Reinforced Genetic Algorithm for Structure-based Drug Design

Anonymous Author(s)
Affiliation
Address
email

Abstract

Structure-based drug design (SBDD) aims to discover drug candidates by finding molecules (ligands) that bind tightly to a disease-related protein (targets), which is the primary approach to computer-aided drug discovery. Recently, applying deep generative models for three-dimensional (3D) molecular design conditioned on protein pockets to solve SBDD has attracted much attention, but their formulation as probabilistic modeling often leads to unsatisfactory optimization performance. On the other hand, traditional combinatorial optimization methods such as genetic algorithms (GA) have demonstrated state-of-the-art performance in various molecular optimization tasks. However, they do not utilize protein target structure to inform design steps but rely on a random-walk-like exploration, which leads to unstable performance and no knowledge transfer between different tasks despite the similar binding physics. To achieve a more stable and efficient SBDD, we propose Reinforced Genetic Algorithm (`RGA`) that uses neural models to prioritize the profitable design steps and suppress random-walk behavior. The neural models take the 3D structure of the targets and ligands as inputs and are pre-trained using native complex structures to utilize the knowledge of the shared binding physics from different targets and then fine-tuned during optimization. We conduct thorough empirical studies on optimizing binding affinity to various disease targets and show that `RGA` outperforms the baselines in terms of docking scores and is more robust to random initializations. The ablation study also indicates that the training on different targets helps improve the performance by leveraging the shared underlying physics of the binding processes.

1 Introduction

Rapid drug discovery that requires less time and cost is of significant interest in pharmaceutical science. Structure-based drug design (SBDD) [1] that leverages the three-dimensional (3D) structures of the disease-related proteins to design drug candidates is one primary approach to accelerate the drug discovery processes with physical simulation and data-driven modeling. According to the lock and key model [2], the molecules that bind tighter to a disease target are more likely to expose bioactivity against the disease, which has been verified experimentally [3]. As AlphaFold2 has provided accurate predictions to most human proteins [4, 5], SBDD has a tremendous opportunity to discover new drugs for new targets that we cannot model before [6].

SBDD could be formulated as an optimization problem where the objective function is the binding affinity estimated by simulations such as docking [2]. The most widely used design method is virtual screening, which exhaustively investigates every molecule in a library and ranks them. Lyu et al. successfully discovered new chemotypes for AmpC β-lactamase and the D_4 dopamine receptor by studying hundreds of millions of molecules with docking simulation [7]. However, the number of

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

the drug-like molecules is large as estimated to be 10^{60} [1], and it is computationally prohibitive to screen all of the possible molecules. Though machine learning approaches have been developed to accelerate screening [8, 9], it is still challenging to screen large enough chemical space within the foreseeable future.

Instead of naively screening a library, designing drug candidates with generative models has been highlighted as a promising strategy, exemplified by [10, 11]. This class of methods models the problem as the generation of ligands conditioned on the protein pockets. However, as generative models are trained to learn the distribution of known active compounds, they tend to produce molecules similar to training data [12], which discourages finding novel molecules and leads to unsatisfactory optimization performance.

A more straightforward solution is a combinatorial optimization algorithm that searches the implicitly defined discrete chemical space. As shown in multiple standard molecule optimization benchmarks [13, 14], combinatorial optimization methods, especially genetic algorithms (GA) [15, 16], often perform better than deep generative models. The key to superior performance is GA's action definition. Specifically, in each generation (iteration), GA maintains a population of possible candidates (a.k.a. parents) and conducts the crossover between two candidates and mutation from a single candidate to generate new offspring. These two types of actions, crossover and mutation, enable global and local traversal over the chemical space, allowing a thorough exploration and superior optimization performance.

However, most GAs select mutation and crossover operations randomly [15], leading to significant variance between independent runs. Especially in SBDD, when the oracle functions are expensive molecular simulations, it is resource-consuming to ensure stability by running multiple times. Further, most current combinatorial methods are designed for general-purpose molecular optimization and simply use a docking simulation as an oracle. It is challenging to leverage the structure of proteins in these methods, and we need to start from scratch whenever we change a protein target, even though the physics of ligand-protein interaction is shared. Ignoring the shared information across tasks leads to unnecessary exploration steps and, thus, demands for many more oracle calls, which require expensive and unnecessary simulations [17].

To overcome these issues in the GA method, we propose Reinforced Genetic Algorithm (`RGA`), which attempts to reformulate an evolutionary process as a Markov decision process and uses neural networks to make informed decisions and suppress the random-walk behavior. Specifically, we utilize an E(3)-equivariant neural network [18] to choose parents and mutation types based on the 3D structure of the ligands and proteins. The networks are pre-trained with various native complex structures to utilize the knowledge of the shared binding physics between different targets and then fine-tuned with a reinforcement learning algorithm during optimizations. We test `RGA`'s performance with various disease-related targets, including the main protease of SARS-CoV-2.

The main contributions of this paper can be summarized as follows:

- We propose an evolutionary Markov decision process (EMDP) that reformulates an evolutionary process as a Markov decision process, where the state is a population of molecules instead of a single molecule (Section 3.2).
- We show the first successful attempt to use a neural model to guide the crossover and mutation operations in a genetic algorithm to suppress random-walk behavior and explore the chemical space intelligently (Section 3.3).
- We present a structure-based de novo drug design algorithm that outperforms baseline methods consistently through thorough empirical studies on optimizing binding affinity by leveraging the underlying binding physics (Section 4).

2 Related Works

We will discuss the related works on methods of drug design and discuss the advantage of the proposed method over the existing works.

General Molecular Design. Molecular generation methods offer a promising direction for the automated design of molecules with desired pharmaceutical properties such as synthesis accessibility and drug-likeliness. Based on how to generate or search molecules, these approaches can be

categorized into two types, (1) deep generative models (DGMs) imitate the molecular data distribution, including variational autoencoder (VAE) [19, 20], generative adversarial network (GAN) [21, 22], normalizing flow model [23, 24], energy based model [25]; and (2) combinatorial optimization methods directly search over the discrete chemical space, including genetic algorithm (GA) [15, 26, 27], reinforcement learning approaches (RL) [28, 29, 30, 31, 32], Bayesian optimization (BO) [33], Markov chain Monte Carlo (MCMC) [34, 35, 36] and gradient ascent [37, 38].

General molecular design algorithms often use general black-box oracle functions, and some are only tested with trivial or self-designed oracles. For example, using penalized octanol-water partition coefficient (LogP) as the oracle function, it grows monotonically with the number of carbons, and thus there exists a trivial policy to optimize LogP. These oracles do not reflect the challenges of real drug discovery, and those algorithms have limited value for pharmaceutical discovery. Recent works are optimizing docking scores to simulate a more realistic discovery scenario [39, 40, 17, 41], same as our work. However, they are still ignoring the information in the given protein structures that could potentially accelerate the design process. However, the extension to leveraging the structural knowledge is nontrivial.

Structure-based Drug Design. Structure-based drug design (SBDD) could utilize the structural information to guide the design of molecules, which are potentially more efficient in drug discovery tasks but poses additional challenges of how to leverage the structures. Since early 1990s, various SBDD algorithms have been proposed, mostly based on combinatorial optimization algorithms such as tree search [42, 43, 44] and evolutionary algorithms [45, 46]. Those methods typically optimize the ligands in the pockets according to a physical model characterizing the binding affinity. For example, RASSE [42] used a force-field-like scoring function [47] to evaluate the partial solutions within a tree search. However, obtaining a fast and accurate model to quantify binding free energy itself is still an unsolved challenge.

