ $(\forall R \in \mathcal{R}_2)$
   7. $B \to BB$ $(\forall BB \in \mathcal{B})$

Example expressions derived from this grammar are “R3(R3(B1,B2),R2(B3))" and “R4(R1(B2,B1))" for the programs in Figure 1.

Identifying a retrosynthetic pathway can be formulated as the problem of searching through the derivations of this grammar conditioned on a target molecule. This unconstrained approach is extremely costly, since the number of possible derivations can explode.

In our syntax-guided grammar, we are interested in a finite set of syntax trees. The syntax tree of a program depicts how the resulting expression is derived by the grammar. These are either provided by an expert who has to meet experimental constraints, or specified via heuristics (e.g., maximum of x reactions, limiting the tree depth to y). For example, the syntax-guided grammar for the set of trees with at most 2 reactions is specified as follows:

1. **Starting symbol**: T
2. **Non-terminal symbols**: R_1, R_2, B
3. **Terminal symbols**:
   - $\{R \in \mathcal{R}_1\}$: Uni-molecular templates
   - $\{R \in \mathcal{R}_2\}$: Bi-molecular templates
   - $\{BB \in \mathcal{B}\}$: Building blocks
4. **Production rules**:
   1. $T \to R_2(B, B)$
   2. $T \to R_1(B)$
   3. $T \to R_1(R_2(B, B))$
   4. $T \to R_1(R_1(B))$
   5. $T \to R_2(B, R_1(B))$
   6. $T \to R_2(B, R_2(B, B))$
   7. $T \to R_2(R_1(B), B)$
   8. $T \to R_2(R_2(B, B), B)$

9. $R_1 \to R \, (\forall R \in \mathcal{R}_1)$
10. $R_2 \to R \, (\forall R \in \mathcal{R}_2)$
11. $B \to BB \, (\forall BB \in \mathcal{B})$

This significantly reduces the number of possible derivations, but two challenges remain:

- How can when pick the initial production rule when the number of syntax trees grow large? *We use an iterative refinement strategy, governed by a Markov Chain Process over the space of syntax trees. The simulation is initialized at the structure predicted from our recognition model Appendix C.*
- How can we use the inductive bias of retrosynthetic analysis when applying rules 9, 10, 11? *We formulate a finite horizon MDP over the space of partial programs, where the actions are restricted to decoding only frontier nodes. This topological order to decoding is consistent with the top-down problem solving done in retrosynthetic analysis. Furthermore, our pretraining and decoding algorithm enumerates all sequences consistent with topological order.*

These two questions are addressed by the design choices in Section 3.3.

F Policy Network

F.1 Featurization

Our dataset $\mathcal{D}$ comprises partial programs $T \in \partial\mathcal{P}$ producing molecules M.Then, we compute node features $\boldsymbol{H}$ and labels $\boldsymbol{Y}$ as:

$$\boldsymbol{h}_n := [\mathrm{FP}_{2048}(M), \mathrm{BB}_{2048}(n), \mathrm{RXN}(n)], \quad \boldsymbol{y}_n := \begin{cases} \mathrm{RXN}(n), & \text{if } i \text{ is a reaction node,} \\ \mathrm{BB}_{256}(n), & \text{otherwise,} \end{cases}$$

where $\mathrm{FP}_d(\cdot)$ computes the d-bit radius 2 Morgan fingerprints, $\mathrm{BB}_d(n) = \mathrm{FP}_d(n_{\mathrm{SMILES}})$ if n is attributed with a building block from $\mathcal{B}$ or $\mathbf{0}_d$ otherwise and $\mathrm{RXN}(n) = \mathrm{one\_hot}_{91}(n_{\mathrm{RXN\_ID}})$ if n is attributed with a reaction from $\mathcal{R}$ or $\mathbf{0}_{91}$ otherwise.

If $\mathcal{N}(T)$ and $\mathcal{E}(T)$ denote the node and edge set of $T \in \partial\mathcal{P}$, then we define, for convenience:

$$\mathrm{RXN}(T) := \{r \in \mathcal{N}(T) \mid \exists c, p \in \mathcal{N}(T) \text{ s.t. } (r, c) \in \mathcal{E}(T) \cap (p, r) \in \mathcal{E}(T)\} \tag{1}$$

$$\mathrm{BB}(T) = \{b \in \mathcal{N}(T) \mid \nexists c \text{ s.t. } (b, c) \in \mathcal{E}(T)\}.\} \tag{2}$$

F.2 Loss Function

Let the superscript (i) indicate the i-th sample in the dataset. The loss function is:

$$\mathcal{L}_{\mathcal{D}}(\Phi) := \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \sum_{n \in \mathrm{RXN}(T^{(i)})} \mathrm{CE}(\pi_{\mathcal{R}}(T^{(i)}, \boldsymbol{H}^{(i)})_n, \boldsymbol{y}_n^{(i)}),$$

$$\mathcal{L}_{\mathcal{D}}(\Omega) := \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \sum_{n \in \mathrm{BB}(T^{(i)})} \mathrm{MSE}(\pi_{\mathcal{B}}(T^{(i)}, \boldsymbol{H}^{(i)})_n, \boldsymbol{y}_n^{(i)}).$$

CE and MSE denote the standard cross entropy loss and mean squared error loss, respectively. For our evaluation metric, we consider accuracy, where the output of $\pi_{\mathcal{B}}$ is interpreted as the nearest building block with respect to cosine distance.

F.3 Auxiliary Training Task

In Section 3.3.1, we defined the representation T to be the parse tree of a partial program. However, we omitted an extra step that was used to preprocess T for training. The motivation for this extra step is discussed deeply in D.2. We add an additional step when preprocessing $\mathcal{D}$: For each T in $\mathcal{D}$, for each node r corresponding to a reaction, we add a new node o_r corresponding to the intermediate

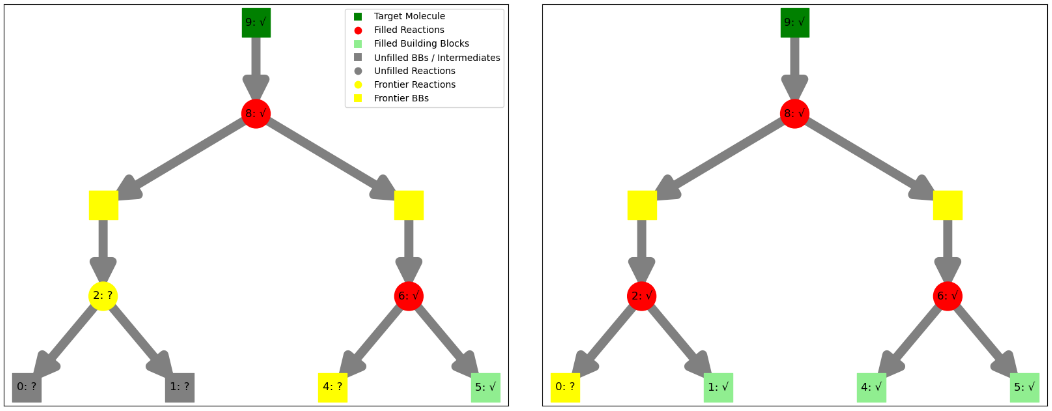


Figure 12: Examples of T' where prediction targets are the frontier reactions (yellow circles), frontier building blocks (numbered yellow squares) and auxiliary intermediates (un-numbered yellow squares).

outcome of the reaction. If $\text{RXN}(T)$ is the reaction nodes of T, we can construct T' from T as follows:

$$\mathcal{N}(T') \leftarrow \mathcal{N}(T) \cup \text{RXN}(T) \tag{3}$$

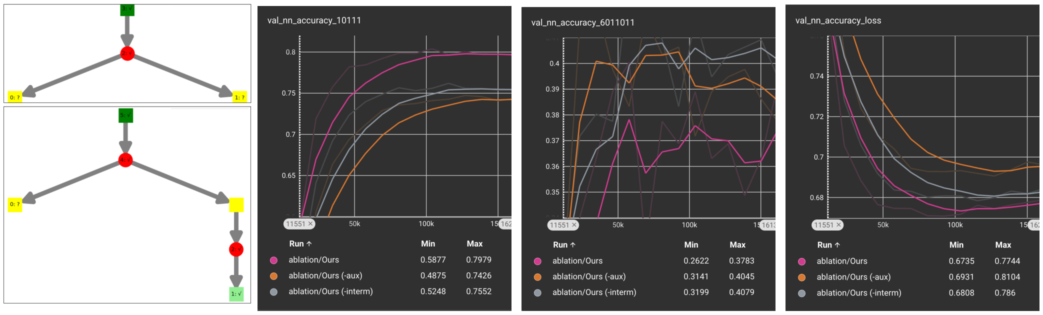
$$\mathcal{E}(T') \leftarrow \mathcal{E}(T) \cup \{(\text{parent}(r), o_r), (o_r, r) \forall r \in \text{RXN}(T)\} \tag{4}$$

Lastly, we attribute each o_r with the intermediate obtained from the original synthetic tree, i.e. executing the output of the program rooted at r. We featurize $\{y_o := \text{FP}_{256}(o_{\text{SMILES}})\}$ and add them as additional prediction targets to $\mathcal{D}$. Examples of T' are given in Figure 12.

F.4 Ablation Study: Auxiliary Task

To understand whether the two key design choices for $\partial\mathcal{P}'$ are justified, we did two ablations:

1. We use the original description of $\partial\mathcal{P}$ in Section 3.3.1, i.e. without the auxiliary task.
2. We use $\partial\mathcal{P}'$, but without attributing the intermediate nodes (so the set of targets is the same as Ablation 1.)



Run ↑	Min	Max
ablation/Ours	0.5877	0.7979
ablation/Ours (-aux)	0.4875	0.7426
ablation/Ours (-interm)	0.5248	0.7552

Run ↑	Min	Max
ablation/Ours	0.2622	0.3783
ablation/Ours (-aux)	0.3141	0.4045
ablation/Ours (-interm)	0.3199	0.4079

Run ↑	Min	Max
ablation/Ours	0.6735	0.7744
ablation/Ours (-aux)	0.6931	0.8104
ablation/Ours (-interm)	0.6808	0.786

(a) Examples from $\partial\mathcal{P}'$

(b) NN accuracy loss over top example Figure 15a

(c) NN accuracy over bottom example Figure 15a

(d) NN accuracy over the validation set

Figure 13: We compare the proposed ablations on the NN accuracy metric over the whole dataset as well as on two specific syntactic classes.

As shown in Figure 13d, using $\partial\mathcal{P}'$ (Ours) achieves higher NN accuracy. This shows the benefit of learning the auxiliary training task. Meanwhile, ablating the auxiliary task (-aux) and ablating the intermediate node (-interm) does not have meaningful difference, indicating our architecture is robust to graph edits which are semantically equivalent. To understand the comparative advantage vs disadvantage of the auxiliary training task, consider the two examples in Figure 15a. The first example is equivalent to learning a single-step backward reaction prediction on *forward* templates[3]. Our model clearly benefits from the auxiliary training task, which provides additional examples for learning the backward reaction steps. However, our model fares worse on predicting the first reactant of the top reaction. This may be due to competing resources. Despite the task being the same (and the set of forward templates are fixed), the model has to allocate sufficient capacity for the auxiliary task, whose output domain is much higher dimensional than $\mathcal{B}$. Ensuring positive transfer from learning the auxiliary task is an interesting extension for future work.

F.5 Ablation Study: Linear Training

Table 6: We follow the same setup as Table 1, but retrain Ours using a dataset constructed with parameter $k = 4$ (whereas we used $k = 3$ in Table 1) to match the linear sampling strategy models, which we refer to as Ours:MC. We evaluate models trained using sampling constants 1 and 10, as described in 4.1.3.

			Avg. Sim. ↑			SA ↓			Diversity ↑	
Dataset	Method	RR ↑	Top-1	Top-3	Top-5	Top-1	Top-3	Top-5	Top-3	Top-5
Test Set	Ours (τ)	56%	0.827	0.633	0.555	3.100	3.019	2.918	0.543	0.628
	Ours:MC1 (τ)	36%	0.732	0.564	0.513	3.048	2.913	2.844	0.609	0.665
	Ours:MC10 (τ)	65%	0.869	0.658	0.609	3.163	3.000	2.928	0.558	0.610
ChEMBL	Ours (τ)	7.6%	0.531	0.443	0.396	2.544	2.510	2.460	0.675	0.727
	Ours (MCMC)	9.2%	0.532	0.486	0.432	2.364	2.310	2.263	0.765	0.759
	Ours:MC1 (MCMC)	2.0%	0.406	0.337	0.289	2.604	2.563	2.439	0.756	0.767
	Ours:MC10 (MCMC)	8.5%	0.519	0.421	0.367	2.644	2.420	2.331	0.618	0.640

F.5.1 Results on Synthesizable Analog Generation

We study whether the efficiency of a sampling-based training strategy comes at a cost of performance. We make two observations from the results in Table 6.

Constant factor matters. The constant multiplier C from $\mathcal{D}_0$ to $\mathcal{D}$ not only determines how much data the model sees each pass for the sake of efficiency. It is also be a parameter controlling the tradeoff between over-representing larger vs smaller templates. If it is larger than the largest number of masks (Table 5), the Linear strategy is essentially deactivated, since all masks will be used. At a lower value, only small programs with at most the number of masks as C are fully represented. Medium-to-larger programs in $\mathcal{D}_0$ are under-represented, at the rate of the fraction of total masks that C constitutes for its template class.

Performance boost in-distribution. We find that for $C = 10$, the performance is *better* across reconstruction, similarity and diversity with comparable SA to the standard training. Meanwhile for $C = 1$, the performance declines sharply. It is likely that standard training is overfitting to masks from larger programs, resulting in poorer generalization. Meanwhile, $C = 10$ downsamples those programs, and its sampling can be viewed as data-level regularization against overfitting.

Slight performance drop out-of-distribution. We find both $C = 1$ and $C = 10$ underperform compared to standard training. For $C = 10$, reconstruction and Top-1 similarity are actually comparable, but its similarity, SA and diversity are noticeably worse than standard training. Since ChEMBL feature predominantly unsynthesizable molecules, it is likely that the distribution of molecular fingerprints better reflect those outputs of the more complex programs in $\mathcal{D}_0$, which are downweighted by higher values of C.

F.5.2 Results on Synthesizable Molecular Design

Depends on the task. We also include preliminary results on synthesizable molecular design. We find the results hold up for the MC models. Encouragingly, for the difficult task of Osimertinib MPO,

[3] For some templates, the forward template is one-to-one. For others, applying the backward template results in an ill-defined precursor, due to the many-to-one characteristic of these templates.

Figure 14: We select the first Oracle from each Table in App. H to compare Ours with Ours (EP). Aside from the ablation networks, we use the same experimental settings as Table 2.