Recently, generative modeling of 3D molecules conditioned on protein targets is attracting more attention [10, 11]. Similar to DGMs in general molecular design, those methods learn the atom's compositional and spatial distribution of native structure of protein-ligand complexes with neural models and design new ligands by complete the complex structure given targets. Deep generative models are end-to-end and data-driven thus surpass the necessity of understanding the physics of interaction. However, as the training objective is to learn the distribution of known active compounds, the models tend to produce molecules close to the training set [12], which is undesired in terms of patentability and leads to unsatisfactory optimization performance.

3 Method

In this paper, we focus on structure-based drug design. The goal is to design drug molecules (a.k.a. ligands) that could bind tightly with the disease-related proteins (a.k.a. targets). Given the 3D structures of the target proteins, including binding site information, docking is a popular computational method for assessing the binding affinity, which can be roughly retrieved as the free energy changes during the binding processes. We present a variant of genetic algorithm that is guided by reinforcement learning and a docking oracle. Next, we will first describe the general evolutionary process used in genetic algorithms (Section 3.1); Then we will present how to model this evolutionary process as a Markov decision process (MDP) where RL framework can be constructed (Section 3.2); After that, we describe the detailed implementation of this MDP framework using multiple policy networks (Section 3.3).

3.1 Evolutionary Process

In this section, we introduce the primary setting of the evolutionary processes. With both optimization performance and synthetic accessibility taken into account [48, 14], we follow the action settings in Autogrow 4.0 [16]. It demonstrated superior performance over other GA variants in the empirical validation of structure-based drug design [16], and its mutation actions originated from chemical reactions so that the designed molecules are more likely to be synthesizable. Specifically, an evolutionary processes starts by randomly sampling a *population* of drug candidates from a library. In each *generation* (iteration), it carries out (i) *crossover* between parents selected from the last generation, and (ii) *mutation* on a single child to obtain the offspring pool. An illustration of both

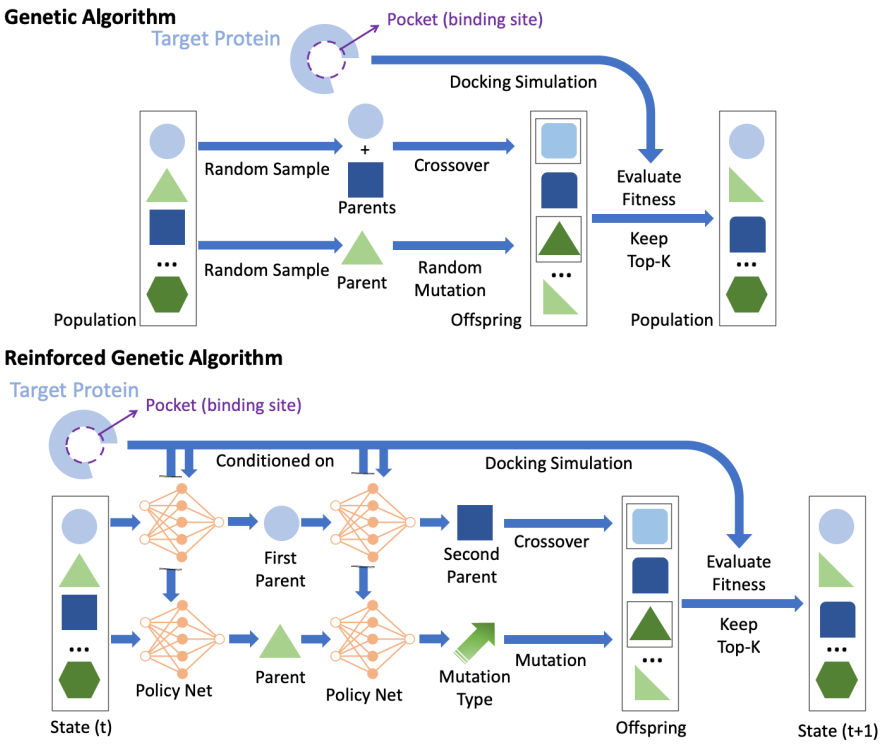


Figure 1: We illustrate one generation (iteration) of GA (top) and RGA pipeline (bottom). Specifically, we train policy networks that take the target and ligand as input to make informed choices on parents and mutation types in RGA.

crossover and mutation operations is available in Appendix. Note that we only adopted the action settings from Autogrow 4.0, without using other tricks such as elitism.

Crossover, also called recombination, combines the structure of two parents to generate new children. Following Autogrow 4.0 [16], we select two parents from the last generation and search for the largest common substructure shared between them. Then we generate two children by randomly switching their decorating moieties, i.e., the side chains attached to the common substructure.

Mutation operates on a single parent molecule and modifies its structure slightly. Following Autogrow 4.0 [16], we adopt transformations based on chemical reactions. Unlike naively defined atom-editing actions, mutation steps based on chemical reactions could ensure all modification is reasonable in reality, leading to a larger probability of designing synthesizable molecules. We included two types of chemical reactions: uni-molecular reactions, which only require one reactant, and bi-molecular reactions, which require two reactants. While uni-molecular reactions could be directly applied to the parent, we sample a purchasable compound to react with the parent when conducting a bi-molecular reaction. In both cases, the parent serves as one reactant, and we use the main product as the child molecule. We use the chemical reactions from [16], which was originally from [46, 49].

Evolution. At the t-th generation (iteration), given a population of molecules denoted as $\mathcal{S}^{(t)}$, we generate an offspring pool denoted as $\mathcal{Q}^{(t)}$ by applying crossover and mutation operations. Then we filter out the ones with undesirable physical and chemical properties (e.g., poor solubility, high toxicity) in the offspring pool and select the most promising K to form the next generation pool ($\mathcal{S}^{(t+1)}$).

3.2 Evolutionary Markov Decision Process

Next we propose the evolutionary Markov decision process (EMDP) that formulates an evolutionary process of genetic algorithm as a Markov decision process (MDP). The primary purpose is to utilize reinforcement learning algorithms to train networks to inform the decision steps to replace random selections. Taking a generation as a state, Markov property that requires $P(\mathcal{S}^{(t+1)}|\mathcal{S}^{(1)}, \cdots, \mathcal{S}^{(t)}) =$

$P(\mathcal{S}^{(t+1)}|\mathcal{S}^{(t)})$ is naturally satisfied by the evolutionary process described above, where $\mathcal{S}^{(t)}$ denotes the state at the t-th generation, which is the population of ligands. We use X to denote a ligand. We elaborate essential components for Markov decision process as follows, and the EMDP pipeline is illustrated in Figure 1.

State Space. We define the population at the t-step generation, $\mathcal{S}^{(t)}$, in the evolutionary process as the state at the t-step in an EMDP. A state includes population of candidate molecules (i.e., ligand, denoted X) and their 3D poses docked to the target, fully observable to the RL agent. At the beginning of the EMDP, we randomly select a population of candidate molecules and use docking simulation to yield their 3D poses as the initial state.

Action Space. The actions in an EMDP are to conduct the two evolutionary steps: crossover and mutation, in a population. For each evolutionary step, we need two actions to conduct it. Concretely, crossover ($X^{\text{parent 1}}_{\text{crossover}}, X^{\text{parent 2}}_{\text{crossover}} \xrightarrow{\text{crossover}} X^{\text{child 1}}_{\text{crossover}}, X^{\text{child 2}}_{\text{crossover}}$) can be divided to:

1. select the first candidate ligand $X^{\text{parent 1}}_{\text{crossover}}$ from the current state (population $\mathcal{S}^{(t)}$);
2. conditioned on the first selected candidate $X^{\text{parent 1}}_{\text{crossover}}$, select the second candidate ligand $X^{\text{parent 2}}_{\text{crossover}}$ from the remaining candidate ligand set $\mathcal{S}^{(t)} - \{X^{\text{parent 1}}_{\text{crossover}}\}$ and apply crossover (Section 3.1) to them.

Mutation ($X^{\text{parent}}_{\text{mutation}} \xrightarrow{\text{mutated by } \xi} X^{\text{child}}_{\text{mutation}}$) can be divided to:

1. select the candidate ligand $X^{\text{parent}}_{\text{mutated}}$ to be mutated from the current state (population $\mathcal{S}^{(t)}$);
2. conditioned on the selected candidate ligand $X^{\text{parent}}_{\text{mutated}}$, select the reaction ξ from the reaction set $\mathcal{R}$ and apply it to $X^{\text{parent}}_{\text{mutated}}$.

As applying the crossover and mutation steps are deterministic, the actions in an EMDP focus on selecting parents and mutation types. Upon finish the action, we could obtain the offspring pool, $\mathcal{Q}^{(t)}$.

State Transition Dynamics. The state transition in an EMDP is identical to the evolution in an evolutionary process. Once we finish the actions and obtain the offspring pool, $\mathcal{Q}^{(t)} = \{X^{\text{child 1}}, X^{\text{child 2}}, \cdots\}$, we apply molecular quality filters to filter out the ones unlikely to be drug and then select the most promising K to form the parent set for the next generation ($\mathcal{S}^{(t+1)}$).

Reward. We define the reward as the binding affinity change (docking score). The actions leading to stronger binding score would be prioritized. As there is no "episode" concept in an EMDP, we treat every step equally.

3.3 Target-Ligand Policy Network

To utilize molecular structures' translational and rotational invariance, we adopt equivariance neural networks (ENNs) [18] as the target-ligand policy neural networks to select the actions in both mutation and crossover steps. Each ligand has a 3D pose that binds to the target protein, and the complex serves as the input of ENN.

Specifically, we want to model a 3D graph $\mathcal{Y}$, which can be ligand, target, or target-ligand complex. The input feature can be described as $\mathcal{Y} = (\mathcal{A}, \mathcal{Z})$, where $\mathcal{A}$ represents atoms' categories (the vocabulary set $\mathcal{V} = \{H, C, O, N, \cdots\}$) and $\mathcal{Z}$ represents 3D coordinates of the atoms. Suppose $\mathbf{D} \in \mathbb{R}^{|\mathcal{V}| \times d}$ is the embedding matrix of all the categories of atoms in a vocabulary set $\mathcal{V}$, is randomly initialized and learnable, d is the hidden dimension in ENN. Each kind of atom corresponds to a row in $\mathbf{D}$. We suppose there are N atoms, and each atom corresponds to a node in the 3D graph. Node embeddings at the l-th layer are denoted as $\mathbf{H}^{(l)} = \{\mathbf{h}_i^{(l)}\}_{i=1}^N$, where $l = 0, 1, \cdots, L$, L is number of layers in ENN. The initial node embedding $\mathbf{h}_i^{(0)} = \mathbf{D}^\top \mathbf{a}_i \in \mathbb{R}^d$ embeds the i-th node, where $\mathbf{a}_i$ is one-hot vector that encode the category of the i-th atom. Coordinate embeddings at the l-th layer are denoted $\mathbf{Z}^{(l)} = \{\mathbf{z}_i^{(l)}\}_{i=1}^N$. The initial coordinate embeddings $\mathbf{Z}^{(0)} = \{\mathbf{z}_i\}_{i=1}^N$ are the real 3D coordinates of all the nodes. The following equation defines the feedforward rules of ENN, for

$i, j = 1, \cdots, N,\ i \neq j,\ l = 0, 1, \cdots, L-1$, we have

$$\begin{aligned}
&\mathbf{w}_{ij}^{(l+1)} = \text{MLP}_e\Big(\mathbf{h}_i^{(l)} \oplus \mathbf{h}_j^{(l)} \oplus ||\mathbf{z}_i^{(l)} - \mathbf{z}_j^{(l)}||_2^2\Big) \ \in \mathbb{R}^d, \qquad \mathbf{v}_i^{(l+1)} = \sum_{j=1, j\neq i}^{N} \mathbf{w}_{ij}^{(l+1)} \ \in \mathbb{R}^d, \\
&\mathbf{z}_i^{(l+1)} = \mathbf{z}_i^{(l)} + \sum_{j=1, j\neq i}^{N} \Big(\mathbf{z}_i^{(l)} - \mathbf{z}_j^{(l)}\Big)\text{MLP}_x\Big(\mathbf{w}_{ij}^{(l)}\Big) \in \mathbb{R}^3, \quad \mathbf{h}_i^{(l+1)} = \text{MLP}_h\Big(\mathbf{h}_i^{(l)} \oplus \mathbf{v}_i^{(l+1)}\Big) \in \mathbb{R}^d, \\
&\mathbf{h}_{\mathcal{Y}} = \sum_{i=1}^{N} \mathbf{h}_i^{(L)} \ \in \mathbb{R}^d \qquad \Longrightarrow \ \underline{\mathbf{h}_{\mathcal{Y}} = \text{ENN}(\mathcal{Y})}
\end{aligned} \tag{1}$$

where $\oplus$ denotes the concatenation of vectors; $\text{MLP}_e(\cdot) : \mathbb{R}^{2d+1} \to \mathbb{R}^d; \text{MLP}_x(\cdot) : \mathbb{R}^d \to \mathbb{R}; \text{MLP}_h(\cdot) : \mathbb{R}^{2d} \to \mathbb{R}^d$ are all two-layer multiple layer perceptrons (MLPs) with Swish activation in the hidden layer [50]. At the l-th layer, $\mathbf{w}_{ij}^{(l)}$ represents the message vector for the edge from node i to node j; $\mathbf{v}_i^{(l)}$ represents the message vector for node i, $\mathbf{z}_i^{(l)}$ is the position embedding for node i; $\mathbf{h}_i^{(l)}$ is the node embedding for node i. $\mathbf{H}^{(L)} = [\mathbf{h}_1^{(L)}, \cdots, \mathbf{h}_N^{(L)}]$ are the node embeddings of the L-th (last) layer. We aggregate them using sum function as readout function to obtain a representation of the 3D graph, denoted $\mathbf{h}_{\mathcal{Y}}$. The whole process is written as $\mathbf{h}_{\mathcal{Y}} = \text{ENN}(\mathcal{Y})$.