(a)

		GSK3β										Median 1									
			Top 1			Top 10			Top 100					Top 1			Top 10			Top 100	
	category	Oracle Calls	Score	SA	AUC	Score	SA	AUC	Score	SA	AUC	Oracle Calls	Score	SA	AUC	Score	SA	AUC	Score	SA	AUC
Ours (MC)	synthesis	5056	**0.98**	**2.045**	**0.923**	**0.97**	**2.294**	**0.893**	0.942	**2.294**	**0.814**	7949	**0.4**	4.12	0.356	**0.342**	**3.902**	0.304	0.295	**4.013**	**0.256**
Ours (MC10)	synthesis											8045	**0.4**	**3.353**	0.357	0.344	4.593	0.297	**0.302**	4.44	0.247
Ours	synthesis	4886	**0.98**	**2.045**	0.891	0.967	2.302	0.848	**0.944**	2.27	0.778	8303	**0.4**	**3.353**	**0.371**	**0.342**	4.161	**0.305**	0.298	4.256	0.252

(b)

		Osimertinib MPO										Perindopril MPO									
				Top 1			Top 10			Top 100				Top 1			Top 10			Top 100	
	category	Oracle Calls	Score	SA	AUC	Score	SA	AUC	Score	SA	AUC	Oracle Calls	Score	SA	AUC	Score	SA	AUC	Score	SA	AUC
Ours (MC)	synthesis	9402	0.865	2.282	0.830	0.853	**2.187**	0.813	0.841	2.189	0.771	10000	0.572	**3.101**	0.541	0.567	**3.072**	0.521	0.555	3.077	0.486
Ours (MC10)	synthesis	5056	**0.98**	**2.045**	**0.923**	**0.97**	2.294	**0.893**	**0.942**	**2.294**	**0.814**	10000	0.596	3.263	0.542	0.574	3.147	0.523	0.556	3.058	**0.489**
Ours	synthesis	10000	0.859	2.263	0.826	0.847	2.21	0.81	0.832	2.249	0.769	10000	**0.622**	3.338	**0.547**	**0.591**	3.378	**0.524**	**0.558**	**3.137**	0.485

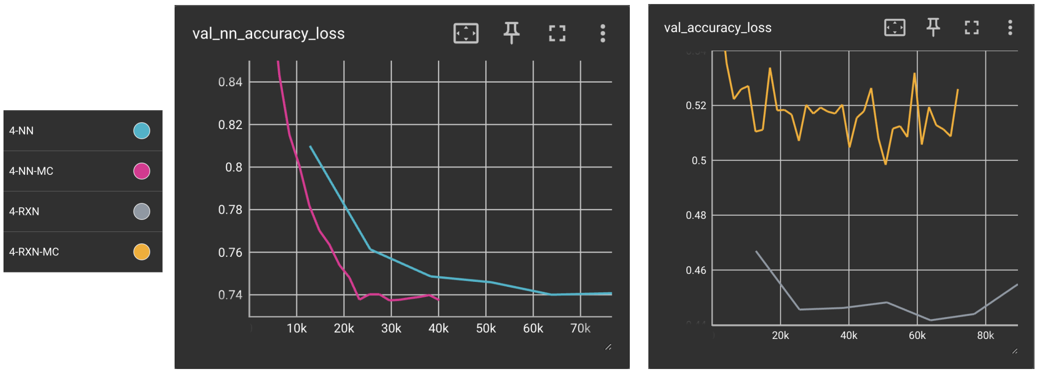
we see for $C = 10$, the results are substantially better. At the same time, Ours remain better for Perindopril MPO, which suggests each training strategy may suit different tasks. We also see for easier tasks like GSK and Median 1, the $C = 1$ model and Ours are essentially interchangeable. This suggests even a low reconstruction accuracy suffices for these Oracles.

Implications. The stratified sampling is designed to enhance supervised policy learning on larger programs while preventing the combinatorial explosion in Algo. 1 and to some extent, overfitting. However, this may benefit easier tasks (e.g. distributions of synthesizable molecules) while disadvantaging harder tasks. Thus, C should be tuned for optimal trade-off between efficiency and downstream synthesizable analog generation performance.

F.6 Model Architecture

We opt for two Graph Neural Networks (for Φ, Ω), each with 5 modules. Each module uses a TransformerConv layer (Shi et al., 2020) (we use 8 attention heads), a ReLU activation, and a Dropout layer. We adopt sinusoidal positional embeddings via numbering nodes using the postorder traversal (to preserve the pairwise node relationships for the same skeleton). Then, we pretrain Φ, Ω with $\mathcal{D}$.

F.7 Training & Convergence Analysis



(a) Legend (b) NN training step vs NN accuracy (c) RXN training step vs accuracy

Figure 15: We plot the number of training steps needed to converge our models under the standard and linear (-MC) training strategies for $C = 1$.

We elaborate on 4.1.3 further with a quantitative comparison of training costs with SynNet Gao et al. (2022). The key difference is $\mathcal{D}_{\text{synnet}}$ batches by *the size of a synthetic tree*, whereas we batch by the synthetic trees (in program form).

SynNet Training Complexity: SynNet serializes the construction of a synthetic tree, so a training epoch does $O(\sum_{\mathcal{D}_0} |T|)$ passes, where $|T|$ is the number of nodes (or number of edges, as they are different by one). The cost of a MLP forward and backward pass for SynNet is

$O(L \cdot H^2 + F \cdot H)$. The total complexity per epoch is $O((\sum |T|)(L \cdot H^2 + F \cdot H))$.

SynthesisNet Linear Training Complexity: As discussed in 4.1.3, a training epoch does $O(\mathcal{D}_0)$ passes, where the constant factor can be adjusted. The cost of a GNN forward and backward pass on a tree T is $O(L \cdot |T| \cdot (F \cdot H + H^2))$, where L, F, H are number of layers, feature and hidden dimension, respectively. The total complexity per epoch is $O(L \cdot (\sum |T|) \cdot (F \cdot H + H^2))$ since we train on the forest of trees over $\mathcal{D}_0$. This is equivalent to $O((\sum |T|)(L \cdot H^2 + F \cdot H))$.

We can conclude the per-epoch complexity following the linear training strategy is equivalent to that of SynNet's. However, what matters in practice is the convergence rate, so we also include a quantitative comparison between convergence plots. We find that even SynthesisNet's standard training strategy is practically equivalent to SynNet in the number of passes needed to converge the model. As a disclaimer, we only include SynthesisNet's numbers using the default setting of $k = 4$, where $|\mathcal{D}_0| \approx 135k$ and $|\mathcal{D}| \approx 818k$. We also show results of linear training with constant factor of 1 (i.e. $|\mathcal{D}| = |\mathcal{D}_0|$).

Convergence Comparison: Both SynNet and SynthesisNet uses a batch size of 64. We see from Figure 15 that SynthesisNet requires $\approx 170k$ training steps (batches) to converge (combining both BB and RXN networks) following standard training. Meanwhile, we refer readers to Figure 13 of Gao et al. (2021), where the convergence plots show at least (with the most generous interpretation) 1M steps needed to converge *each* of the action, reactant1, reaction and reactant2 networks. Pooling the training of all networks, we give a *very* generous estimate of $1M$ batches for SynNet to converge. What's left is to figure the average scaling size factor from a batch of trees (ours) to a batch of tree nodes (SynNet), which computes as $818k/135k \approx 6$, which implies $\approx 6 \cdot 170k \approx 1M$ steps. We can conclude the SynthesisNet with *standard* training is comparable if not more efficient in the number of training steps to converge the model. Should efficiency be of further concern, we suggest using the Linear Training strategy, and adjusting the sample constant factor accordingly.

Training and Inference Time: Converging the RXN and BB networks took us ≈ 1 and 5 hours on a single NVIDIA RTX A6000. A single inference call to the surrogate takes a few seconds.

F.8 Attention Visualization

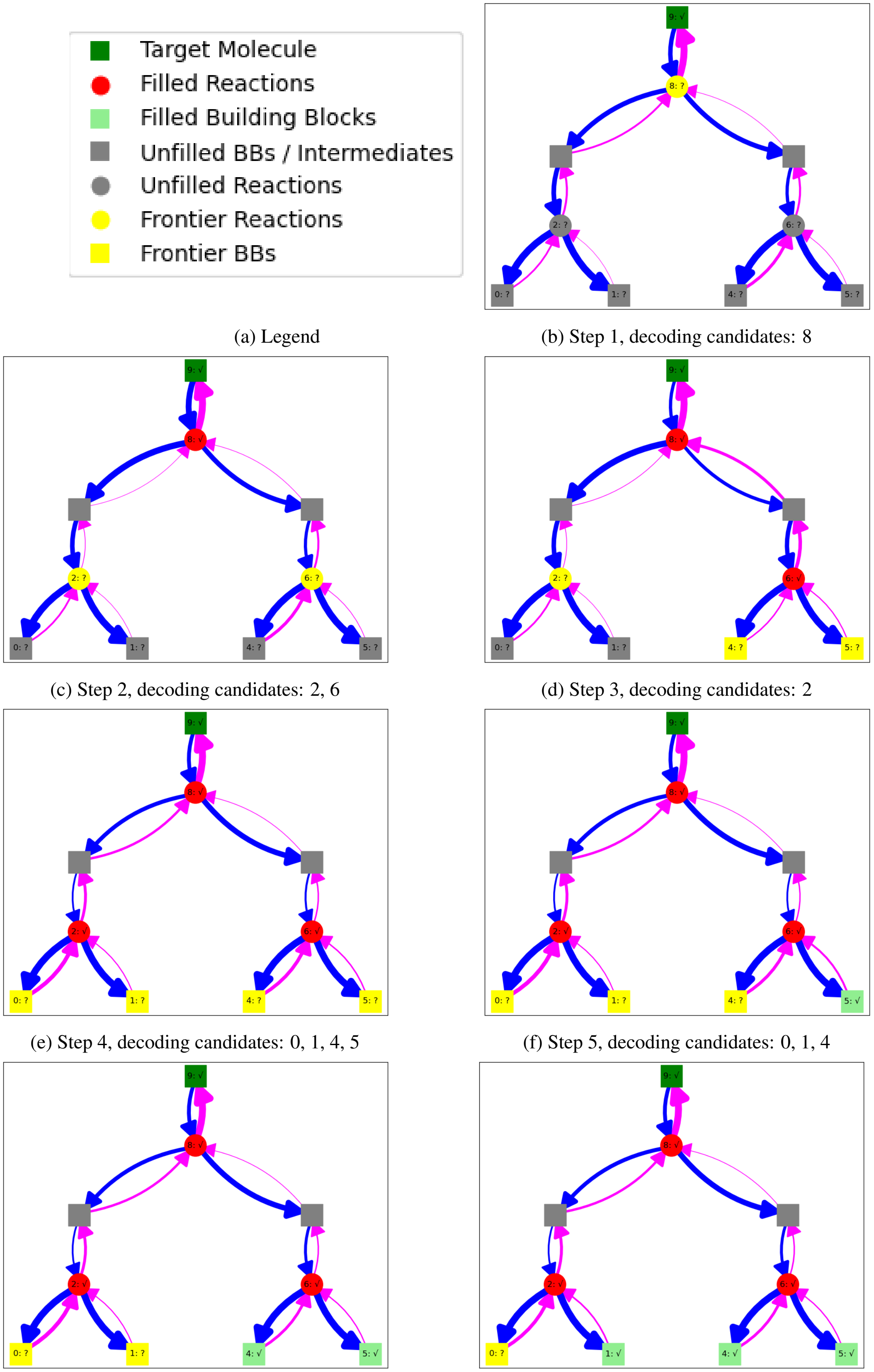
We elucidate how our policy network leverages the full horizon of the MDP to dynamically adjust the propagation of information throughout the decoding process. Since our decoding algorithm decodes once for every topological order of the nodes, the actual attention dynamics can vary significantly. Thus, we show a prototypical decoding order where:

1. All reactions are decoded before building blocks.
2. If decoding a reaction, the reaction node which $\pi_{\mathcal{R}}$ predicts with the highest probability is decoded.
3. If decoding a building block, the node where the embedding from $\pi_{\mathcal{B}}$ has minimal distance to a building block is decoded.

In (Shi et al., 2020), each TransformerConv layer l produces an attention weight for each edge, $[\alpha_{i,j}^{(l)}]$ where $\sum_j \alpha_{i,j}^{(l)} = 1$. We average over all layers to obtain the mean attention weight for each directed edge, i.e., we set the thickness of each edge (i, j) in each subfigure of Figure 16 to be proportional $\sum_l \alpha_{i,j}^{(l)}$.

We make some generation observations:

- The information flow along child-parent edges indicate usage of the full horizon. This is the main feature of our approach compared to traditional search methods like retrosynthesis.
- Our positional embeddings enables asymmetric modeling. SMIRKS templates specify the order of reactants and is usually not arbitrary. We observe that more often than not, the parent attends to its left child more than the right child. This may be a consequence of template definition conventions, where the first reactant is the major precursor. The subtree under the node more likely to be the major precursor is more important for predicting the reaction.



(a) Legend

(b) Step 1, decoding candidates: 8

(c) Step 2, decoding candidates: 2, 6

(d) Step 3, decoding candidates: 2

(e) Step 4, decoding candidates: 0, 1, 4, 5

(f) Step 5, decoding candidates: 0, 1, 4

(g) Step 6, decoding candidates: 0, 1

(h) Step 7, decoding candidates: 0

Figure 16: Case Study of Attention Flow

Now, we do a detailed walkthrough the 7-step decoding process to understand the evolution of the information flow. Each subfigure corresponds to the state of the MDP after a number of decoding steps, with the candidates of decoding colored in yellow. The attention scores are computed during the inference of Φ or Ω and averaged.

1. In Figure 16b, we see that 8 attends significantly to the target, unsurprisingly. 8 also attends to both its children, and attends more to its left child, which is a prior consistent with our general observation.
2. In Figure 16c, we see that after a specific reaction is instantiated at 8, the attention dynamics somewhat change. The edge from 8 to its left child thickens, while the edge from the left child to 8 thins. This is likely because now that the identity of 8 is known, it no longer needs to attend to its left child. The reciprocal relationship now intensifies, as the first reactant of 8 now attends to 8.
3. In Figure 16d, after the reaction at 6 is decoded, we see the information propagate back up the tree and to the other subtree to inform 2. We see the edge along the path from $6-8$ thickens, indicating the representation of 8 is informed with new information, and in turn propagates it to 2.
4. In Figure 16e, after the reaction at 2 is decoded, we see the same phenomenon happen, where the information flow again propagates back up and to the other subtree. However, we see this comes with a tradeoff, as 6 attends to its parent less, and instead reverts to its original attention strength to its children. We hypothesize the identity of 2 has a strong effect on the posterior of 6. This is an example where branching out to try more possible orders of decoding would facilitate a more complete algorithm.
5. In Figure 16f, we see how determining 5 causes 6 to attend more to 5 than it does to its parent. Knowledge of 5 allows the explaining away of 4.
6. In Figure 16g, we note instances of a general phenomenon: the second reactant is decoded followed by the first. Empirically, the distribution of the second reactant has lower entropy than the first. 4 was inferred after 5 as the knowledge of its parent reaction and sibling reactant likely constrains its posterior significantly.
7. In Figure 16h, we see a similar phenomenon where the representation of 2 attends slightly more to 1 after it is decoded.

In summary, the syntax structure of the full horizon is crucial during the decoding process. The attention scores allow us to visualize the dynamic propagation of information as nodes are decoded. Our observations highlights the flexibility of this approach compared to an infinite horizon formulation.

Table 7: Hyperparameters of our GA.

	Parameter	Value
General	Max. generations	200
	Population size	128
	Offspring size	512
	Seed initialization	random
	Fingerprint size	2048
	Early stopping warmup	30
	Early stopping patience	10
	Early stopping Δ	0.01
Semantic evolution	Parent selection prob. of i	$\propto (\text{rank}(i)+10)$
	Num. crossover bits n_{cross}	$\mathcal{N}(1024, 205)$
	Num. mutate bits n_{flip}	12
	Prob. mutate bits p_{flip}	0.5
Syntactic mutation	Num. tree edits n_{edit}	$\mathcal{U}\{1,2,3\}$
Surrogate	Max. topological orders	5
	Sampling strategy	greedy

G Genetic Algorithm

Our genetic algorithm (GA) is designed to mimic SynNet's (Gao et al., 2021), and settings are given in Table 7. We fix the same number of offspring fitness evaluations per generation to ensure a fair comparison, strategically allocating the evaluations between offspring generated using semantic evolution and those generated using syntactic mutation.

G.1 Semantic Evolution

Given two parents $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, semantic evolution samples a child $\boldsymbol{x}_*$ as follows. We combine n_{cross} random bits from $\boldsymbol{x}_1$ and the other $2048 - n_{\text{cross}}$ bits from $\boldsymbol{x}_2$ and then, with probability p_{flip}, flipping n_{flip} random bits of the crossover result.

G.2 Syntactic Mutation

Given a child $\boldsymbol{x}_*$ from Appendix G.1, syntactic mutation performs n_{edit} edits on $T = \tau(\boldsymbol{x}_*)$ to obtain a syntactic analog T. With equal probability, each edit either adds or removes a random leaf. To do so, we enumerate all possible additions and removals, and ignore the ones that produce an empty tree or a tree with more than 4 reaction nodes. The edit is uniformly sampled from all such choices, or no operation is performed if no viable choices exist. Using the surrogate, the siblings $(\boldsymbol{x}_*, T)$ and $(\boldsymbol{x}_*, T)$ are then turned into two fingerprints, and one of with the higher expected improvement under a Gaussian process (GP) is selected. Our GP uses a radial basis function kernel with length scale 1 and is fitted using the population and offspring from the preceding generation.