Crossover Policy Network. We design two policy networks for two corresponding actions in a crossover, as mentioned in Section 3.2. (1) the first action in crossover operation is to select the first parent ligand $X_{\text{crossover}}^{\text{parent 1}}$ from the population $\mathcal{S}^{(t)}$. Similar to the first action in mutation operation, we obtain a valid probability distribution over all the available ligands based on target-ligand complex as input feature and ENN as the neural network architecture, the selection probability of the ligand $X_{\text{crossover}}^{\text{parent 1}} \in \mathcal{S}^{(t)}$ is $p_{\text{crossover}}^{(1)}(X_{\text{crossover}}^{\text{parent 1}}|\mathcal{S}^{(t)}) = \frac{\exp\left(\text{MLP}(\mathbf{h}_{\mathcal{T}\&X_{\text{crossover}}^{\text{parent 1}}})\right)}{\sum_{X'\in\mathcal{S}^{(t)}} \exp\left(\text{MLP}(\mathbf{h}_{\mathcal{T}\&X'})\right)}$, where $\mathcal{T}$ and X denotes target and ligand (including 3D pose), respectively, $\mathcal{T}\&X$ denotes target-ligand complex. (2) The second action is to select the second parent ligand conditioned on the first parent ligand selected in the first action. Specifically, for ligand in the remaining population set, we concatenate the ENN's embedding of the target, first parent ligand $X_{\text{crossover}}^{\text{parent 1}}$ and the second parent ligand $X_{\text{mutation}}^{\text{parent 2}}$, and feed it into an MLP to estimate a scalar as an unnormalized probability. The unnormalized probabilities for all the ligands in the remaining population set are normalized via softmax function, i.e., $p_{\text{crossover}}^{(2)}(X_{\text{crossover}}^{\text{parent 2}}|X_{\text{crossover}}^{\text{parent 1}}, \mathcal{S}^{(t)}) = \text{Softmax}\big\{\text{MLP}(\mathbf{h}_{\mathcal{T}} \oplus \mathbf{h}_{X_{\text{crossover}}^{\text{parent 1}}} \oplus \mathbf{h}_{X_{\text{crossover}}^{\text{parent 2}}}), \cdots, \big\}_{X_{\text{crossover}}^{\text{parent 2}} \in \mathcal{S}^{(t)} - \{X_{\text{crossover}}^{\text{parent 1}}\}}$. Given two parents ligands, crossover finds the largest substructure that the two parent compounds share and generates a child by combining their decorating moieties. Thus, the generation of child ligands are deterministic, and the probability of the generated ligands $X_{\text{crossover}}^{\text{child}}$ is

$$\begin{aligned}
&p_{\text{crossover}}(X_{\text{crossover}}^{\text{child 1}}, X_{\text{crossover}}^{\text{child 2}}|\mathcal{S}^{(t)}) = p_{\text{crossover}}(X_{\text{crossover}}^{\text{parent 1}}, X_{\text{crossover}}^{\text{parent 2}}|\mathcal{S}^{(t)}) \\
&= p_{\text{crossover}}^{(1)}(X_{\text{crossover}}^{\text{parent 1}}|\mathcal{S}^{(t)}) \cdot p_{\text{crossover}}^{(2)}(X_{\text{crossover}}^{\text{parent 2}}|X_{\text{crossover}}^{\text{parent 1}}, \mathcal{S}^{(t)}).
\end{aligned} \tag{2}$$

Mutation Policy Network. We design two policy networks for two corresponding actions in mutation, as mentioned in Section 3.2. (1) the first action in mutation operation is to select a candidate ligand to be mutated from population $\mathcal{S}^{(t)}$. It models the 3D target-ligand complex to learn if there is improvement space in the current complex. Formally, we obtain a valid probability distribution over all the available ligands based on target-ligand complex as input feature and ENN as neural architecture, the selection probability of the ligand $X_{\text{mutation}}^{\text{parent}} \in \mathcal{S}^{(t)}$ is $p_{\text{mutation}}^{(1)}(X_{\text{mutation}}^{\text{parent}}|\mathcal{S}^{(t)}) = \frac{\exp\left(\text{MLP}(\mathbf{h}_{\mathcal{T}\&X_{\text{mutation}}^{\text{parent}}})\right)}{\sum_{X'\in\mathcal{S}^{(t)}} \exp\left(\text{MLP}(\mathbf{h}_{\mathcal{T}\&X'})\right)}$, where $\mathcal{T}\&X$ denotes target-ligand complex, $\mathbf{h}_{\mathcal{T}\&X} = \text{ENN}(\mathcal{T}\&X)$ represents the ENN's embedding of target-ligand complex. (2) The second action is to select the SMARTS reaction from the reaction set conditioned on the selected ligand in the first action. Specifically, for each reaction, we generate the new ligand $X_{\text{mutation}}^{\text{child}}$, then obtain the embedding of target, first ligand $X_{\text{mutation}}^{\text{parent}}$ and the new ligand $X_{\text{mutation}}^{\text{child}}$ through ENN, concatenate these three embeddings and feed it into an MLP to estimate a scalar as unnormalized

probability. The unnormalized probabilities for all the reactions are normalized via softmax function, i.e., $p^{(2)}_{\text{mutation}}(\xi|X^{\text{parent}}_{\text{mutation}}, \mathcal{S}^{(t)}) = \text{Softmax}\big\{\text{MLP}(\mathbf{h}_{\mathcal{T}} \oplus \mathbf{h}_{X^{\text{parent}}_{\text{mutation}}} \oplus \mathbf{h}_{X^{\text{child}}_{\text{mutation}}}]), \cdots, \big\}_{\xi\in\mathcal{R}}$, where $X^{\text{parent}}_{\text{mutation}} \xrightarrow{\text{mutated by } \xi} X^{\text{child}}_{\text{mutation}}$, $\mathcal{R}$ is the reaction set. The probability of the generated ligand $X^{\text{child}}_{\text{mutation}}$ is

$$p_{\text{mutation}}(X^{\text{child}}_{\text{mutation}}|\mathcal{S}^{(t)}) = p^{(1)}_{\text{mutation}}(X^{\text{parent}}_{\text{mutation}}|\mathcal{S}^{(t)}) \cdot p^{(2)}_{\text{mutation}}(\xi|X^{\text{parent}}_{\text{mutation}}, \mathcal{S}^{(t)}). \tag{3}$$

Policy Gradient. We leverage policy gradient to train the target-ligand policy neural network. Specifically, we consider maximizing the expected reward as objective via REINFORCE [28],

$$\max\ \mathbb{E}_{X\sim p(X|\mathcal{S}^{(t)})}\big[\text{Reward}(X)\big], \tag{4}$$

where $p(X)$ is defined in Equation (2) and (3) for crossover and mutation, respectively. The whole pipeline is illustrated in Figure 1. To provide a warm start and leverage the structural information, we pretrain the ENNs on 3D target-ligand binding affinity prediction task, where the inputs are the target-ligand complexes, and the outputs are their binding affinity.

4 Experiment

In this section, we briefly describe the experimental setup and results. The Appendix includes more details, including software configuration, implementation details, dataset description & processing, hyperparameter tuning, code repository, ablation study, and additional experimental results.

4.1 Experimental Setup

Docking Simulation. We adopt AutoDock Vina [51] to evaluate the binding affinity. The docking score estimated by AutoDock Vina is called Vina score and roughly characterizes the free energy changes of binding processes in kcal/mol. Thus lower Vina score means a stronger binding affinity between the ligand and target. We picked various disease-related proteins, including G-protein coupling receptors (GPCRs) and kinases from DUD-E [52] and the SARS-CoV-2 main protease [53] as targets. Please see the Appendix for more information.

Baselines. The baseline methods cover traditional brute-force search methods (Screening), deep generative models (JTVAE and Gen3D), genetic algorithm (GA+D, graph-GA, Autogrow 4.0), reinforcement learning methods (MolDQN, RationaleRL, REINVENT, GEGL), and MCMC method (MARS). Gen3D and Autogrow 4.0 are structure-based drug design methods, while others are general-purpose molecular design methods. Although methods explicitly utilizing target structures are relatively few, we add general-purpose molecular design methods optimizing the same docking oracle scores as ours, which is a common use case, as baselines [15, 14]. Concretely, **Screening** mimics high throughput screening via sampling from ZINC database randomly; **JTVAE** (Junction Tree Variational Auto-Encoder) [20] uses a Bayesian optimization on the latent space to indirectly optimize molecules; **Gen3D** [10] is an auto-regressive generative model that grows 3D structures atom-wise inside the binding pocket; **GA+D** [26] represents molecule as SELFIES string [54] and uses genetic algorithm enhanced by a discriminator neural network; **Graph-GA** [15] conduct genetic algorithm on molecular graph representation; **Autogrow 4.0** [16] is the state-of-the-art genetic algorithm in structure-based drug design; **MolDQN** (Molecule Deep Q-Network) [30] leverages deep Q-value learning to grow molecules atom-wisely; **RationaleRL** [31] uses rationale (e.g., functional groups or subgraphs) as the building block and a policy gradient method to guide the training of graph neural network-based generator; **REINVENT** [28] represent molecules as SMILES string and uses policy gradient based reinforcement learning methods to guide the training of the RNN generator; **GEGL** (genetic expert-guided learning) [32] uses LSTM guided by reinforcement learning to imitate the GA exploration; **MARS** (Markov Molecule Sampling) [35] leverages Markov chain Monte Carlo sampling (MCMC) with adaptive proposal and annealing scheme to search chemical space. To conduct a fair comparison, we limit the number of oracle calls to 1,000 times for each method. Implementation details of all the baseline methods are available in Appendix.

Dataset: we randomly select molecules from ZINC [55] database (around 250 thousands drug-like molecules) as 0-th generation of the genetic algorithms (`RGA`, Autogrow 4.0, GA+D). ZINC also serves as the training data for pretraining the model in JTVAE, REINVENT, RationaleRL, etc. We adopt CrossDocked2020 [56] dataset that contains around 22 million ligand-protein complexes as

Table 1: The summarized performance of different methods. The mean and standard deviation across targets are reported. Arrows (↑, ↓) indicate the direction of better performance. For each metric, the best method is underlined and the top-3 methods are bolded. RGA-pretrain and RGA-KT are two variants of RGA that without pretraining and without training on different target proteins, respectively.

Method	TOP-100↓	TOP-10↓	TOP-1↓	Nov↑	Div↑	QED↑	SA↓
screening	-9.351±0.643	-10.433±0.563	-11.400±0.630	0.0±0.0%	0.858±0.005	0.678±0.022	2.689±0.077
MARS	-7.758±0.612	-8.875±0.711	-9.257±0.791	100.0±0.0%	**0.877±0.001**	**0.709±0.008**	**2.450±0.034**
MolDQN	-6.287±0.396	-7.043±0.487	-7.501±0.402	100.0±0.0%	**0.877±0.009**	0.170±0.024	5.833±0.182
GEGL	-9.064±0.920	-9.91±0.990	-10.45±1.040	100.0±0.0%	0.853±0.003	0.643±0.014	2.99±0.054
REINVENT	-10.181±0.441	-11.234±0.632	-12.010±0.833	100.0±0.0%	0.857±0.011	0.445±0.058	2.596±0.116
RationaleRL	-9.233±0.920	-10.834±0.856	-11.642±1.102	100.0±0.0%	0.717±0.025	0.315±0.023	2.919±0.126
JTVAE	-9.291±0.702	-10.242±0.839	-10.963±1.133	98.0±0.027%	0.867±0.001	0.593±0.035	3.222±0.136
Gen3D	-8.686±0.450	-9.285±0.584	-9.832±0.324	100.0±0.0%	**0.870±0.006**	0.701±0.016	3.450±0.120
GA+D	-7.487±0.757	-8.305±0.803	-8.760±0.796	99.2±0.011%	0.834±0.035	0.405±0.024	5.024±0.164
Graph-GA	**-10.848±0.860**	**-11.702±0.930**	**-12.302±1.010**	100.0±0.0%	0.811±0.037	0.456±0.067	3.503±0.367
Autogrow 4.0	**-11.371±0.398**	**-12.213±0.623**	**-12.474±0.839**	100.0±0.0%	0.852±0.011	**0.748±0.022**	**2.497±0.049**
RGA (ours)	**-11.867±0.170**	**-12.564±0.287**	**-12.869±0.473**	100.0±0.0%	0.857±0.020	**0.742±0.036**	**2.473±0.048**
RGA - pretrain	-11.443±0.219	-12.424±0.386	-12.435±0.654	100.0±0.0%	0.854±0.035	0.750±0.034	2.494±0.043
RGA - KT	-11.434±0.169	-12.437±0.354	-12.502±0.603	100.0±0.0%	0.853±0.028	0.738±0.034	2.501±0.050

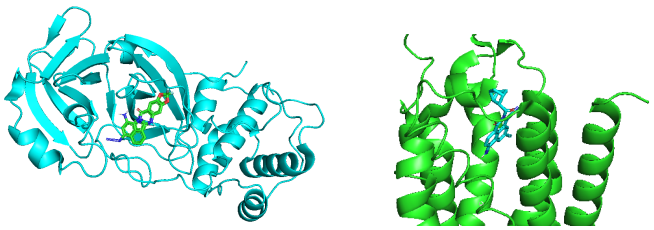
the training data for pretraining the policy neural networks, as mentioned in Section 3.3. More descriptions are available in Appendix.

Metrics. The selection of evaluation metrics follows recent works in molecule optimization [20, 26, 31, 35] and structure-based drug design [16, 10, 14]. For each method, we select top-100 molecules with the best docking scores for evaluation and consider the following metrics: **TOP-1/10/100** (average docking score of top-1/10/100 molecules): docking score directly measures the binding affinity between the ligand and target and is the most informative metric in structure-based drug design; **Novelty (Nov)** (% of the generated molecules that are not in training set); **Diversity (Div)** (average pairwise Tanimoto distance between the Morgan fingerprints); We also evaluate some simple pharmaceutical properties, including quantitative drug-likeness (**QED**) and synthetic accessibility (**SA**). QED score indicates drug-likeliness ranging from 0 to 1 (higher the better). SA score ranges from 1 to 10 (lower the better).

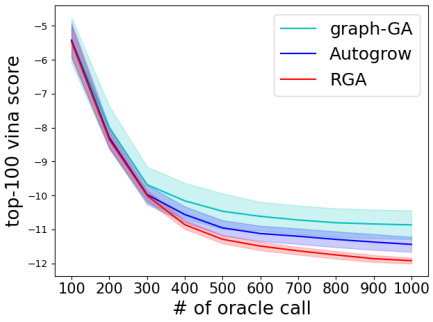
4.2 Results

Stronger Optimization Performance. We summarized the main results of the structure-based drug design in Table 1. We evaluate all the methods on all targets and report each metric's mean and standard deviations across all targets. Our result shows RGA achieves the best performance in TOP-100/10/1 scores among all methods we compared. Compared to Autogrow 4.0, RGA's better performance in docking score demonstrates that the policy networks contribute positively to the chemical space navigation and eventually help discover more potent binding molecules. On the other hand, including longer-range navigation steps enabled by crossover leads to superior performance than other RL methods (REINVENT, MolDQN, GEGL and RationaleRL) that only focus on local modifications. In addition, we also observed competitive structure quality measured by QED (> 0.7) and SA_Score (< 2.5) in Autogrow 4.0 and RGA without involving them as optimization objectives, thanks to the mutation steps originating from chemical reactions. We visualize two designed ligands with optimal affinity for closer inspection in Figure 2(a) and 2(b), and find both ligands bind tightly with the targets.