G.3 Surrogate Checkpoint

The surrogate checkpoint was trained as described in Appendix F. To lower the runtime of the GA, we only reconstruct using a random subset of the input skeleton's possible topological orders. For each topological order, we follow a greedy decoding scheme where reactions are decoded before building blocks, as described in Appendix F.8.

H Full Results on TDC Oracles

Table 8: Guacamol structural target-directed benchmarks: Median 1 & 2 (average similarity to multiple molecules) and Celecoxib Rediscovery (hit expansion around Celecoxib).

		Median 1									
			Top 1			Top 10			Top 100		
	category	Oracle Calls	Score	SA	AUC	Score	SA	AUC	Score	SA	AUC
synnet	synthesis	5245	0.245 (15|3)	4.066 (17|4)	0.237 (15|3)	0.228 (13|3)	3.807 (11|3)	0.219 (12|2)	0.198 (14|3)	3.687 (6|3)	0.188 (12|2)
pasithea	string	3876.8	0.216 (22|9)	5.432 (26|10)	0.213 (21|8)	0.183 (24|10)	4.486 (23|8)	0.179 (24|10)	0.138 (26|10)	4.319 (18|6)	0.134 (25|10)
dog_ae	synthesis	1448.4	0.204 (25|4)	3.285 5|1	0.201 (25|4)	0.174 (25|4)	3.192 4|1	0.172 (25|4)	0.139 (24|4)	**2.984 (1|1)**	0.135 (24|4)
smiles_vae_bo	string	10000	0.267 (14|6)	4.307 (20|7)	0.253 (12|6)	0.223 (14|6)	4.244 (20|6)	0.203 (16|6)	0.181 (17|7)	4.252 (14|5)	0.161 (18|7)
jt_vae_bo	graph	4706.6	0.212 (23|9)	5.21 (25|10)	0.21 (22|9)	0.184 (23|9)	4.187 (17|9)	0.18 (22|9)	0.151 (23|9)	4.052 (10|4)	0.145 (22|8)
moldqn	graph	10000	0.188 (26|10)	4.654 (23|9)	0.144 (26|10)	0.169 (26|10)	4.972 (25|10)	0.123 (26|10)	0.139 (24|10)	5.387 (25|10)	0.094 (26|10)
mars	graph	3698.6	0.233 (18|7)	3.823 (15|7)	0.228 (17|6)	0.217 (16|5)	4.063 (14|7)	0.208 (14|5)	0.181 (17|6)	4.13 (11|5)	0.17 (15|5)
selfies_lstm_hc	string	10000	0.363 (5|4)	3.533 (13|4)	0.269 (10|5)	0.339 (5|4)	3.669 (10|4)	0.24 (10|5)	0.286 (7|4)	3.833 (8|2)	0.201 (10|5)
gp_bo	graph	10000	0.345 (7|2)	4.111 (18|8)	0.317 (5|2)	0.333 (6|1)	3.992 (13|6)	0.302 (4|1)	0.317 (3|1)	4.452 (22|9)	0.276 (3|1)
smiles_ga	string	3093	0.208 (24|10)	4.708 (24|9)	0.205 (23|9)	0.2 (21|8)	5.686 (26|10)	0.192 (20|8)	0.199 (13|6)	5.93 (26|10)	0.186 (13|6)
mimosa	graph	10000	0.296 (10|3)	3.481 (11|4)	0.271 (9|3)	0.276 (10|3)	3.928 (12|5)	0.244 (9|3)	0.251 (10|3)	4.263 (16|7)	0.213 (8|3)
reinvent	string	6022	0.4 (1|1)	3.353 (6|2)	0.368 (2|1)	**0.399 (1|1)**	3.415 (6|2)	**0.357 (1|1)**	**0.383 (1|1)**	4.032 (9|3)	**0.325 (1|1)**
smiles_lstm_hc	string	10000	0.389 (4|3)	4.199 (19|6)	0.299 (7|3)	0.351 (3|3)	4.299 (21|7)	0.256 (7|4)	0.315 (4|3)	4.243 (13|4)	0.214 (7|4)
selfies_vae_bo	string	10000	0.232 (19|7)	4.539 (22|8)	0.226 (18|7)	0.212 (18|7)	4.612 (24|9)	0.202 (17|7)	0.175 (19|8)	4.45 (21|8)	0.16 (19|8)
dog_gen	synthesis	8862.6	0.323 (8|2)	3.459 (10|3)	0.244 (14|2)	0.296 (8|2)	3.419 (7|2)	0.218 (13|3)	0.261 (9|2)	3.483 (4|2)	0.182 (14|3)
stoned	string	4140.8	0.295 (11|5)	3.911 (16|5)	0.282 (8|4)	0.283 (9|5)	4.236 (19|5)	0.267 (6|3)	0.264 (8|5)	4.782 (24|9)	0.245 (6|3)
gflownet	graph	10000	0.237 (17|6)	2.83 (3|3)	0.225 (19|7)	0.217 (16|5)	3.026 (3|3)	0.202 (17|6)	0.187 (15|5)	3.332 (3|2)	0.166 (16|6)
reinvent_selfies	string	4209.2	0.4 (1|1)	3.353 (6|2)	0.368 (2|1)	0.396 (2|2)	3.399 (5|1)	0.356 (2|2)	0.34 (2|2)	3.779 (7|1)	0.3 (2|2)
graph_mcts	graph	10000	0.243 (16|5)	2.692 (2|2)	0.235 (16|5)	0.212 (18|7)	**2.898 (1|1)**	0.196 (19|7)	0.164 (21|7)	3.545 (5|3)	0.144 (23|9)
dst	graph	10000	0.281 (12|4)	3.754 (14|6)	0.257 (11|4)	0.263 (11|4)	3.651 (9|4)	0.233 (11|4)	0.231 (11|4)	4.153 (12|6)	0.194 (11|4)
selfies_ga	string	8900.6	0.219 (21|8)	2.877 (4|1)	0.202 (24|10)	0.199 (22|9)	3.467 (8|3)	0.18 (22|9)	0.172 (20|9)	4.445 (20|7)	0.153 (20|9)
gflownet_al	graph	10000	0.229 (20|8)	**1.877 (1|1)**	0.224 (20|8)	0.203 (20|8)	2.933 (2|2)	0.191 (21|8)	0.164 (21|7)	3.262 (2|1)	0.146 (21|7)
screening	N/A	10000	0.272 (13|2)	4.361 (21|2)	0.25 (13|2)	0.223 (14|2)	4.37 (22|2)	0.206 (15|2)	0.182 (16|2)	4.386 (19|1)	0.162 (17|2)
mol_pal	N/A	10000	0.31 (9|1)	3.385 (9|1)	0.302 (6|1)	0.257 (12|1)	4.21 (18|1)	0.25 (8|1)	0.211 (12|1)	4.645 (23|2)	0.203 (9|1)
graph_ga	graph	10000	0.351 (6|1)	3.492 (12|5)	0.32 (4|1)	0.331 (7|2)	4.137 (15|8)	0.295 (5|2)	0.31 (5|2)	4.304 (17|8)	0.265 (4|2)
Ours	synthesis	8303	**0.4 (1|1)**	3.353 (6|2)	**0.371 (1|1)**	0.342 4|1	4.161 (16|4)	0.305 (3|1)	0.298 6|1	4.256 (15|4)	0.252 (5|1)

		Median 2									
			Top 1			Top 10			Top 100		
	category	Oracle Calls	Score	SA	AUC	Score	SA	AUC	Score	SA	AUC
synnet	synthesis	4350.6	0.259 (11|3)	2.364 (5|3)	0.253 (8|2)	0.244 (10|3)	2.473 2|1	0.237 (8|2)	0.214 (11|3)	**2.459 (1|1)**	0.206 (8|2)
pasithea	string	2010.8	0.195 (21|9)	2.882 (13|6)	0.194 (21|9)	0.182 (22|9)	2.747 (9|4)	0.181 (20|9)	0.156 (23|9)	2.813 (11|5)	0.154 (23|9)
dog_ae	synthesis	1119.2	0.201 (19|4)	2.347 3|1	0.199 (17|4)	0.185 (20|4)	2.495 (3|2)	0.183 (18|4)	0.16 (22|4)	2.591 (4|3)	0.157 (21|4)
smiles_vae_bo	string	10000	0.223 (14|6)	2.759 (10|4)	0.213 (14|6)	0.207 (15|7)	2.608 (5|2)	0.196 (15|7)	0.182 (15|7)	2.626 (6|2)	0.17 (15|7)
jt_vae_bo	graph	5246.8	0.193 (22|7)	3.013 (14|3)	0.192 (22|7)	0.183 (21|7)	3.125 (17|4)	0.181 (20|6)	0.165 (21|8)	3.196 (16|3)	0.16 (18|6)
moldqn	graph	10000	0.109 (26|10)	5.837 (26|10)	0.095 (26|10)	0.1 (26|10)	5.666 (26|10)	0.089 (26|10)	0.085 (26|10)	5.554 (26|10)	0.072 (26|10)
mars	graph	4298.2	0.204 (17|4)	3.036 (15|4)	0.197 (18|4)	0.19 (18|5)	3.446 (22|9)	0.182 (19|5)	0.17 (18|6)	3.774 (23|9)	0.16 (18|6)
selfies_lstm_hc	string	10000	0.274 (8|4)	2.184 (2|1)	0.229 (12|5)	0.262 (9|5)	**2.402 (1|1)**	0.207 (12|5)	0.24 (9|5)	2.564 (3|1)	0.179 (13|6)
gp_bo	graph	10000	0.337 (2|1)	3.4 (22|8)	**0.31 (1|1)**	**0.33 (1|1)**	3.358 (21|8)	**0.298 (1|1)**	**0.313 (1|1)**	3.303 (18|5)	**0.276 (1|1)**
smiles_ga	string	6634.6	0.211 (15|7)	3.352 (20|8)	0.204 (15|7)	0.208 (14|6)	3.453 (23|8)	0.199 (14|6)	0.204 (12|6)	3.751 (22|8)	0.192 (10|5)
mimosa	graph	10000	0.238 (13|3)	2.739 (9|1)	0.228 (13|3)	0.229 (12|3)	2.826 (10|1)	0.215 (10|3)	0.216 (10|3)	3.08 (15|2)	0.196 (9|3)
reinvent	string	10000	0.333 (3|2)	2.475 (7|3)	0.29 (2|1)	0.326 (2|1)	2.661 (7|3)	0.278 (2|1)	0.313 (1|1)	2.737 (10|4)	0.259 (2|1)
smiles_lstm_hc	string	10000	**0.34 (1|1)**	3.266 (18|7)	0.277 (4|2)	0.318 (3|2)	3.026 (15|7)	0.249 (6|3)	0.291 (5|3)	2.682 (8|3)	0.218 (7|4)
selfies_vae_bo	string	10000	0.206 (16|8)	2.45 (6|2)	0.202 (16|8)	0.192 (17|8)	2.896 (13|6)	0.186 (16|8)	0.169 (19|8)	3.065 (14|7)	0.16 (18|8)
dog_gen	synthesis	10000	0.297 6|1	2.708 (8|4)	0.23 (11|3)	0.287 6|1	2.571 (4|3)	0.213 (11|3)	0.263 (7|2)	2.511 (2|2)	0.189 (12|3)
stoned	string	9341.8	0.266 (10|5)	3.785 (24|9)	0.25 (9|4)	0.264 (8|4)	3.913 (24|9)	0.246 (7|4)	0.26 (8|4)	4.097 (24|9)	0.237 (5|2)
gflownet	graph	10000	0.198 (20|6)	3.513 (23|9)	0.195 (19|5)	0.189 (19|6)	3.351 (20|7)	0.181 (20|6)	0.175 (17|5)	3.63 (20|7)	0.165 (17|5)
reinvent_selfies	string	9729.8	0.313 (5|3)	2.766 (11|5)	0.27 (5|3)	0.31 (5|3)	2.841 (12|5)	0.256 (5|2)	0.301 (3|2)	2.988 (12|6)	0.233 (6|3)
graph_mcts	graph	10000	0.149 (25|9)	3.203 (16|5)	0.142 (24|9)	0.14 (25|9)	3.123 (16|3)	0.133 (24|9)	0.128 (25|9)	3.252 (17|4)	0.118 (24|9)
dst	graph	10000	0.202 (18|5)	2.778 (12|2)	0.195 (19|5)	0.195 (16|4)	2.835 (11|2)	0.186 (16|4)	0.178 (16|4)	3.039 (13|1)	0.167 (16|4)
selfies_ga	string	10000	0.161 (24|10)	5.071 (25|10)	0.129 (25|10)	0.157 (24|10)	5.005 (25|10)	0.122 (25|10)	0.149 (24|10)	4.988 (25|10)	0.111 (25|10)
gflownet_al	graph	10000	0.191 (23|8)	3.269 (19|6)	0.183 (23|8)	0.182 (22|8)	3.338 (18|5)	0.174 (23|8)	0.167 (20|7)	3.717 (21|8)	0.157 (21|8)
screening	N/A	10000	0.245 (12|2)	3.223 (17|2)	0.233 (10|2)	0.213 (13|2)	2.732 (8|1)	0.201 (13|2)	0.184 (14|2)	2.671 (7|2)	0.171 (14|2)
mol_pal	N/A	10000	0.273 (9|1)	**2.12 (1|1)**	0.267 (7|1)	0.237 (11|1)	2.897 (14|2)	0.232 (9|1)	0.198 (13|1)	2.593 (5|1)	0.192 (10|1)
graph_ga	graph	10000	0.324 (4|2)	3.383 (21|7)	0.289 (3|2)	0.316 (4|2)	3.346 (19|6)	0.274 (3|2)	0.3 (4|2)	3.335 (19|6)	0.252 (3|2)
Ours	synthesis	9117	0.292 (7|2)	2.353 (4|2)	0.269 (6|1)	0.285 (7|2)	2.649 (6|4)	0.257 (4|1)	0.272 6|1	2.682 (8|4)	0.238 (4|1)

		Celecoxib									
			Top 1			Top 10			Top 100		
	category	Oracle Calls	Score	SA	AUC	Score	SA	AUC	Score	SA	AUC
synnet	synthesis	7377.4	0.526 (9|3)	2.234 (14|4)	0.487 (9|3)	0.479 (9|3)	2.262 (7|3)	0.443 (8|3)	0.411 (9|3)	2.466 (12|4)	0.376 (8|3)
pasithea	string	3901.6	0.355 (22|9)	2.383 (18|7)	0.353 (21|8)	0.317 (21|9)	2.319 (14|4)	0.314 (21|9)	0.247 (23|10)	2.512 (14|5)	0.243 (21|9)
dog_ae	synthesis	1097.4	0.406 (17|4)	2.23 (13|3)	0.403 (15|4)	0.361 (17|4)	2.278 (10|4)	0.357 (15|4)	0.289 (20|4)	2.395 (7|3)	0.283 (18|4)
smiles_vae_bo	string	10000	0.425 (14|5)	2.225 (12|4)	0.409 (14|5)	0.382 (15|6)	2.312 (13|3)	0.355 (16|6)	0.325 (16|7)	2.374 (6|2)	0.293 (16|7)
jt_vae_bo	graph	4538	0.39 (20|7)	2.408 (20|7)	0.387 (19|6)	0.306 (22|7)	2.496 (18|6)	0.3 (22|7)	0.25 (22|7)	2.744 (19|6)	0.241 (22|7)
moldqn	graph	10000	0.128 (26|10)	4.708 (26|10)	0.115 (26|10)	0.112 (26|10)	4.951 (26|10)	0.1 (26|10)	0.094 (26|10)	5.255 (26|10)	0.08 (26|10)
mars	graph	7404.6	0.487 (11|3)	2.16 (8|4)	0.429 (10|3)	0.448 (10|3)	2.3 (11|4)	0.38 (14|5)	0.394 (11|4)	2.468 (13|3)	0.318 (15|5)
selfies_lstm_hc	string	10000	0.586 (8|4)	2.205 (10|2)	0.427 (11|4)	0.535 (8|4)	2.193 (3|1)	0.387 (11|4)	0.474 (8|4)	2.239 (2|1)	0.326 (14|6)
gp_bo	graph	10000	0.946 (2|1)	2.179 (9|5)	**0.809 (1|1)**	0.859 (2|1)	2.274 (8|2)	**0.725 (1|1)**	0.803 (2|1)	2.648 (16|5)	**0.638 (2|1)**
smiles_ga	string	3104.6	0.358 (21|8)	3.378 (24|9)	0.352 (22|9)	0.356 (19|7)	3.801 (24|9)	0.345 (18|7)	0.35 (15|6)	3.999 (24|9)	0.332 (13|5)
mimosa	graph	9260	0.438 (13|5)	3.064 (22|9)	0.422 (13|5)	0.428 (11|4)	3.081 (21|8)	0.395 (10|3)	0.406 (10|3)	3.322 (22|9)	0.355 (10|3)
reinvent	string	6641.8	**0.96 (1|1)**	2.157 (7|1)	0.803 (2|1)	**0.862 (1|1)**	2.355 (16|6)	0.715 (2|1)	**0.821 (1|1)**	2.724 (17|6)	0.647 (1|1)
smiles_lstm_hc	string	10000	0.851 (3|2)	2.22 (11|3)	0.621 (5|2)	0.786 (3|2)	2.311 (12|2)	0.54 (6|3)	0.695 (3|2)	2.403 (9|3)	0.45 (6|3)
selfies_vae_bo	string	10000	0.391 (19|7)	2.366 (17|6)	0.389 (17|6)	0.353 (20|8)	2.323 (15|5)	0.327 (20|8)	0.29 (19|8)	2.446 (10|4)	0.262 (20|8)
dog_gen	synthesis	9319	0.76 5|1	2.09 (6|2)	0.526 (7|2)	0.682 6|1	2.189 2|1	0.466 (7|2)	0.584 6|1	2.243 (3|2)	0.389 (7|2)
stoned	string	6255	0.401 (18|6)	3.129 (23|8)	0.389 (17|6)	0.398 (14|5)	3.477 (23|8)	0.383 (12|5)	0.393 (12|5)	3.657 (23|8)	0.368 (9|4)
gflownet	graph	10000	0.409 (16|6)	2.864 (21|8)	0.375 (20|7)	0.36 (18|6)	3.12 (22|9)	0.328 (19|6)	0.308 (18|6)	3.237 (21|8)	0.276 (19|6)
reinvent_selfies	string	7480.2	0.751 (7|3)	2.283 (15|5)	0.618 (6|3)	0.722 (5|3)	2.568 (19|7)	0.574 (5|2)	0.685 (5|3)	2.725 (18|7)	0.516 (4|2)
graph_mcts	graph	10000	0.329 (24|9)	2.047 (3|1)	0.297 (23|8)	0.296 (23|8)	2.256 (6|1)	0.265 (23|8)	0.233 (25|9)	2.4 (8|1)	0.205 (24|9)
dst	graph	9723.8	0.459 (12|4)	2.082 (5|3)	0.424 (12|4)	0.422 (13|5)	2.277 (9|3)	0.381 (13|4)	0.381 (13|5)	2.446 (10|2)	0.335 (12|4)
selfies_ga	string	10000	0.289 (25|10)	3.961 (25|10)	0.242 (25|10)	0.27 (25|10)	4.122 (25|10)	0.224 (25|10)	0.252 (21|9)	4.547 (25|10)	0.2 (25|10)
gflownet_al	graph	10000	0.333 (23|8)	2.4 (19|6)	0.29 (24|9)	0.286 (24|9)	2.686 (20|7)	0.258 (24|9)	0.24 (24|8)	3.018 (20|7)	0.214 (23|8)
screening	N/A	10000	0.419 (15|2)	2.344 (16|2)	0.396 (16|2)	0.373 (16|2)	2.232 (5|2)	0.353 (17|2)	0.317 (17|2)	2.332 (5|2)	0.29 (17|2)
mol_pal	N/A	10000	0.511 (10|1)	**1.908 (1|1)**	0.498 (8|1)	0.427 (12|1)	**2.097 (1|1)**	0.416 (9|1)	0.364 (14|1)	2.245 (4|1)	0.351 (11|1)
graph_ga	graph	9685.4	0.811 (4|2)	2.067 (4|2)	0.684 (3|2)	0.757 (4|2)	2.406 (17|5)	0.631 (3|2)	0.693 (4|2)	2.622 (15|4)	0.559 (3|2)
Ours	synthesis	7042	0.753 (6|2)	1.959 2|1	0.662 (4|1)	0.668 (7|2)	2.195 (4|2)	0.582 (4|1)	0.572 (7|2)	**2.211 (1|1)**	0.503 (5|1)

Table 9: Bioactivity Oracles for GSK3B, JNK3, and DRD2

		GSK3β									
			Top 1			Top 10			Top 100		
	category	Oracle Calls	Score	SA	AUC	Score	SA	AUC	Score	SA	AUC
synnet	synthesis	9059.4	0.932 (8|3)	2.576 (5|2)	0.855 (6|3)	0.901 (8|3)	2.747 (6|3)	0.79 (6|3)	0.797 (9|3)	2.897 (6|4)	0.656 (7|2)
pasithea	string	2531.2	0.414 (24|10)	3.34 (14|5)	0.402 (23|9)	0.294 (25|10)	3.277 (11|3)	0.282 (24|10)	0.151 (26|10)	3.782 (13|4)	0.141 (26|10)
dog_ae	synthesis	1319	0.778 (12|4)	2.727 (8|4)	0.756 (10|4)	0.624 (14|4)	2.86 (7|4)	0.602 (13|4)	0.378 (18|4)	2.874 (5|3)	0.356 (16|4)
smiles_vae_bo	string	10000	0.606 (20|7)	3.005 (10|2)	0.537 (20|7)	0.473 (21|8)	2.901 (8|2)	0.386 (20|7)	0.284 (21|8)	2.948 (7|2)	0.215 (21|8)
jt_vae_bo	graph	4972.4	0.512 (23|8)	3.546 (16|4)	0.483 (22|8)	0.38 (23|8)	3.403 (14|3)	0.351 (21|8)	0.223 (24|9)	3.581 (11|3)	0.202 (23|8)
moldqn	graph	10000	0.344 (26|10)	5.731 (23|10)	0.287 (26|10)	0.286 (26|10)	5.871 (23|10)	0.242 (26|10)	0.218 (25|10)	6.215 (24|10)	0.177 (25|10)
mars	graph	4285.4	0.684 (17|6)	3.277 (12|3)	0.63 (18|7)	0.607 (17|7)	4.148 (17|5)	0.553 (17|7)	0.536 (15|7)	4.598 (19|6)	0.464 (15|7)
selfies_lstm_hc	string	10000	0.65 (19|6)	3.854 (18|7)	0.54 (19|6)	0.602 (18|6)	4.311 (18|7)	0.424 (19|6)	0.503 (16|6)	4.422 (17|7)	0.293 (19|7)
gp_bo	graph	10000	0.986 (3|1)	2.74 (9|2)	0.879 (5|1)	0.975 (3|1)	3.057 (10|2)	0.852 (2|1)	0.957 (3|1)	3.133 (10|2)	0.809 (2|1)
smiles_ga	string	6214.6	0.722 (15|5)	6.22 (24|8)	0.669 (14|5)	0.709 (11|5)	6.321 (24|8)	0.63 (12|5)	0.687 (11|5)	6.214 (23|8)	0.587 (11|5)
mimosa	graph	10000	0.718 (16|5)	5.328 (21|8)	0.641 (17|6)	0.701 (12|4)	4.942 (21|8)	0.555 (16|6)	0.672 (12|4)	4.903 (21|8)	0.475 (14|6)
reinvent	string	6530.4	0.972 (5|2)	3.106 (11|3)	0.894 (3|2)	0.969 (4|2)	3.401 (13|5)	**0.866 (1|1)**	**0.965 (1|1)**	3.735 (12|3)	**0.824 (1|1)**
smiles_lstm_hc	string	10000	**1.0 (1|1)**	2.406 (3|1)	0.938 (2|1)	0.984 (2|1)	2.44 (2|1)	0.84 (4|2)	0.943 (5|2)	2.414 (2|1)	0.671 (6|3)
selfies_vae_bo	string	10000	0.564 (21|8)	3.525 (15|6)	0.507 (21|8)	0.421 (22|9)	3.382 (12|4)	0.351 (21|8)	0.262 (22|9)	3.786 (14|5)	0.207 (22|9)
dog_gen	synthesis	10000	**1.0 (1|1)**	2.592 (7|3)	**0.961 (1|1)**	**0.989 (1|1)**	2.586 (3|2)	0.832 (5|2)	0.96 (2|1)	2.601 (3|2)	0.629 (8|3)
stoned	string	8132.4	0.766 (13|4)	6.235 (25|9)	0.704 (12|4)	0.757 (10|4)	6.33 (25|9)	0.67 (10|4)	0.734 (10|4)	6.341 (25|9)	0.622 (9|4)
gflownet	graph	10000	0.726 (14|4)	4.19 (19|6)	0.693 (13|4)	0.692 (13|5)	4.437 (19|6)	0.652 (11|4)	0.638 (13|5)	4.545 (18|5)	0.586 (12|4)
reinvent_selfies	string	7079.2	0.964 (6|3)	3.337 (13|4)	0.824 (8|3)	0.957 (6|3)	3.87 (16|6)	0.781 (8|3)	0.935 (6|3)	3.997 (16|6)	0.712 (5|2)
graph_mcts	graph	10000	0.405 (25|9)	3.69 (17|5)	0.355 (25|9)	0.334 (24|9)	3.568 (15|4)	0.282 (24|9)	0.232 (23|8)	3.848 (15|4)	0.184 (24|9)
dst	graph	9058.2	0.862 (9|3)	5.478 (22|9)	0.739 (11|3)	0.844 (9|3)	5.423 (22|9)	0.672 (9|3)	0.822 (8|3)	5.708 (22|9)	0.599 (10|3)
selfies_ga	string	10000	0.534 (22|9)	7.408 (26|10)	0.363 (24|10)	0.512 (20|7)	7.348 (26|10)	0.343 (23|9)	0.483 (17|7)	7.19 (26|10)	0.309 (18|6)
gflownet_al	graph	10000	0.676 (18|7)	4.197 (20|7)	0.643 (16|5)	0.623 (15|6)	4.487 (20|7)	0.59 (14|5)	0.555 (14|6)	4.74 (20|7)	0.505 (13|5)
screening	N/A	10000	0.836 (10|1)	2.305 (2|1)	0.658 (15|2)	0.561 (19|2)	2.739 (5|2)	0.439 (18|2)	0.312 (20|2)	2.959 (8|2)	0.236 (20|2)
mol_pal	N/A	10000	0.82 (11|2)	2.589 (6|2)	0.776 (9|1)	0.62 (16|1)	2.723 (4|1)	0.556 (15|1)	0.369 (19|1)	2.788 (4|1)	0.319 (17|1)
graph_ga	graph	9529	0.946 (7|2)	2.452 (4|1)	0.829 (7|2)	0.937 (7|2)	2.906 (9|1)	0.789 (7|2)	0.919 (7|2)	3.079 (9|1)	0.738 (4|2)
Ours	synthesis	4886	0.98 (4|2)	**2.045 (1|1)**	0.891 (4|2)	0.967 (5|2)	**2.302 (1|1)**	0.848 (3|1)	0.944 (4|2)	**2.27 (1|1)**	0.778 (3|1)

		JNK3									
			Top 1			Top 10			Top 100		
	category	Oracle Calls	Score	SA	AUC	Score	SA	AUC	Score	SA	AUC
synnet	synthesis	6240	0.798 (7|3)	2.116 (3|2)	0.723 (3|1)	0.715 (8|3)	**2.172 (1|1)**	0.631 (5|2)	0.563 (10|3)	2.244 (2|2)	0.466 (9|2)
pasithea	string	2707.4	0.21 (24|10)	2.804 (9|4)	0.207 (24|10)	0.158 (24|10)	2.669 (9|3)	0.155 (24|10)	0.081 (26|10)	3.433 (13|5)	0.076 (24|10)
dog_ae	synthesis	1219.8	0.554 (13|4)	2.38 (6|4)	0.54 (12|4)	0.493 (13|4)	2.625 (7|4)	0.47 (12|4)	0.263 (17|4)	2.734 (8|4)	0.246 (16|4)
smiles_vae_bo	string	10000	0.432 (18|5)	2.696 (8|3)	0.376 (17|5)	0.302 (21|8)	2.495 (6|2)	0.242 (18|6)	0.162 (21|8)	2.67 (6|2)	0.124 (21|8)
jt_vae_bo	graph	5853.2	0.404 (22|8)	2.884 (10|1)	0.354 (19|8)	0.257 (23|8)	2.894 (11|1)	0.223 (20|8)	0.139 (23|8)	3.208 (10|1)	0.12 (22|8)
moldqn	graph	10000	0.152 (26|10)	6.144 (24|9)	0.135 (26|10)	0.13 (26|10)	6.043 (23|9)	0.111 (25|9)	0.093 (24|9)	6.014 (23|9)	0.075 (25|9)
mars	graph	5520.6	0.646 (10|4)	3.6 (15|3)	0.549 (10|4)	0.587 (11|4)	3.652 (15|3)	0.489 (11|4)	0.497 (11|4)	3.941 (16|4)	0.387 (11|4)
selfies_lstm_hc	string	10000	0.428 (19|6)	2.668 (7|2)	0.305 (22|8)	0.303 (20|7)	3.402 (14|6)	0.207 (23|9)	0.216 (19|7)	3.529 (14|6)	0.137 (19|7)
gp_bo	graph	10000	0.698 (9|3)	3.753 (17|4)	0.593 (9|3)	0.69 (9|3)	3.744 (16|4)	0.566 (7|1)	0.676 (8|3)	3.867 (15|3)	0.525 (4|1)
smiles_ga	string	8293.6	0.414 (20|7)	5.871 (23|9)	0.34 (21|7)	0.393 (17|5)	6.37 (24|9)	0.317 (17|5)	0.375 (14|5)	6.55 (24|9)	0.289 (14|5)
mimosa	graph	10000	0.498 (15|6)	4.579 (20|7)	0.403 (16|7)	0.484 (14|6)	4.482 (20|7)	0.361 (15|7)	0.457 (12|5)	4.337 (19|6)	0.302 (13|6)
reinvent	string	8382.2	0.954 (2|2)	3.242 (14|6)	**0.814 (1|1)**	**0.949 (1|1)**	3.322 (12|5)	**0.784 (1|1)**	**0.943 (1|1)**	3.412 (12|4)	**0.743 (1|1)**
smiles_lstm_hc	string	10000	**0.968 (1|1)**	2.22 (4|1)	0.788 (2|2)	0.935 (2|2)	2.307 (3|1)	0.661 (2|2)	0.851 (2|2)	2.442 (3|1)	0.49 (5|3)
selfies_vae_bo	string	10000	0.414 (20|7)	3.041 (12|5)	0.342 (20|6)	0.263 (22|9)	2.756 (10|4)	0.209 (22|8)	0.147 (22|9)	3.133 (9|3)	0.113 (23|9)
dog_gen	synthesis	9832.2	0.948 3|1	2.342 (5|3)	0.709 (4|2)	0.887 3|1	2.482 (5|3)	0.596 (6|3)	0.803 3|1	2.492 (5|3)	0.437 (10|3)
stoned	string	8115.8	0.62 (11|4)	5.611 (22|8)	0.543 (11|4)	0.613 (10|4)	5.6 (22|8)	0.524 (10|4)	0.588 (9|4)	5.499 (22|8)	0.482 (8|4)
gflownet	graph	10000	0.54 (14|5)	5.025 (21|8)	0.493 (13|5)	0.499 (12|5)	4.638 (21|8)	0.44 (13|5)	0.438 (13|6)	4.808 (21|8)	0.367 (12|5)
reinvent_selfies	string	6067.4	0.838 (5|3)	3.75 (16|7)	0.671 (6|3)	0.821 (4|3)	3.921 (17|7)	0.632 (4|3)	0.782 (5|3)	3.974 (17|7)	0.567 (2|2)
graph_mcts	graph	10000	0.178 (25|9)	4.237 (18|5)	0.145 (25|9)	0.134 (25|9)	3.956 (18|5)	0.111 (25|9)	0.084 (25|10)	4.006 (18|5)	0.066 (26|10)
dst	graph	10000	0.79 (8|2)	6.876 (25|10)	0.601 (7|1)	0.781 (7|2)	7.002 (25|10)	0.557 (8|2)	0.749 (7|2)	6.672 (25|10)	0.49 (5|2)
selfies_ga	string	10000	0.392 (23|9)	7.372 (26|10)	0.235 (23|9)	0.378 (18|6)	7.357 (26|10)	0.22 (21|7)	0.357 (15|6)	7.302 (26|10)	0.195 (18|6)
gflownet_al	graph	10000	0.464 (16|7)	4.31 (19|6)	0.433 (15|6)	0.403 (16|7)	4.34 (19|6)	0.363 (14|6)	0.324 (16|7)	4.527 (20|7)	0.272 (15|7)
screening	N/A	10000	0.456 (17|2)	2.99 (11|2)	0.363 (18|2)	0.309 (19|2)	2.637 (8|2)	0.239 (19|2)	0.168 (20|2)	2.678 (7|2)	0.127 (20|2)
mol_pal	N/A	10000	0.608 (12|1)	2.114 (2|1)	0.458 (14|1)	0.405 (15|1)	2.477 (4|1)	0.339 (16|1)	0.234 (18|1)	2.463 (4|1)	0.2 (17|1)
graph_ga	graph	10000	0.818 (6|1)	3.172 (13|2)	0.598 (8|2)	0.813 (6|1)	3.323 (13|2)	0.554 (9|3)	0.797 (4|1)	3.351 (11|2)	0.489 (7|3)
Ours	synthesis	10000	0.85 (4|2)	**1.771 (1|1)**	0.698 (5|3)	0.818 (5|2)	2.234 (2|2)	0.639 (3|1)	0.755 (6|2)	**2.156 (1|1)**	0.536 (3|1)