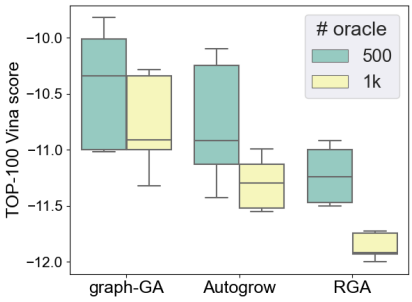
Suppressed Random-Walk Behavior. Especially in SBDD, when the oracle functions are expensive molecular simulations, robustness to random seeds is essential for improving the worst-case performance of algorithms. One of the major issues in traditional GAs is that they have a significant variance between multiple independent runs as they randomly select parents for crossover and mutation types. To examine this behavior, we run five independent runs for RGA, Autogrow 4.0 and graph-GA (three best baselines, all are GA methods) on all targets and plot the standard deviations between runs in Figure 2(c) and 2(d). With policy networks guiding the action steps, we observed that the random-walk behavior in Autogrow 4.0 was suppressed in RGA, indicated by the smaller variance. Especially in the later learning phase (after 500 oracle calls), the policy networks are fine-tuned and guide the search more intelligently. This advantage leads to improved worst-case performance and a higher probability of successfully identifying bioactive drug candidates with constrained resources.



(a) Example of 7l11, -10.8 kcal/mol (b) Example of 3eml, -13.2 kcal/mol



(c) TOP-100 vs # oracle call



(d) Score bar for different runs

Figure 2: (a) and (b): Example of ligand poses (generated by RGA) and binding sites of target structures. Example of 7l11: the PDB ID of target is 7l11, which is SARS-COV-2(2019-NCOV) main protease, the Vina score is -10.8 kcal/mol. Example of 3eml: the PDB ID is 3eml, which is a human A2A Adenosine receptor, the Vina score is -13.2 kcal/mol. (c) and (d): studies of suppressed random-walk behavior. (c) reports TOP-100 docking score as a function of oracle calls. The results are the means and standard deviations of 5 independent runs. (d) shows the bars of TOP-100 docking score for various independent runs.

Knowledge Transfer Between Protein Targets. To verify if RGA benefited from learning the shared physics of ligand-target interaction, we conducted an ablation study whose results are in the last two rows of Table 1. Specifically, we compare RGA with two variants: (1) RGA-pretrain that does not pretrain the policy network with all native complex structures in the CrossDocked2020; (2) RGA-KT (knowledge transfer) that fine-tune the networks with data of individual target independently. We find that both strategies positively contribute to RGA on TOP-100/10/1 docking score. These results demonstrate the policy networks successfully learn the shared physics of ligand-target interactions and leverage the knowledge to improve their performance.

5 Conclusion

In this paper, we propose Reinforced Genetic Algorithm (RGA) to tackle the structure-based drug design problem. RGA reformulate the evolutionary process in genetic algorithms as a Markov decision process called evolutionary Markov decision process (EMDP) so that the searching processes could benefit from trained neural models. Specifically, we train policy networks to choose the parents to crossover and mutate instead of randomly sampling them. Further, we also leverage the common physics of the ligand-target interaction and adopt a knowledge-transfer strategy that uses data from other targets to train the networks. Through empirical study, we show that RGA has strong and robust optimization performance, consistently outperforming baseline methods in terms of docking score.

Though we adopted mutations originating from chemical reactions and the structural quality metrics seem good, we need to emphasize that the designed molecules from RGA do not guarantee synthesizability [48], as the crossover operations may break inheriting synthesizability. Directly working on synthetic pathways could solve the problem [27, 57], but the extension is not trivial. As for future direction, we expect to theoretically analyze the EMDP formulation and the performance of RGA. We also expect to generalize RGA to other combinatorial optimization scenarios, such as symbolic laws discovery [58], quantum circuits design [59]), etc.

References

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

[2] Ashutosh Tripathi and Vytas A Bankaitis. Molecular docking: From lock and key to combination lock. *Journal of molecular medicine and clinical applications*, 2(1), 2017.

[3] Assaf Alon, Jiankun Lyu, Joao M Braz, Tia A Tummino, Veronica Craik, Matthew J O'Meara, Chase M Webb, Dmytro S Radchenko, Yurii S Moroz, Xi-Ping Huang, et al. Structures of the $\sigma 2$ receptor enable docking for bioactive ligand discovery. *Nature*, 600(7890):759–764, 2021.

[4] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, 2021.

[5] Mihaly Varadi, Stephen Anyango, Mandar Deshpande, Sreenath Nair, Cindy Natassia, Galabina Yordanova, David Yuan, Oana Stroe, Gemma Wood, Agata Laydon, et al. Alphafold protein structure database: Massively expanding the structural coverage of protein-sequence space with high-accuracy models. *Nucleic acids research*, 50(D1):D439–D444, 2022.

[6] Feng Ren, Xiao Ding, Min Zheng, Mikhail Korzinkin, Xin Cai, Wei Zhu, Alexey Mantsyzov, Alex Aliper, Vladimir Aladinskiy, Zhongying Cao, et al. Alphafold accelerates artificial intelligence powered drug discovery: Efficient discovery of a novel Cyclin-dependent Kinase 20 (CDK20) small molecule inhibitor. *arXiv preprint arXiv:2201.09647*, 2022.

[7] 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.

[8] David E Graff, Eugene I Shakhnovich, and Connor W Coley. Accelerating high-throughput virtual screening through molecular pool-based active learning. *Chemical science*, 12(22):7866–7881, 2021.

[9] Francesco Gentile, Jean Charle Yaacoub, James Gleave, Michael Fernandez, Anh-Tien Ton, Fuqiang Ban, Abraham Stern, and Artem Cherkasov. Artificial intelligence–enabled virtual screening of ultra-large chemical libraries with deep docking. *Nature Protocols*, pages 1–26, 2022.

[10] Shitong Luo, Jiaqi Guan, Jianzhu Ma, and Jian Peng. A 3D generative model for structure-based drug design. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[11] Yibo Li, Jianfeng Pei, and Luhua Lai. Structure-based de novo drug design using 3D deep generative models. *Chemical science*, 12(41):13664–13675, 2021.

[12] W Patrick Walters and Mark Murcko. Assessing the impact of generative AI on medicinal chemistry. *Nature biotechnology*, 38(2):143–145, 2020.

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

[14] Kexin Huang, Tianfan Fu, Wenhao Gao, Yue Zhao, Yusuf Roohani, Jure Leskovec, Connor W Coley, Cao Xiao, Jimeng Sun, and Marinka Zitnik. Therapeutics Data Commons: machine learning datasets and tasks for therapeutics. *NeurIPS Track Datasets and Benchmarks*, 2021.

[15] Jan H Jensen. A graph-based genetic algorithm and generative model/Monte Carlo Tree Search for the exploration of chemical space. *Chemical science*, 10(12):3567–3572, 2019.

[16] Jacob O Spiegel and Jacob D Durrant. AutoGrow4: an open-source genetic algorithm for de novo drug design and lead optimization. *Journal of cheminformatics*, 12(1):1–16, 2020.

[17] Austin Tripp, Gregor NC Simm, and José Miguel Hernández-Lobato. A fresh look at de novo molecular design benchmarks. In *NeurIPS 2021 AI for Science Workshop*, 2021.

[18] Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E (n) equivariant graph neural networks. *Proceedings of the 38th International Conference on Machine Learning, ICML*, 139:9323–9332, 2021.

[19] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 2018.

[20] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. *ICML*, 2018.

[21] Gabriel Lima Guimaraes, Benjamin Sanchez-Lengeling, Carlos Outeiral, Pedro Luis Cunha Farias, and Alán Aspuru-Guzik. Objective-Reinforced Generative Adversarial Networks (ORGAN) for sequence generation models. *arXiv preprint arXiv:1705.10843*, 2017.