		DRD2									
			Top 1			Top 10			Top 100		
	category	Oracle Calls	Score	SA	AUC	Score	SA	AUC	Score	SA	AUC
synnet	synthesis	6472.6	**1.0 (1|1)**	2.58 (7|3)	0.985 (4|3)	0.998 (9|3)	2.806 (8|4)	**0.969 (1|1)**	0.986 (12|3)	2.845 (7|4)	0.898 (4|2)
pasithea	string	2983.2	0.592 (24|10)	4.057 (22|7)	0.559 (23|9)	0.275 (25|10)	3.894 (22|7)	0.256 (25|10)	0.065 (25|10)	3.858 (19|7)	0.06 (25|10)
dog_ae	synthesis	1337	0.999 (11|4)	2.444 (5|2)	0.986 (3|2)	0.979 (15|4)	2.495 (4|2)	0.944 (6|4)	0.589 (16|4)	2.641 (5|3)	0.543 (14|4)
smiles_vae_bo	string	10000	0.899 (20|8)	2.282 (3|2)	0.82 (17|7)	0.774 (18|7)	2.948 (9|3)	0.555 (19|8)	0.319 (19|8)	3.388 (12|5)	0.187 (19|8)
jt_vae_bo	graph	4099.2	0.778 (23|8)	3.226 (14|4)	0.742 (21|7)	0.558 (23|8)	3.585 (18|6)	0.507 (21|7)	0.197 (23|8)	3.84 (18|6)	0.17 (22|7)
moldqn	graph	10000	0.049 (26|10)	4.346 (23|10)	0.03 (26|10)	0.033 (26|10)	5.489 (24|10)	0.025 (26|10)	0.024 (26|10)	6.036 (25|10)	0.019 (26|10)
mars	graph	7840.6	0.995 (13|4)	4.019 (21|9)	0.939 (10|3)	0.989 (13|5)	3.778 (21|9)	0.892 (12|3)	0.96 (14|5)	3.859 (20|7)	0.753 (10|3)
selfies_lstm_hc	string	10000	**1.0 (1|1)**	3.351 (16|5)	0.848 (16|6)	**1.0 (1|1)**	3.262 (13|5)	0.729 (16|6)	0.993 (10|5)	3.335 (9|4)	0.51 (15|6)
gp_bo	graph	10000	**1.0 (1|1)**	2.992 (11|2)	0.958 (8|2)	0.999 (8|2)	3.201 (11|3)	0.924 (8|2)	0.998 (7|2)	3.372 (11|3)	0.871 (8|2)
smiles_ga	string	4286.4	0.987 (15|6)	6.56 (26|10)	0.931 (12|5)	0.987 (14|6)	6.737 (26|10)	0.908 (11|5)	0.987 (11|6)	6.751 (26|10)	0.876 (7|4)
mimosa	graph	9879.4	0.994 (14|5)	3.315 (15|5)	0.88 (15|5)	0.991 (12|4)	3.54 (17|5)	0.8 (14|5)	0.981 (13|4)	3.66 (17|5)	0.71 (13|5)
reinvent	string	6760.2	**1.0 (1|1)**	2.408 (4|3)	0.968 (7|2)	**1.0 (1|1)**	2.419 (3|2)	0.946 (5|1)	**1.0 (1|1)**	2.551 (3|2)	0.909 (2|1)
smiles_lstm_hc	string	10000	**1.0 (1|1)**	**2.24 (1|1)**	0.957 (9|3)	**1.0 (1|1)**	2.295 (2|1)	0.919 (9|3)	**1.0 (1|1)**	**2.308 (1|1)**	0.788 (9|5)
selfies_vae_bo	string	10000	0.941 (19|7)	2.63 (8|4)	0.808 (18|8)	0.772 (20|9)	3.251 (12|4)	0.569 (18|7)	0.293 (21|9)	3.658 (16|6)	0.175 (21|9)
dog_gen	synthesis	10000	**1.0 (1|1)**	2.244 2|1	**0.995 (1|1)**	**1.0 (1|1)**	**2.271 (1|1)**	0.949 (4|3)	**1.0 (1|1)**	2.31 2|1	0.74 (11|3)
stoned	string	8182.4	0.997 (12|5)	4.973 (24|8)	0.935 (11|4)	0.997 (11|5)	5.369 (23|8)	0.914 (10|4)	0.997 (8|4)	5.614 (23|8)	0.881 (6|3)
gflownet	graph	10000	0.951 (17|6)	3.664 (20|8)	0.792 (20|6)	0.836 (17|6)	3.485 (16|4)	0.591 (17|6)	0.493 (17|6)	4.032 (21|8)	0.279 (18|6)
reinvent_selfies	string	5523.8	**1.0 (1|1)**	3.485 (19|6)	0.979 (6|1)	**1.0 (1|1)**	3.358 (15|6)	0.943 (7|2)	**1.0 (1|1)**	3.307 (8|3)	0.898 (4|2)
graph_mcts	graph	10000	0.587 (25|9)	3.458 (18|7)	0.477 (24|9)	0.402 (24|9)	3.707 (20|8)	0.3 (24|9)	0.18 (24|9)	3.51 (15|4)	0.121 (24|9)
dst	graph	8121	**1.0 (1|1)**	3.024 (12|3)	0.887 (14|4)	0.998 (9|3)	3.12 (10|2)	0.821 (13|4)	0.994 (9|3)	3.351 (10|2)	0.739 (12|4)
selfies_ga	string	10000	0.837 (22|9)	5.477 (25|9)	0.426 (25|10)	0.773 (19|8)	5.561 (25|9)	0.383 (23|9)	0.679 (15|7)	5.736 (24|9)	0.314 (17|7)
gflownet_al	graph	10000	0.864 (21|7)	3.368 (17|6)	0.717 (22|8)	0.637 (22|7)	3.631 (19|7)	0.469 (22|8)	0.254 (22|7)	4.268 (22|9)	0.166 (23|8)
screening	N/A	10000	0.95 (18|2)	3.047 (13|2)	0.798 (19|2)	0.741 (21|2)	3.289 (14|2)	0.545 (20|2)	0.308 (20|2)	3.397 (13|1)	0.187 (19|2)
mol_pal	N/A	10000	0.965 (16|1)	2.462 (6|1)	0.903 (13|1)	0.873 (16|1)	2.734 (6|1)	0.783 (15|1)	0.477 (18|1)	3.473 (14|2)	0.403 (16|1)
graph_ga	graph	8326.4	**1.0 (1|1)**	2.758 (10|1)	0.991 (2|1)	**1.0 (1|1)**	2.786 (7|1)	0.964 (2|1)	**1.0 (1|1)**	2.815 (6|1)	**0.924 (1|1)**
Ours	synthesis	9068	**1.0 (1|1)**	2.663 (9|4)	0.981 (5|4)	**1.0 (1|1)**	2.565 (5|3)	0.96 (3|2)	**1.0 (1|1)**	2.578 (4|2)	0.901 (3|1)

Tables 8, 9, 10 and 11 are comprehensive results against baselines taxonomized in (Gao et al., 2022). We evaluate the average score of the Top K molecules, their average synthetic accessibility (Ertl & Schuffenhauer, 2009) and top K AUC (AUC of no. oracle calls vs score plot), for K=1,10,100. Like

Table 10: Guacamol multi-objective Oracles for properties of known drugs: Osimertinib, Fexofenadine, Ranolazine.

		Osimertinib MPO									
			Top 1			Top 10			Top 100		
	category	Oracle Calls	Score	SA	AUC	Score	SA	AUC	Score	SA	AUC
synnet	synthesis	6329.4	0.821 (12|3)	3.122 (9|4)	0.814 (8|2)	0.811 (13|3)	3.005 (8|4)	0.797 (8|2)	0.789 (13|3)	3.036 (8|4)	0.761 (7|2)
pasithea	string	1715.2	0.793 (22|9)	2.815 (7|3)	0.791 (21|9)	0.757 (24|10)	3.01 (9|3)	0.752 (22|9)	0.662 (25|10)	3.347 (11|5)	0.645 (24|9)
dog_ae	synthesis	1257.8	0.793 (22|4)	2.819 (8|3)	0.79 (22|4)	0.759 (23|4)	2.824 (6|3)	0.751 (23|4)	0.688 (24|4)	2.882 (4|3)	0.662 (22|4)
smiles_vae_bo	string	10000	0.802 (20|8)	2.811 (6|2)	0.795 (19|7)	0.784 (18|7)	2.812 (5|2)	0.773 (19|7)	0.753 (19|8)	2.958 (5|2)	0.714 (18|7)
jt_vae_bo	graph	5121	0.803 (18|8)	3.854 (16|4)	0.799 (18|8)	0.784 (18|8)	3.285 (12|1)	0.776 (17|8)	0.752 (20|8)	3.457 (12|1)	0.728 (17|8)
moldqn	graph	10000	0.7 (26|10)	5.762 (25|10)	0.692 (25|10)	0.686 (26|10)	5.752 (24|10)	0.677 (25|10)	0.651 (26|10)	5.55 (24|10)	0.638 (26|10)
mars	graph	4046.6	0.809 (17|7)	4.871 (22|9)	0.8 (17|7)	0.797 (16|7)	4.932 (22|9)	0.779 (16|7)	0.776 (16|7)	5.264 (22|9)	0.732 (16|7)
selfies_lstm_hc	string	10000	0.832 (10|6)	3.366 (10|4)	0.805 (12|6)	0.822 (10|6)	3.151 (11|5)	0.783 (14|6)	0.804 (11|6)	3.206 (9|3)	0.733 (15|6)
gp_bo	graph	10000	0.838 (8|2)	4.296 (20|7)	0.806 (11|3)	0.828 (9|2)	4.137 (17|4)	0.789 (11|4)	0.813 (9|2)	4.273 (18|5)	0.751 (11|5)
smiles_ga	string	4815.2	0.835 (9|5)	6.086 (26|10)	0.827 (5|4)	0.835 (8|5)	6.01 (26|10)	0.82 (5|4)	0.834 (5|4)	6.265 (26|10)	0.8 (4|3)
mimosa	graph	10000	0.817 (13|4)	4.09 (18|5)	0.804 (14|5)	0.814 (12|4)	4.223 (19|6)	0.791 (9|2)	0.805 (10|3)	4.104 (16|3)	0.752 (10|4)
reinvent	string	10000	**0.909 (1|1)**	3.552 (11|5)	**0.852 (1|1)**	**0.905 (1|1)**	3.494 (13|6)	**0.839 (1|1)**	**0.897 (1|1)**	3.486 (13|6)	**0.808 (1|1)**
smiles_lstm_hc	string	10000	0.859 (4|3)	2.712 (4|1)	0.819 (7|5)	0.848 (4|3)	2.773 (4|1)	0.799 (7|5)	0.829 (7|5)	2.785 (3|1)	0.751 (11|5)
selfies_vae_bo	string	10000	0.803 (18|7)	3.715 (13|6)	0.794 (20|8)	0.781 (21|8)	3.05 (10|4)	0.768 (20|8)	0.75 (22|9)	3.333 (10|4)	0.713 (19|8)
dog_gen	synthesis	10000	0.851 (6|2)	2.728 (5|2)	0.804 (14|3)	0.843 (7|2)	2.658 (3|2)	0.776 (17|3)	0.827 (8|2)	2.714 (2|2)	0.707 (20|3)
stoned	string	9326.2	0.848 (7|4)	5.619 (24|9)	0.831 (4|3)	0.848 (4|3)	5.871 (25|9)	0.824 (3|2)	0.847 (4|3)	6.093 (25|9)	0.801 (2|2)
gflownet	graph	10000	0.817 (13|4)	3.819 (15|3)	0.805 (12|4)	0.798 (15|6)	4.221 (18|5)	0.787 (12|5)	0.779 (15|6)	4.441 (19|6)	0.76 (9|3)
reinvent_selfies	string	9951.4	0.879 (3|2)	3.938 (17|7)	0.835 (3|2)	0.874 (2|2)	3.926 (16|7)	0.822 (4|3)	0.866 (2|2)	3.98 (15|7)	0.792 (5|4)
graph_mcts	graph	10000	0.738 (25|9)	3.622 (12|1)	0.721 (24|9)	0.723 (25|9)	3.656 (14|2)	0.702 (24|9)	0.691 (23|9)	4.116 (17|4)	0.656 (23|9)
dst	graph	9173.2	0.827 (11|3)	4.338 (21|8)	0.803 (16|6)	0.818 (11|3)	4.336 (20|7)	0.787 (12|5)	0.803 (12|4)	4.513 (20|7)	0.744 (13|6)
selfies_ga	string	10000	0.785 (24|10)	5.515 (23|8)	0.689 (26|10)	0.778 (22|9)	5.401 (23|8)	0.672 (26|10)	0.769 (18|7)	5.531 (23|8)	0.645 (24|9)
gflownet_al	graph	10000	0.812 (16|6)	4.109 (19|6)	0.807 (9|2)	0.8 (14|5)	4.501 (21|8)	0.79 (10|3)	0.78 (14|5)	4.54 (21|8)	0.761 (7|2)
screening	N/A	10000	0.802 (20|2)	2.421 (3|2)	0.788 (23|2)	0.783 (20|2)	2.9 (7|2)	0.766 (21|2)	0.751 (21|2)	2.988 (7|2)	0.705 (21|2)
mol_pal	N/A	10000	0.816 (15|1)	**2.134 (1|1)**	0.807 (9|1)	0.794 (17|1)	2.551 (2|1)	0.78 (15|1)	0.77 (17|1)	2.981 (6|1)	0.737 (14|1)
graph_ga	graph	10000	0.881 (2|1)	3.78 (14|2)	0.849 (2|1)	0.873 (3|1)	3.88 (15|3)	0.834 (2|1)	0.861 (3|1)	3.931 (14|2)	0.801 (2|1)
Ours	synthesis	10000	0.859 (4|1)	2.263 (2|1)	0.826 (6|1)	0.847 (6|1)	**2.21 (1|1)**	0.81 (6|1)	0.832 (6|1)	**2.249 (1|1)**	0.769 (6|1)

		Fexofenadine MPO									
			Top 1			Top 10			Top 100		
	category	Oracle Calls	Score	SA	AUC	Score	SA	AUC	Score	SA	AUC
synnet	synthesis	6763.2	0.798 (9|3)	3.423 (9|4)	0.782 (4|2)	0.786 (8|2)	3.337 (8|4)	0.764 (4|2)	0.749 (10|2)	3.29 (8|4)	0.722 (5|2)
pasithea	string	5605.6	0.708 (21|9)	2.707 (2|1)	0.705 (19|7)	0.666 (24|10)	3.385 (9|3)	0.662 (23|9)	0.597 (24|10)	3.865 (9|3)	0.585 (23|9)
dog_ae	synthesis	1247.6	0.724 (18|4)	3.085 (5|2)	0.719 (14|4)	0.687 (20|4)	2.922 (3|2)	0.681 (18|4)	0.634 (22|4)	2.977 (3|2)	0.619 (19|4)
smiles_vae_bo	string	10000	0.72 (19|8)	3.633 (10|3)	0.702 (20|8)	0.692 (19|8)	3.034 (4|1)	0.673 (19|7)	0.65 (19|8)	2.998 (4|1)	0.618 (20|8)
jt_vae_bo	graph	4385.2	0.703 (24|8)	3.314 (6|1)	0.699 (22|8)	0.675 (23|8)	3.892 (10|1)	0.67 (21|8)	0.633 (23|8)	3.903 (10|1)	0.618 (20|8)
moldqn	graph	10000	0.533 (26|10)	5.844 (23|9)	0.5 (26|10)	0.516 (26|10)	5.717 (23|9)	0.479 (26|10)	0.482 (26|10)	5.524 (22|8)	0.432 (26|10)
mars	graph	6559	0.756 (13|4)	5.337 (20|8)	0.732 (12|4)	0.741 (13|4)	5.319 (19|7)	0.713 (11|4)	0.717 (14|5)	5.698 (23|9)	0.671 (12|5)
selfies_lstm_hc	string	10000	0.769 (12|6)	5.692 (21|7)	0.719 (14|6)	0.754 (12|6)	5.322 (20|7)	0.696 (14|6)	0.727 (12|6)	5.226 (20|8)	0.652 (14|6)
gp_bo	graph	10000	0.805 (8|2)	6.402 (25|10)	0.743 (9|3)	0.794 (6|2)	6.051 (24|10)	0.724 (9|3)	0.774 (6|2)	5.89 (24|10)	0.687 (9|3)
smiles_ga	string	8784.8	0.771 (11|5)	4.631 (15|6)	0.733 (10|5)	0.765 (11|5)	4.909 (17|6)	0.723 (10|5)	0.757 (8|5)	5.141 (18|6)	0.703 (7|4)
mimosa	graph	10000	0.744 (14|5)	4.257 (13|2)	0.724 (13|5)	0.738 (14|5)	4.452 (13|2)	0.709 (12|5)	0.724 (13|4)	4.39 (13|2)	0.674 (11|4)
reinvent	string	10000	**0.91 (1|1)**	4.234 (11|4)	0.804 (3|2)	**0.903 (1|1)**	4.271 (11|4)	0.787 (3|2)	**0.892 (1|1)**	4.261 (12|5)	0.754 (2|2)
smiles_lstm_hc	string	10000	0.818 (6|4)	2.896 (4|2)	0.757 (7|4)	0.794 (6|4)	3.1 (5|2)	0.727 (7|4)	0.764 (7|4)	3.166 (6|2)	0.682 (10|5)
selfies_vae_bo	string	10000	0.707 (22|10)	4.239 (12|5)	0.701 (21|9)	0.683 (22|9)	4.347 (12|5)	0.672 (20|8)	0.644 (21|9)	3.977 (11|4)	0.621 (18|7)
dog_gen	synthesis	10000	0.809 (7|2)	2.707 2|1	0.733 (10|3)	0.769 (9|3)	**2.687 (1|1)**	0.697 (13|3)	0.736 (11|3)	**2.669 (1|1)**	0.642 (16|3)
stoned	string	9898.2	0.851 (2|2)	7.051 (26|10)	0.806 (2|1)	0.851 (2|2)	7.171 (26|10)	**0.8 (1|1)**	0.848 (2|2)	7.17 (26|10)	**0.779 (1|1)**
gflownet	graph	10000	0.727 (17|7)	4.349 (14|3)	0.717 (16|6)	0.712 (16|6)	4.464 (14|3)	0.696 (14|6)	0.678 (16|6)	4.637 (15|4)	0.656 (13|6)
reinvent_selfies	string	10000	0.843 (5|3)	5.721 (22|8)	0.765 (6|3)	0.835 (3|3)	5.437 (22|8)	0.743 (6|3)	0.819 (3|3)	5.186 (19|7)	0.706 (6|3)
graph_mcts	graph	10000	0.612 (25|9)	4.672 (16|4)	0.597 (25|9)	0.594 (25|9)	4.523 (15|4)	0.575 (25|9)	0.562 (25|9)	4.42 (14|3)	0.523 (25|9)
dst	graph	10000	0.778 (10|3)	5.056 (19|7)	0.745 (8|2)	0.767 (10|3)	5.349 (21|8)	0.727 (7|2)	0.753 (9|3)	5.458 (21|7)	0.692 (8|2)
selfies_ga	string	10000	0.737 (15|7)	6.111 (24|9)	0.608 (24|10)	0.73 (15|7)	6.359 (25|9)	0.587 (24|10)	0.717 (14|7)	6.455 (25|9)	0.554 (24|10)
gflownet_al	graph	10000	0.733 (16|6)	4.721 (17|5)	0.717 (16|6)	0.706 (17|7)	4.681 (16|5)	0.691 (16|7)	0.673 (17|7)	4.717 (16|5)	0.647 (15|7)
screening	N/A	10000	0.707 (22|2)	3.331 (7|2)	0.693 (23|2)	0.686 (21|2)	3.147 (6|2)	0.668 (22|2)	0.649 (20|2)	3.026 (5|2)	0.615 (22|2)
mol_pal	N/A	10000	0.71 (20|1)	**2.08 (1|1)**	0.706 (18|1)	0.696 (18|1)	2.806 (2|1)	0.686 (17|1)	0.665 (18|1)	2.973 (2|1)	0.64 (17|1)
graph_ga	graph	10000	0.846 (4|1)	4.852 (18|6)	0.777 (5|1)	0.831 (4|1)	4.926 (18|6)	0.763 (5|1)	0.818 (4|1)	4.896 (17|6)	0.733 (4|1)
Ours	synthesis	8914	0.849 (3|1)	3.417 (8|3)	**0.816 (1|1)**	0.827 (5|1)	3.291 (7|3)	0.791 (2|1)	0.806 (5|1)	3.188 (7|3)	0.75 (3|1)

		Ranolazine MPO									
			Top 1			Top 10			Top 100		
	category	Oracle Calls	Score	SA	AUC	Score	SA	AUC	Score	SA	AUC
synnet	synthesis	7443.8	0.783 (10|3)	3.355 (10|3)	0.765 (6|2)	0.771 (10|3)	3.628 (12|4)	0.743 (4|1)	0.745 (12|3)	3.772 (13|4)	0.691 (5|1)
pasithea	string	1002.8	0.443 (24|10)	2.897 (3|2)	0.438 (24|10)	0.354 (24|10)	2.945 (2|2)	0.348 (24|10)	0.219 (24|10)	**2.67 (1|1)**	0.212 (24|10)
dog_ae	synthesis	1163.4	0.745 (16|4)	3.725 (12|4)	0.737 (11|4)	0.701 (16|4)	3.505 (9|3)	0.69 (12|4)	0.589 (18|4)	3.307 (8|3)	0.567 (15|4)
smiles_vae_bo	string	10000	0.599 (19|8)	3.144 (5|4)	0.564 (20|8)	0.524 (20|8)	2.998 (5|3)	0.458 (20|8)	0.393 (20|8)	2.734 (2|2)	0.319 (21|8)
jt_vae_bo	graph	4350.6	0.588 (20|8)	3.948 (14|2)	0.583 (18|8)	0.525 (19|8)	3.671 (13|1)	0.509 (19|8)	0.363 (21|8)	3.439 (10|1)	0.337 (20|8)
moldqn	graph	10000	0.171 (26|10)	4.846 (22|9)	0.084 (26|10)	0.104 (26|10)	5.138 (23|10)	0.052 (26|10)	0.036 (26|10)	5.221 (23|10)	0.018 (26|10)
mars	graph	2681.4	0.776 (12|3)	4.661 (21|8)	0.766 (5|1)	0.76 (14|4)	4.661 (21|8)	0.742 (5|1)	0.72 (15|5)	4.776 (22|9)	0.685 (7|2)
selfies_lstm_hc	string	10000	0.795 (9|5)	4.12 (16|7)	0.679 (15|6)	0.77 (12|7)	4.545 (19|7)	0.615 (17|6)	0.725 (14|7)	4.601 (19|7)	0.502 (18|7)
gp_bo	graph	10000	0.818 (6|1)	4.142 (17|4)	0.763 (7|2)	0.807 (5|1)	4.414 (18|6)	0.736 (7|2)	0.79 (5|1)	4.453 (17|5)	0.695 (4|1)
smiles_ga	string	10000	0.78 (11|6)	5.895 (24|8)	0.721 (12|5)	0.775 (9|5)	5.904 (24|8)	0.699 (11|5)	0.766 (9|5)	5.997 (24|8)	0.671 (8|4)
mimosa	graph	10000	0.773 (14|4)	3.982 (15|3)	0.674 (16|6)	0.768 (13|3)	3.994 (15|3)	0.641 (14|5)	0.758 (11|3)	4.162 (15|3)	0.588 (13|5)
reinvent	string	10000	**0.865 (1|1)**	2.815 (2|1)	**0.789 (1|1)**	0.858 (2|2)	**2.87 (1|1)**	0.761 (2|2)	0.849 (2|2)	2.885 (5|3)	0.72 (2|2)
smiles_lstm_hc	string	10000	0.824 (4|4)	2.938 (4|3)	0.758 (10|4)	0.807 (5|4)	3.195 (7|4)	0.715 (9|4)	0.784 (6|4)	3.323 (9|4)	0.631 (10|5)
selfies_vae_bo	string	10000	0.565 (21|9)	3.154 (6|5)	0.535 (21|9)	0.489 (22|9)	3.584 (10|5)	0.453 (22|9)	0.363 (21|9)	3.527 (11|5)	0.314 (22|9)
dog_gen	synthesis	10000	0.824 (4|1)	3.183 (7|1)	0.761 (8|3)	0.808 (4|1)	3.072 (6|1)	0.712 (10|3)	0.782 (7|2)	3.17 (6|1)	0.602 (12|3)
stoned	string	10000	0.863 (2|2)	6.895 (26|10)	0.785 (2|2)	**0.86 (1|1)**	6.892 (26|10)	**0.766 (1|1)**	**0.855 (1|1)**	6.93 (26|10)	**0.739 (1|1)**
gflownet	graph	9429.6	0.702 (18|7)	4.881 (23|10)	0.681 (13|4)	0.675 (17|6)	4.773 (22|9)	0.653 (13|4)	0.649 (16|6)	4.734 (21|8)	0.615 (11|4)
reinvent_selfies	string	10000	0.851 (3|3)	3.786 (13|6)	0.779 (3|3)	0.846 (3|3)	3.586 (11|6)	0.748 (3|3)	0.836 (3|3)	3.641 (12|6)	0.696 (3|3)
graph_mcts	graph	10000	0.37 (25|9)	3.629 (11|1)	0.317 (25|9)	0.304 (25|9)	3.804 (14|2)	0.239 (25|9)	0.176 (25|9)	3.904 (14|2)	0.122 (25|9)
dst	graph	9505.8	0.752 (15|5)	4.318 (19|6)	0.659 (17|7)	0.746 (15|5)	4.379 (17|5)	0.633 (15|6)	0.73 (13|4)	4.653 (20|7)	0.58 (14|6)
selfies_ga	string	10000	0.776 (12|7)	6.278 (25|9)	0.576 (19|7)	0.771 (10|6)	6.103 (25|9)	0.556 (18|7)	0.763 (10|6)	6.197 (25|9)	0.525 (17|6)
gflownet_al	graph	10000	0.705 (17|6)	4.471 (20|7)	0.681 (13|4)	0.666 (18|7)	4.594 (20|7)	0.633 (15|6)	0.617 (17|7)	4.597 (18|6)	0.543 (16|7)
screening	N/A	10000	0.533 (23|2)	**2.744 (1|1)**	0.486 (23|2)	0.456 (23|2)	2.987 (3|1)	0.412 (23|2)	0.358 (23|2)	2.778 (3|1)	0.302 (23|2)
mol_pal	N/A	10000	0.556 (22|1)	3.287 (9|2)	0.517 (22|1)	0.495 (21|1)	2.991 (4|2)	0.458 (20|1)	0.397 (19|1)	2.88 (4|2)	0.357 (19|1)
graph_ga	graph	10000	0.811 (7|2)	4.242 (18|5)	0.76 (9|3)	0.801 (8|2)	4.278 (16|4)	0.729 (8|3)	0.782 (7|2)	4.401 (16|4)	0.671 (8|3)
Ours	synthesis	10000	0.808 (8|2)	3.201 (8|2)	0.774 (4|1)	0.805 (7|2)	3.205 (8|2)	0.741 (6|2)	0.794 (4|1)	3.254 (7|2)	0.686 (6|2)

Table 11: Guacamol multi-objective Oracles for properties of known drugs: Perindopril, Amlodipine, Sitagliptin, and Zaleplon. Disclaimer: We used version 1.0.0. PMO Gao et al. (2022), where we adopted the numbers from, used 0.3.6. We'd like to mention there were changes since 0.3.6 that may affect the Sitagliptin and Zaleplon evaluations.