[22] Nicola De Cao and Thomas Kipf. MolGAN: An implicit generative model for small molecular graphs, 2018.

[23] Chence Shi, Minkai Xu, Zhaocheng Zhu, Weinan Zhang, Ming Zhang, and Jian Tang. GraphAF: a flow-based autoregressive model for molecular graph generation. In *ICLR*, 2020.

[24] Youzhi Luo, Keqiang Yan, and Shuiwang Ji. GraphDF: A discrete flow model for molecular graph generation. *Proceedings of the 38th International Conference on Machine Learning, ICML*, 139:7192–7203, 2021.

[25] Meng Liu, Keqiang Yan, Bora Oztekin, and Shuiwang Ji. GraphEBM: Molecular graph generation with energy-based models. *arXiv preprint arXiv:2102.00546*, 2021.

[26] AkshatKumar Nigam, Pascal Friederich, Mario Krenn, and Alán Aspuru-Guzik. Augmenting genetic algorithms with deep neural networks for exploring the chemical space. In *ICLR*, 2020.

[27] Wenhao Gao, Rocío Mercado, and Connor W Coley. Amortized tree generation for bottom-up synthesis planning and synthesizable molecular design. *International Conference on Learning Representations*, 2022.

[28] M. Olivecrona, T. Blaschke, O. Engkvist, and H. Chen. Molecular de-novo design through deep reinforcement learning. *Journal of Cheminformatics*, 2017.

[29] Jiaxuan You, Bowen Liu, Rex Ying, Vijay Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. In *NIPS*, 2018.

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

[31] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Multi-objective molecule generation using interpretable substructures. In *International Conference on Machine Learning*, pages 4849–4859. PMLR, 2020.

[32] Sungsoo Ahn, Junsu Kim, Hankook Lee, and Jinwoo Shin. Guiding deep molecular optimization with genetic exploration. *Advances in neural information processing systems*, 33:12008–12021, 2020.

[33] Ksenia Korovina, Sailun Xu, Kirthevasan Kandasamy, Willie Neiswanger, Barnabas Poczos, Jeff Schneider, and Eric Xing. ChemBO: Bayesian optimization of small organic molecules with synthesizable recommendations. In *International Conference on Artificial Intelligence and Statistics*, pages 3393–3403. PMLR, 2020.

[34] Tianfan Fu, Cao Xiao, Xinhao Li, Lucas M Glass, and Jimeng Sun. MIMOSA: Multi-constraint molecule sampling for molecule optimization. *AAAI*, 2020.

[35] Yutong Xie, Chence Shi, Hao Zhou, Yuwei Yang, Weinan Zhang, Yong Yu, and Lei Li. MARS: Markov molecular sampling for multi-objective drug discovery. In *ICLR*, 2021.

[36] Yoshua Bengio, Tristan Deleu, Edward J. Hu, Salem Lahlou, Mo Tiwari, and Emmanuel Bengio. GFlowNet foundations. *CoRR*, abs/2111.09266, 2021.

[37] Tianfan Fu, Wenhao Gao, Cao Xiao, Jacob Yasonik, Connor W Coley, and Jimeng Sun. Differentiable scaffolding tree for molecular optimization. *International Conference on Learning Representations*, 2022.

[38] Cynthia Shen, Mario Krenn, Sagi Eppel, and Alan Aspuru-Guzik. Deep molecular dreaming: Inverse machine learning for de-novo molecular design and interpretability with surjective representations. *Machine Learning: Science and Technology*, 2021.

[39] Tobiasz Cieplinski, Tomasz Danel, Sabina Podlewska, and Stanislaw Jastrzebski. We should at least be able to design molecules that dock well. *arXiv:2006.16955*, 2020.

[40] Casper Steinmann and Jan H Jensen. Using a genetic algorithm to find molecules with good docking scores. *PeerJ Physical Chemistry*, 3:e18, 2021.

[41] Soojung Yang, Doyeong Hwang, Seul Lee, Seongok Ryu, and Sung Ju Hwang. Hit and lead discovery with explorative RL and fragment-based molecule generation. *Advances in Neural Information Processing Systems*, 34, 2021.

[42] Zhaowen Luo, Renxiao Wang, and Luhua Lai. RASSE: a new method for structure-based drug design. *Journal of chemical information and computer sciences*, 36(6):1187–1194, 1996.

[43] Valerie Gillet, A Peter Johnson, Pauline Mata, Sandor Sike, and Philip Williams. SPROUT: a program for structure generation. *Journal of computer-aided molecular design*, 7(2):127–153, 1993.

[44] David A Pearlman and Mark A Murcko. CONCEPTS: New dynamic algorithm for de novo drug suggestion. *Journal of computational chemistry*, 14(10):1184–1193, 1993.

[45] Dominique Douguet, Etienne Thoreau, and Gérard Grassy. A genetic algorithm for the automated generation of small organic molecules: drug design using an evolutionary algorithm. *Journal of computer-aided molecular design*, 14(5):449–466, 2000.

[46] Jacob D Durrant, Steffen Lindert, and J Andrew McCammon. AutoGrow 3.0: an improved algorithm for chemically tractable, semi-automated protein inhibitor design. *Journal of Molecular Graphics and Modelling*, 44:104–112, 2013.

[47] Renxiao Wang, Liang Liu, Luhua Lai, and Youqi Tang. SCORE: A new empirical method for estimating the binding affinity of a protein-ligand complex. *Molecular modeling annual*, 4(12):379–394, 1998.

[48] 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.

[49] 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.

[50] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[51] Oleg Trott and Arthur J Olson. Autodock Vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. *Journal of computational chemistry*, 31(2):455–461, 2010.

[52] Michael M Mysinger, Michael Carchia, John J Irwin, and Brian K Shoichet. Directory of useful decoys, enhanced (DUD-E): better ligands and decoys for better benchmarking. *Journal of medicinal chemistry*, 55(14):6582–6594, 2012.

[53] 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.

[54] Mario Krenn, Florian Häse, AkshatKumar Nigam, Pascal Friederich, and Alan Aspuru-Guzik. Self-referencing embedded strings (SELFIES): A 100% robust molecular string representation. *Machine Learning: Science and Technology*, 1(4):045024, 2020.

[55] Teague Sterling and John J Irwin. ZINC 15–ligand discovery for everyone. *Journal of chemical information and modeling*, 55(11):2324–2337, 2015.

[56] Paul G Francoeur, Tomohide Masuda, Jocelyn Sunseri, Andrew Jia, Richard B Iovanisci, Ian Snyder, and David R Koes. Three-dimensional convolutional neural networks and a cross-docked data set for structure-based drug design. *Journal of Chemical Information and Modeling*, 60(9):4200–4215, 2020.

[57] 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.

[58] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *science*, 324(5923):81–85, 2009.

[59] Yuxuan Du, Tao Huang, Shan You, Min-Hsiu Hsieh, and Dacheng Tao. Quantum circuit architecture search: error mitigation and trainability enhancement for variational quantum solvers. *arXiv preprint arXiv:2010.10217*, 2020.

[60] Greg Landrum et al. RDKit: Open-source cheminformatics, 2006.

[61] 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):8, 2009.

[62] Jan Peters and J Andrew Bagnell. Policy gradient methods. *Scholarpedia*, 5(11):3698, 2010.

[63] Chengxi Zang and Fei Wang. MoFlow: an invertible flow model for generating molecular graphs. In *ACM SIGKDD*, pages 617–626, 2020.