		Perindopril MPO									
			Top 1			Top 10			Top 100		
	category	Oracle Calls	Score	SA	AUC	Score	SA	AUC	Score	SA	AUC
synnet	synthesis	8018.6	0.61 (4|2)	3.775 (14|4)	**0.582 (1|1)**	0.589 (5|2)	3.717 (13|4)	**0.559 (1|1)**	0.547 (5|2)	3.749 (13|4)	**0.514 (1|1)**
pasithea	string	1001	0.448 (23|9)	3.382 (8|3)	0.447 (23|9)	0.425 (23|9)	3.023 (6|2)	0.423 (22|9)	0.37 (23|9)	3.077 (6|2)	0.365 (23|9)
dog_ae	synthesis	1339	0.464 (20|4)	3.073 (3|1)	0.461 (17|4)	0.438 (21|4)	2.986 (3|1)	0.433 (18|4)	0.385 (22|4)	**2.781 (1|1)**	0.375 (22|4)
smiles_vae_bo	string	10000	0.484 (16|6)	2.888 (2|1)	0.472 (16|6)	0.459 (16|6)	**2.971 (1|1)**	0.444 (17|7)	0.423 (18|7)	2.967 (4|1)	0.399 (16|7)
jt_vae_bo	graph	5224.4	0.463 (21|8)	3.547 (11|2)	0.456 (20|7)	0.439 (20|7)	3.325 (10|2)	0.432 (19|6)	0.404 (21|8)	3.361 (12|2)	0.391 (18|6)
moldqn	graph	10000	0.283 (26|10)	4.468 (21|7)	0.248 (25|10)	0.254 (26|10)	5.334 (25|10)	0.214 (25|10)	0.162 (26|10)	5.355 (25|10)	0.125 (26|10)
mars	graph	6346.8	0.489 (15|5)	4.906 (24|9)	0.478 (14|5)	0.48 (13|5)	5.016 (24|9)	0.464 (12|4)	0.463 (12|4)	5.271 (24|9)	0.433 (11|4)
selfies_lstm_hc	string	10000	0.522 (10|4)	3.542 (10|4)	0.474 (15|5)	0.502 (11|5)	3.206 (9|4)	0.45 (14|5)	0.469 (11|5)	3.161 (9|4)	0.401 (15|6)
gp_bo	graph	10000	0.562 (8|2)	4.354 (19|6)	0.514 (7|2)	0.549 (8|2)	3.993 (16|4)	0.495 (6|2)	0.524 (9|3)	3.916 (14|3)	0.462 (7|2)
smiles_ga	string	3487.2	0.459 (22|8)	3.586 (12|5)	0.455 (21|8)	0.457 (18|7)	4.267 (19|7)	0.449 (15|6)	0.454 (13|6)	4.462 (19|7)	0.438 (10|5)
mimosa	graph	10000	0.558 (9|3)	4.917 (25|10)	0.509 (8|3)	0.549 (8|2)	4.942 (23|8)	0.492 (7|3)	0.53 (8|2)	4.879 (23|8)	0.46 (8|3)
reinvent	string	9435.8	**0.644 (1|1)**	4.533 (22|9)	**0.555 (3|1)**	**0.642 (1|1)**	4.468 (20|8)	**0.539 (3|1)**	**0.636 (1|1)**	4.467 (20|8)	**0.513 (2|1)**
smiles_lstm_hc	string	10000	0.569 (7|3)	3.778 (15|6)	0.516 (6|3)	0.554 (7|3)	3.449 (12|5)	0.491 (8|3)	0.533 (7|3)	3.347 (11|5)	0.448 (9|4)
selfies_vae_bo	string	10000	0.482 (17|7)	3.352 (7|2)	0.46 (18|7)	0.445 (19|8)	3.039 (7|3)	0.431 (20|8)	0.407 (19|8)	3.09 (7|3)	0.384 (20|8)
dog_gen	synthesis	10000	0.588 (6|3)	3.249 (5|2)	0.507 (9|3)	0.576 (6|3)	3.001 (4|2)	0.475 (10|3)	0.547 (5|2)	2.949 (3|2)	0.423 (14|3)
stoned	string	10000	0.522 (10|4)	4.395 (20|8)	0.495 (11|4)	0.521 (10|4)	4.723 (22|9)	0.49 (9|4)	0.514 (10|4)	4.811 (22|9)	0.474 (6|3)
gflownet	graph	10000	0.478 (18|6)	4.195 (18|5)	0.457 (19|6)	0.458 (17|6)	4.211 (17|5)	0.431 (20|7)	0.424 (17|6)	4.278 (18|6)	0.385 (19|7)
reinvent_selfies	string	9077.4	0.61 (4|2)	3.811 (16|7)	0.536 (5|2)	0.61 (3|2)	3.906 (14|6)	0.518 (5|2)	0.609 (2|2)	3.921 (15|6)	0.489 (4|2)
graph_mcts	graph	10000	0.335 (25|9)	3.147 (4|1)	0.311 (24|9)	0.311 (25|9)	3.187 (8|1)	0.278 (24|9)	0.263 (25|9)	3.175 (10|1)	0.219 (24|9)
dst	graph	10000	0.503 (13|4)	3.765 (13|3)	0.49 (12|4)	0.481 (12|4)	3.944 (15|3)	0.464 (12|4)	0.453 (14|5)	4.262 (17|5)	0.427 (12|5)
selfies_ga	string	10000	0.338 (24|10)	5.876 (26|10)	0.187 (26|10)	0.325 (24|10)	5.53 (26|10)	0.173 (26|10)	0.294 (24|10)	5.433 (26|10)	0.155 (25|10)
gflownet_al	graph	10000	0.465 (19|7)	4.599 (23|8)	0.45 (22|8)	0.438 (21|8)	4.514 (21|7)	0.422 (23|8)	0.405 (20|7)	4.564 (21|7)	0.376 (21|8)
screening	N/A	10000	0.501 (14|2)	**2.853 (1|1)**	0.48 (13|2)	0.465 (15|2)	2.984 (2|1)	0.447 (16|2)	0.426 (16|2)	2.971 (5|2)	0.399 (16|2)
mol_pal	N/A	10000	0.504 (12|1)	3.433 (9|2)	0.497 (10|1)	0.48 (13|1)	3.018 (5|2)	0.469 (11|1)	0.44 (15|1)	2.898 (2|1)	0.424 (13|1)
graph_ga	graph	10000	0.625 (2|1)	4.169 (17|4)	0.561 (2|1)	0.614 (2|1)	4.23 (18|6)	0.54 (2|1)	0.592 (3|1)	4.187 (16|4)	0.504 (3|1)
Ours	synthesis	10000	0.622 (3|1)	3.338 (6|3)	0.547 (4|2)	0.591 (4|1)	3.378 (11|3)	0.524 (4|2)	0.558 (4|1)	3.137 (8|3)	0.485 (5|2)

		Amlodipine MPO									
			Top 1			Top 10			Top 100		
	category	Oracle Calls	Score	SA	AUC	Score	SA	AUC	Score	SA	AUC
synnet	synthesis	7665	0.597 (13|3)	2.877 (7|4)	0.583 (13|2)	0.585 (10|3)	2.864 (6|4)	0.567 (9|2)	0.559 (9|3)	2.808 (7|4)	0.535 (8|2)
pasithea	string	2345.6	0.585 (16|8)	2.852 (5|2)	0.585 (11|6)	0.509 (22|10)	2.569 (3|1)	0.507 (20|9)	0.45 (22|10)	2.655 (4|1)	0.444 (21|9)
dog_ae	synthesis	1319.4	0.539 (21|4)	2.716 (3|3)	0.536 (20|4)	0.513 (21|4)	2.579 (4|3)	0.509 (19|4)	0.469 (21|4)	2.547 (3|3)	0.458 (20|4)
smiles_vae_bo	string	10000	0.612 (11|5)	2.928 (10|3)	0.604 (9|5)	0.559 (15|7)	2.927 (9|3)	0.536 (14|6)	0.502 (16|8)	2.745 (6|2)	0.477 (15|7)
jt_vae_bo	graph	5053.6	0.585 (16|4)	2.852 (5|1)	0.585 (11|4)	0.526 (17|4)	3.005 (11|2)	0.521 (16|4)	0.484 (19|5)	3.202 (14|3)	0.47 (16|4)
moldqn	graph	10000	0.384 (26|10)	5.915 (26|10)	0.344 (26|10)	0.355 (26|10)	6.004 (26|10)	0.312 (26|10)	0.316 (26|10)	6.292 (26|10)	0.23 (26|10)
mars	graph	5914.8	0.546 (20|6)	4.172 (24|9)	0.525 (21|6)	0.526 (17|4)	3.821 (21|8)	0.506 (21|6)	0.496 (17|4)	4.096 (21|8)	0.466 (18|6)
selfies_lstm_hc	string	10000	0.6 (12|6)	3.141 (13|5)	0.572 (17|8)	0.58 (11|5)	2.894 (8|2)	0.534 (15|7)	0.554 (12|6)	2.865 (8|3)	0.487 (13|6)
gp_bo	graph	10000	0.682 (5|2)	3.498 (20|6)	0.609 (8|2)	0.664 (5|2)	3.449 (17|5)	0.585 (7|2)	0.639 (5|2)	3.523 (18|5)	0.54 (6|2)
smiles_ga	string	4340.4	0.57 (19|9)	3.165 (14|6)	0.567 (18|9)	0.564 (12|6)	4.331 (25|10)	0.551 (10|5)	0.559 (9|5)	4.548 (25|10)	0.536 (7|4)
mimosa	graph	10000	0.594 (14|3)	2.893 (8|2)	0.593 (10|3)	0.564 (12|3)	3.785 (20|7)	0.545 (11|3)	0.55 (13|3)	4.101 (22|9)	0.511 (11|3)
reinvent	string	9258.2	0.736 (3|2)	3.544 (21|9)	0.655 (2|1)	0.734 (2|1)	3.612 (18|7)	0.637 (2|1)	0.729 (2|1)	3.448 (17|7)	0.609 (2|1)
smiles_lstm_hc	string	10000	0.739 (2|1)	2.937 (11|4)	0.639 (3|2)	0.714 (3|2)	2.941 (10|4)	0.596 (6|4)	0.669 (4|3)	2.979 (10|4)	0.535 (8|5)
selfies_vae_bo	string	10000	0.593 (15|7)	2.806 (4|1)	0.583 (13|7)	0.532 (16|8)	3.058 (12|5)	0.518 (18|8)	0.49 (18|9)	3.093 (12|5)	0.466 (18|8)
dog_gen	synthesis	10000	0.621 (9|2)	**2.245 (1|1)**	0.557 (19|3)	0.606 (9|2)	**2.36 (1|1)**	0.537 (12|3)	0.583 (8|2)	**2.348 (1|1)**	0.49 (12|3)
stoned	string	7362.8	0.639 (8|4)	3.494 (19|8)	0.618 (7|4)	0.636 (6|4)	3.956 (23|8)	0.611 (3|2)	0.631 (6|4)	4.371 (23|8)	0.595 (3|2)
gflownet	graph	10000	0.483 (24|8)	3.166 (15|4)	0.469 (23|8)	0.466 (23|7)	3.192 (13|3)	0.445 (23|8)	0.44 (23|7)	3.201 (13|2)	0.399 (22|7)
reinvent_selfies	string	8645.4	0.706 (4|3)	3.39 (17|7)	0.628 (4|3)	0.7 (4|3)	3.365 (16|6)	0.608 (4|3)	0.684 (3|2)	3.399 (16|6)	0.575 (5|3)
graph_mcts	graph	10000	0.484 (23|7)	4.171 (23|8)	0.474 (22|7)	0.463 (24|8)	3.87 (22|9)	0.448 (22|7)	0.425 (24|8)	3.845 (20|7)	0.386 (23|8)
dst	graph	5753	0.583 (18|5)	3.051 (12|3)	0.576 (16|5)	0.525 (19|6)	2.797 (5|1)	0.519 (17|5)	0.483 (20|6)	2.88 (9|1)	0.47 (16|4)
selfies_ga	string	10000	0.528 (22|10)	4.189 (25|10)	0.421 (25|10)	0.525 (19|9)	4.228 (24|9)	0.401 (25|10)	0.513 (14|7)	4.52 (24|9)	0.366 (25|10)
gflownet_al	graph	10000	0.467 (25|9)	3.402 (18|5)	0.45 (24|9)	0.443 (25|9)	3.326 (15|4)	0.429 (24|9)	0.412 (25|9)	3.28 (15|4)	0.375 (24|9)
screening	N/A	10000	0.613 (10|2)	2.9 (9|1)	0.582 (15|2)	0.563 (14|2)	2.879 (7|1)	0.537 (12|2)	0.505 (15|2)	2.712 (5|1)	0.479 (14|2)
mol_pal	N/A	10000	0.652 (6|1)	**3.279 (16|2)**	0.623 (6|1)	0.614 (8|1)	3.205 (14|2)	0.584 (8|1)	0.555 (11|1)	3.043 (11|2)	0.515 (10|1)
graph_ga	graph	10000	**0.784 (1|1)**	3.656 (22|7)	**0.688 (1|1)**	**0.769 (1|1)**	3.721 (19|6)	**0.663 (1|1)**	**0.744 (1|1)**	3.757 (19|6)	**0.624 (1|1)**
Ours	synthesis	7235	0.648 (7|1)	2.634 (2|2)	0.627 (5|1)	0.632 (7|1)	2.489 (2|2)	0.608 (4|1)	0.617 (7|1)	2.377 (2|2)	0.58 (4|1)

		Sitagliptin MPO									
			Top 1			Top 10			Top 100		
	category	Oracle Calls	Score	SA	AUC	Score	SA	AUC	Score	SA	AUC
synnet	synthesis	3347.4	0.067 (22|3)	3.629 (6|4)	0.06 (20|3)	0.03 (22|3)	3.245 (3|3)	0.026 (18|3)	0.009 (22|3)	2.985 (3|3)	0.007 (18|3)
pasithea	string	9766	0.231 (12|8)	7.041 (24|8)	0.176 (9|6)	0.138 (13|8)	6.79 (24|8)	0.089 (10|6)	0.049 (12|7)	6.772 (24|8)	0.027 (10|6)
dog_ae	synthesis	1940	0.04 (24|4)	3.011 (3|2)	0.037 (23|4)	0.01 (24|4)	3.049 (2|2)	0.01 (23|4)	0.002 (24|4)	2.694 (2|2)	0.002 (23|4)
smiles_vae_bo	string	10000	0.114 (19|9)	4.594 (11|4)	0.088 (18|9)	0.034 (20|10)	3.764 (7|2)	0.024 (19|9)	0.01 (20|10)	3.293 (5|1)	0.007 (18|9)
jt_vae_bo	graph	5491.6	0.17 (16|6)	4.398 (9|1)	0.135 (14|6)	0.063 (16|6)	4.277 (10|1)	0.046 (16|6)	0.015 (17|6)	3.908 (9|1)	0.01 (17|6)
moldqn	graph	10000	0.016 (26|10)	5.94 (21|9)	0.01 (26|10)	0.006 (26|10)	5.641 (20|8)	0.003 (26|10)	0.001 (26|10)	5.318 (20|8)	0.001 (25|9)
mars	graph	10000	0.083 (20|7)	5.257 (19|7)	0.04 (22|7)	0.034 (20|7)	4.418 (12|3)	0.016 (22|7)	0.01 (20|7)	4.088 (12|3)	0.005 (22|7)
selfies_lstm_hc	string	10000	0.35 (7|5)	5.232 (18|6)	0.203 (8|5)	0.23 (8|5)	5.096 (19|6)	0.117 (8|5)	0.102 (8|5)	5.146 (19|6)	0.041 (9|5)
gp_bo	graph	10000	0.318 (8|2)	4.906 (15|4)	0.238 (7|2)	0.267 (7|2)	4.84 (16|6)	0.187 (7|2)	0.196 (7|2)	4.808 (16|6)	0.117 (7|2)
smiles_ga	string	8642.2	0.505 (3|2)	6.764 (23|7)	0.397 (3|2)	0.481 (3|2)	6.554 (23|7)	0.364 (3|2)	0.437 (3|2)	6.675 (23|7)	0.307 (3|2)
mimosa	graph	10000	0.21 (14|4)	5.511 (20|8)	0.136 (13|5)	0.179 (11|3)	5.705 (21|9)	0.103 (9|3)	0.101 (9|3)	5.522 (21|9)	0.053 (8|3)
reinvent	string	9966.8	**0.08 (21|10)**	4.017 (8|2)	0.055 (21|10)	0.035 (19|9)	**3.926 (8|3)**	0.022 (21|10)	0.011 (18|9)	3.424 (7|2)	0.007 (18|9)
smiles_lstm_hc	string	10000	0.263 (9|6)	3.777 (7|1)	0.129 (15|8)	0.187 (9|6)	3.724 (6|1)	0.066 (13|8)	0.088 (10|6)	3.913 (10|3)	0.021 (11|7)
selfies_vae_bo	string	10000	0.244 (11|7)	4.466 (10|3)	0.173 (10|7)	0.14 (12|7)	4.946 (17|5)	0.084 (11|7)	0.039 (13|8)	4.701 (15|4)	0.021 (11|7)
dog_gen	synthesis	10000	0.253 (10|2)	3.178 (4|3)	0.103 (16|2)	0.182 (10|2)	3.572 (5|4)	0.048 (15|2)	0.085 (11|2)	3.403 (6|4)	0.016 (14|2)
stoned	string	8761.4	0.527 (2|1)	7.651 (26|10)	0.407 (2|1)	0.518 (2|1)	7.587 (26|10)	0.393 (2|1)	0.483 (2|1)	7.243 (26|10)	**0.352 (1|1)**
gflownet	graph	10000	0.045 (23|8)	5.019 (16|5)	0.028 (24|8)	0.017 (23|8)	4.622 (13|4)	0.009 (24|8)	0.004 (23|8)	4.415 (14|5)	0.002 (23|8)
reinvent_selfies	string	8262.8	0.41 (5|4)	4.718 (12|5)	0.257 (6|4)	0.363 (5|4)	4.756 (15|4)	0.194 (6|4)	0.269 (6|4)	4.897 (17|5)	0.118 (6|4)
graph_mcts	graph	10000	0.211 (13|3)	5.225 (17|6)	0.139 (12|4)	0.106 (15|5)	4.963 (18|7)	0.056 (14|5)	0.026 (15|5)	4.953 (18|7)	0.013 (16|5)
dst	graph	9999.8	0.205 (15|5)	4.745 (13|2)	0.159 (11|3)	0.111 (14|4)	4.679 (14|5)	0.076 (12|4)	0.027 (14|4)	4.18 (13|4)	0.018 (13|4)
selfies_ga	string	10000	0.483 (4|3)	7.286 (25|9)	0.311 (5|3)	0.469 (4|3)	7.044 (25|9)	0.281 (5|3)	0.436 (4|3)	7.072 (25|9)	0.233 (5|3)
gflownet_al	graph	10000	0.028 (25|9)	4.777 (14|3)	0.02 (25|9)	0.009 (25|9)	4.33 (11|2)	0.006 (25|9)	0.002 (24|9)	4.026 (11|2)	0.001 (25|9)
screening	N/A	10000	0.143 (17|1)	3.414 (5|2)	0.077 (19|2)	0.04 (18|2)	3.545 (4|1)	0.023 (20|2)	0.011 (18|2)	3.241 (4|1)	0.006 (21|2)
mol_pal	N/A	10000	0.118 (18|2)	2.764 (2|1)	0.1 (17|1)	0.051 (17|1)	4.024 (9|2)	0.044 (17|1)	0.018 (16|1)	3.563 (8|2)	0.015 (15|1)
graph_ga	graph	10000	**0.689 (1|1)**	6.426 (22|10)	**0.492 (1|1)**	**0.658 (1|1)**	6.37 (22|10)	**0.433 (1|1)**	**0.578 (1|1)**	6.249 (22|10)	0.33 (2|1)
Ours	synthesis	10000	0.388 (6|1)	**2.309 (1|1)**	0.344 (4|1)	0.358 (6|1)	**2.517 (1|1)**	0.313 (4|1)	0.323 (5|1)	**2.549 (1|1)**	0.248 (4|1)

		Zaleplon MPO									
			Top 1			Top 10			Top 100		
	category	Oracle Calls	Score	SA	AUC	Score	SA	AUC	Score	SA	AUC
synnet	synthesis	8349	0.403 (6|2)	3.85 (7|4)	0.377 (3|2)	0.381 (7|2)	3.911 (7|4)	0.341 (4|2)	0.28 (9|2)	3.982 (8|4)	0.224 (7|2)
pasithea	string	10000	0.243 (20|9)	6.586 (24|8)	0.186 (19|9)	0.14 (19|9)	6.461 (24|8)	0.092 (19|9)	0.05 (18|9)	6.76 (24|8)	0.028 (18|9)
dog_ae	synthesis	1478.6	0.157 (22|4)	3.148 (3|2)	0.145 (21|4)	0.055 (24|4)	3.33 (2|2)	0.05 (22|4)	0.007 (24|4)	3.111 (2|2)	0.006 (24|4)
smiles_vae_bo	string	10000	0.14 (23|10)	4.068 (8|2)	0.094 (23|10)	0.072 (22|10)	4.077 (8|2)	0.04 (23|10)	0.013 (23|10)	3.757 (6|2)	0.007 (23|10)
jt_vae_bo	graph	5540.8	0.303 (14|3)	4.223 (13|1)	0.267 (12|3)	0.161 (18|6)	4.354 (12|2)	0.126 (17|6)	0.035 (19|6)	4.374 (11|1)	0.024 (19|6)
moldqn	graph	10000	0.043 (26|10)	6.008 (21|9)	0.026 (26|10)	0.018 (26|10)	6.274 (23|10)	0.011 (25|9)	0.005 (25|9)	6.245 (23|10)	0.003 (25|9)
mars	graph	2442.4	0.297 (15|4)	4.809 (16|4)	0.293 (10|2)	0.213 (16|5)	5.468 (20|8)	0.187 (13|3)	0.101 (15|5)	5.709 (20|8)	0.083 (14|5)
selfies_lstm_hc	string	10000	0.361 (10|7)	4.136 (9|3)	0.304 (9|6)	0.31 (11|7)	5.199 (18|6)	0.218 (10|6)	0.214 (13|7)	5.529 (19|6)	0.103 (12|7)
gp_bo	graph	10000	0.269 (19|6)	4.975 (18|6)	0.252 (16|5)	0.253 (15|4)	4.898 (15|4)	0.222 (9|2)	0.217 (12|3)	5.065 (17|6)	0.166 (9|2)
smiles_ga	string	5972.4	0.397 (7|4)	6.173 (23|7)	0.35 (6|3)	0.389 (6|4)	6.146 (22|7)	0.334 (5|2)	0.378 (5|3)	6.226 (22|7)	0.311 (3|2)
mimosa	graph	9892.6	0.288 (16|5)	6.052 (22|10)	0.205 (18|6)	0.274 (13|2)	6.003 (21|9)	0.172 (15|5)	0.239 (11|2)	6.194 (21|9)	0.133 (10|3)
reinvent	string	6099.6	0.478 (2|1)	4.146 (10|4)	0.383 (2|1)	0.476 (2|1)	4.188 (10|3)	0.358 (2|1)	0.464 (2|1)	4.073 (9|3)	0.325 (2|1)
smiles_lstm_hc	string	10000	0.414 (5|3)	3.621 (6|1)	0.287 (11|7)	0.39 (5|3)	3.615 (5|1)	0.207 (11|7)	0.331 (8|6)	3.707 (5|1)	0.111 (11|6)
selfies_vae_bo	string	10000	0.38 (8|5)	4.217 (12|6)	0.323 (8|5)	0.28 (12|8)	4.541 (13|5)	0.207 (11|7)	0.1 (16|8)	4.905 (14|5)	0.059 (16|8)
dog_gen	synthesis	9331.8	0.344 (12|3)	3.238 (4|3)	0.171 (20|3)	0.314 (10|3)	3.445 (3|3)	0.123 (18|3)	0.253 (10|3)	3.549 (3|3)	0.073 (15|3)
stoned	string	9305	0.374 (9|6)	6.982 (26|10)	0.334 (7|4)	0.374 (8|5)	7.02 (26|10)	0.326 (7|4)	0.369 (6|4)	7.065 (26|10)	0.308 (4|3)
gflownet	graph	9043.6	0.118 (24|8)	5.376 (19|7)	0.067 (24|8)	0.07 (23|8)	5.114 (17|6)	0.036 (24|8)	0.029 (20|7)	4.926 (15|4)	0.011 (21|8)
reinvent_selfies	string	4660.6	0.441 (3|2)	4.155 (11|5)	0.37 (4|2)	0.434 (3|2)	4.345 (11|4)	0.333 (6|3)	0.384 (4|2)	4.338 (10|4)	0.258 (6|4)
graph_mcts	graph	10000	0.167 (21|7)	4.887 (17|5)	0.113 (22|7)	0.097 (21|7)	5.002 (16|5)	0.059 (21|7)	0.029 (20|7)	5.064 (16|5)	0.015 (20|7)
dst	graph	10000	0.344 (12|2)	4.355 (15|3)	0.258 (15|4)	0.259 (14|3)	4.102 (9|1)	0.177 (14|4)	0.156 (14|4)	4.451 (12|2)	0.09 (13|4)
selfies_ga	string	10000	0.36 (11|8)	6.586 (24|8)	0.263 (13|8)	0.354 (9|6)	6.698 (25|9)	0.244 (8|5)	0.337 (7|5)	6.783 (25|9)	0.214 (8|5)
gflownet_al	graph	10000	0.048 (25|9)	4.276 (14|2)	0.029 (25|9)	0.02 (25|9)	4.706 (14|3)	0.01 (26|10)	0.005 (25|9)	4.725 (13|3)	0.002 (26|10)
screening	N/A	10000	0.281 (18|2)	3.612 (5|2)	0.223 (17|2)	0.124 (20|2)	3.819 (6|2)	0.073 (20|2)	0.02 (22|2)	3.688 (4|1)	0.011 (21|2)
mol_pal	N/A	10000	0.287 (17|1)	2.764 (2|1)	0.262 (14|1)	0.191 (17|1)	3.495 (4|1)	0.169 (16|1)	0.055 (17|1)	3.816 (7|2)	0.046 (17|1)
graph_ga	graph	10000	0.421 (4|1)	5.4 (20|8)	0.367 (5|1)	0.412 (4|1)	5.458 (19|7)	0.347 (3|1)	0.39 (3|1)	5.306 (18|7)	0.305 (5|1)
Ours	synthesis	10000	**0.577 (1|1)**	**2.503 (1|1)**	**0.547 (1|1)**	**0.573 (1|1)**	**2.412 (1|1)**	**0.528 (1|1)**	**0.545 (1|1)**	**2.365 (1|1)**	**0.472 (1|1)**

(Gao et al., 2022), we limit to 10000 Oracle calls, truncating and padding to 10000 if convergence occurs before 10000 calls. For each cell, numbers are followed by rankings. $X(R_1|R_2)$ means score X is ranked R_1-best amongst all methods for that column and R_2-best amongst in-category methods. We visualize the rankings in Figure 17 to facilitate easier interpretation of the results.

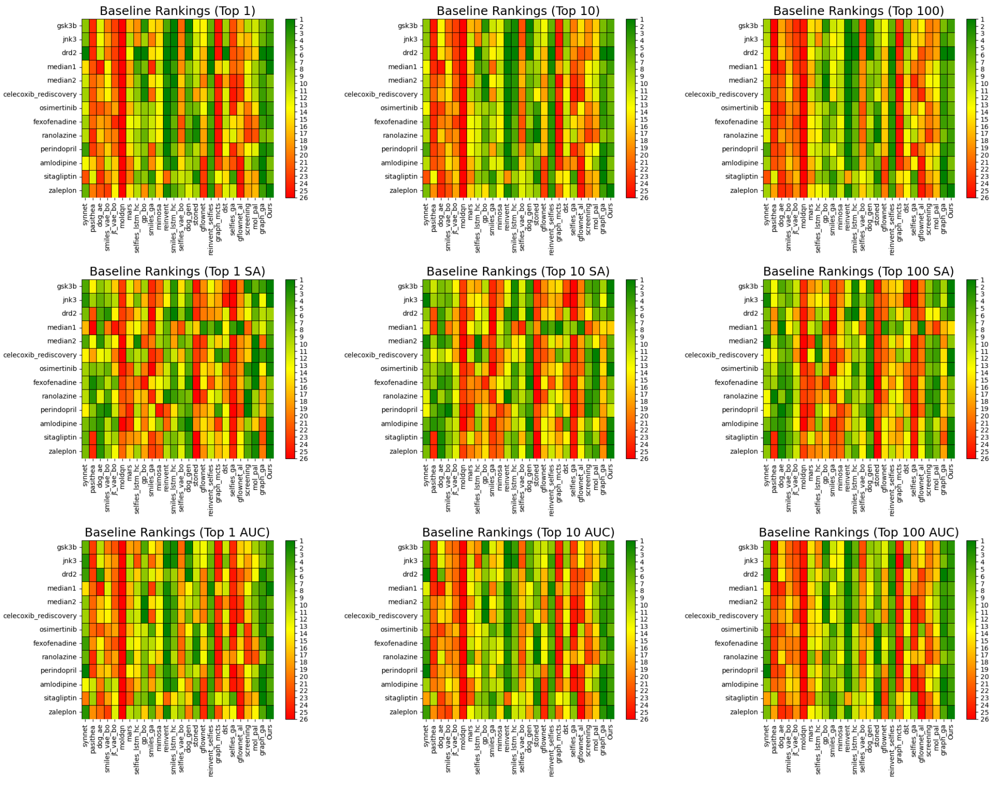
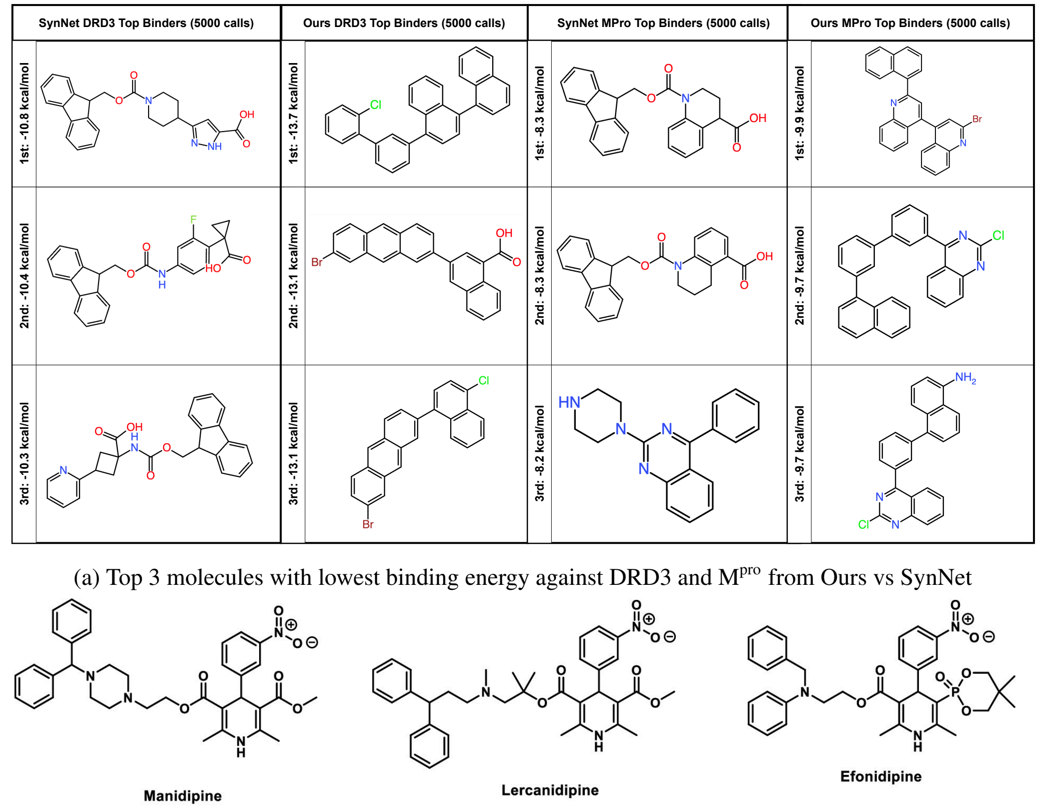


Figure 17: Ranking of our method against baselines on Top k Average Scores (top), SA Scores (middle) and AUC (bottom), for $k = 1, 10, 100$ (left, middle, right).

I Docking Case Study with AutoDock Vina



(a) Top 3 molecules with lowest binding energy against DRD3 and M^{pro} from Ours vs SynNet

(b) Top binders against M^{pro} from literature, based on consensus docking scores (Ghahremanpour et al., 2020)

We structurally analyze the top molecules discovered by our method, visualized in Figure 18a.

For our optimized binders against DRD3, the chlorine substituent and polycyclic aromatic structure suggest good potential for binding through $\pi - \pi$ interactions and halogen bonding. The bromine and carboxyl groups can enhance binding affinity through halogen bonding and hydrogen bonding, respectively. The polycyclic structure further supports $\pi - \pi$ stacking interactions. In general, they have a comparable binding capability to the baseline molecules, but with simpler structures, so the ease of synthesis for the predicted molecules are higher than the baseline molecules.

For our optimized binders against Mpro, the three predicted molecules contain multiple aromatic rings in conjugation with halide groups. The conformation structures of the multiple aligned aromatic rings play a significant role in docking and achieve ideal molecular pose and binding affinity to Mpro, compared to the baseline molecules shown in Figure 18b. The predicted structures indicate stronger $\pi - \pi$ interaction and halogen bonding compared with the baselines. In terms of ease of synthesis, Bromination reactions are typically straightforward, but multiple fused aromatic rings can take several steps to achieve. In general, the second and third can be easier to synthesize than the top binder due to less aromatic rings performed. However, the literature molecules appeared to be even harder to synthesize due to their high complexity structures. So the predicted molecules obtained a general higher ease of synthesis than the baseline molecules. Compared with the other baseline molecules, e.g. Manidipine, Lercanidipine, Efonidipine (Dihydropyridines), known for their calcium channel blocking activity, but not specifically protease inhibitors, Azelastine, Cinnoxicam, Idarubicin vary widely in their primary activities, not specifically designed for protease inhibition. Talampicillin and Lapatinib are also primarily designed for other mechanisms of action. Boceprevir, Nelfinavir, Indinavir, on the other hand, are known protease inhibitors with structures optimized for binding to protease active sites, so can serve as strong benchmarks. Overall, the binding effectiveness of the predicted molecules are quite comparable to the baseline molecules.