[64] Minkai Xu, Lantao Yu, Yang Song, Chence Shi, Stefano Ermon, and Jian Tang. GeoDiff: A geometric diffusion model for molecular conformation generation. In *International Conference on Learning Representations*, 2021.

[65] Henry Moss, David Leslie, Daniel Beck, Javier Gonzalez, and Paul Rayson. BOSS: Bayesian optimization over string spaces. *Advances in neural information processing systems*, 33:15476–15486, 2020.

[66] Xiufeng Yang, Jinzhe Zhang, Kazuki Yoshizoe, Kei Terayama, and Koji Tsuda. ChemTS: an efficient python library for de novo molecular generation. *Science and technology of advanced materials*, 18(1):972–976, 2017.

[67] Xiufeng Yang, Tanuj Kr Aasawat, and Kazuki Yoshizoe. Practical massively parallel Monte-Carlo Tree Search applied to molecular design. *arXiv preprint arXiv:2006.10504*, 2020.

[68] Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth Stanley, and Jeff Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. *Advances in neural information processing systems*, 31, 2018.

[69] Evan Z Liu, Aditi Raghunathan, Percy Liang, and Chelsea Finn. Decoupling exploration and exploitation for meta-reinforcement learning without sacrifices. In *International Conference on Machine Learning*, pages 6925–6935. PMLR, 2021.

[70] 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.

[71] Youngchun Kwon, Seokho Kang, Youn-Suk Choi, and Inkoo Kim. Evolutionary design of molecules based on deep learning and a genetic algorithm. *Scientific reports*, 11(1):1–11, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 5.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See the supplementary materials.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See the supplementary materials.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See the supplementary materials.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See the supplementary materials.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See the supplementary materials.
   (b) Did you mention the license of the assets? [Yes] See the supplementary materials.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See the supplementary materials.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Contents

1 Introduction **1**

2 Related Works **2**

3 Method **3**
3.1 Evolutionary Process . . . 3
3.2 Evolutionary Markov Decision Process . . . 4
3.3 Target-Ligand Policy Network . . . 5

4 Experiment **7**
4.1 Experimental Setup . . . 7
4.2 Results . . . 8

5 Conclusion **9**

A Mathematical Notation **15**

B Illustration of Genetic Algorithm **16**

C Baseline Setup **17**

D Additional Experimental Setup **19**
D.1 Docking Simulation . . . 19
D.2 Dataset . . . 19
D.3 Evaluation metrics . . . 19

E Implementation Details **20**
E.1 Software/Hardware Configuration . . . 20
E.2 Hyperparameter Setup . . . 20
E.3 Code Repository . . . 20

F Additional Experiment **20**
F.1 Efficiency Study . . . 20
F.2 Pretraining Equivariant neural network . . . 20
F.3 Additional Ablation Study . . . 21
F.4 Scalability . . . 22
F.5 Significance Studies . . . 23
F.6 Example of the generated ligand . . . 23

G Additional Discussion on Related Work **23**

A Mathematical Notation

For ease of exposition, we list the mathematical notations in Table 2. All the mathematical notations are divided into three parts: (1) notation for genetic algorithm (Section 3.1); (2) notation for equivariance neural networks (ENN) [18] (Section 3.3); (3) notations for policy network (Section 3.3).

Table 2: Mathematical Notations. All the mathematical notations are divided into three parts: (1) notation for genetic algorithm (Section 3.1); (2) notation for equivariance neural networks (ENN) [18] (Section 3.3); (3) notations for policy network (Section 3.3).

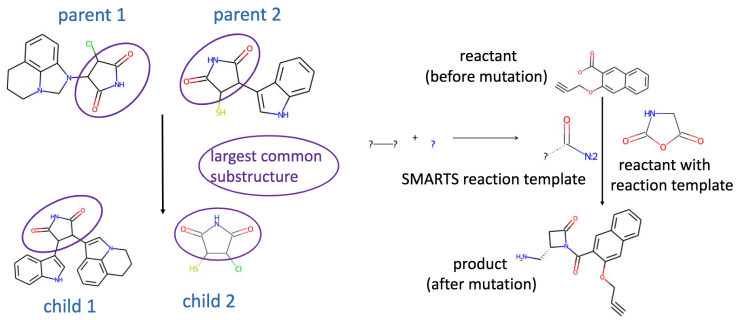
Notations	Descriptions
X	ligand (drug molecule, including 3D pose)
$\mathcal{T}$	target (target protein related to the disease)
$\mathcal{S}^{(t)}$	the state (population of molecule) at the t-th generation.
$\mathcal{Q}^{(t)}$	offspring pool at the t-th generation.
K	the number of molecule in the state, i.e., size of population.
$X_{\text{crossover}}^{\text{parent 1/2}}$	the first/second parent molecule in the crossover.
$X_{\text{crossover}}^{\text{child 1/2}}$	the first/second child molecule in the crossover.
$X_{\text{mutation}}^{\text{parent}}$	parent molecule in the mutation
$X_{\text{mutation}}^{\text{child}}$	child molecule in the mutation
$\xi \in \mathcal{R}$	the selection reaction in the mutation
$\mathcal{R}$	the reaction set (library) for mutation
ENN	equivariance neural networks [18]
$\mathcal{V} = \{H, C, O, N, \cdots\}$	vocabulary set of atoms
$\mathcal{Y} = (\mathcal{A}, \mathcal{Z})$	3D structure
$\mathcal{A}$	categories of all the atoms
$\mathbf{a}_i$	one-hot vector that encode category of i-th atom
$\mathcal{Z}$	3D coordinates of the atoms
$\mathbf{D} \in \mathbb{R}^{\lvert\mathcal{V}\rvert \times d}$	the embedding matrix of all the categories of atoms
d	the hidden dimension in ENN.
N	number of atoms in the input of ENN.
L	number of layers in ENN
$l = 0, 1, \cdots, L$	index of layer in ENN
MLP	multiple layer perceptrons
$\text{MLP}_e(\cdot), \text{MLP}_x(\cdot), \text{MLP}_h(\cdot)$	two-layer MLP in ENN with Swish activation [50] in hidden layer
$\oplus$	the concatenation of vectors
$\mathbf{Z}^{(0)} = \{\mathbf{z}_i\}_{i=1}^N$	initial coordinate embeddings, real 3D coordinates of all the nodes.
$\mathbf{H}^{(l)} = \{\mathbf{h}_i^{(l)}\}_{i=1}^N$	Node embeddings at the l-th layer
$\mathbf{h}_i^{(0)} = \mathbf{D}^\top \mathbf{a}_i \in \mathbb{R}^d$	The initial node embedding that embeds the i-th node
$\mathbf{Z}^{(l)} = \{\mathbf{z}_i^{(l)}\}_{i=1}^N$	Coordinate embeddings at the l-th layer
$\mathbf{w}_{ij}^{(l)}$	message vector for the edge from node i to node j at l-th layer
$\mathbf{v}_i^{(l)}$	message vector for node i at l-th layer
$\mathbf{z}_i^{(l)}$	the position embedding for node i at l-th layer
$\mathbf{h}_i^{(l)}$	the node embedding for node i at l-th layer
$\mathbf{h}_{\mathcal{Y}} = \text{ENN}(\mathcal{Y})$	ENN representation of the 3D graph $\mathcal{Y}$ (Equation 1)
$p_{\text{crossover}}^{(1)}(X_{\text{crossover}}^{\text{parent 1}} \mid \mathcal{S}^{(t)})$	probability to select the first parent molecule in crossover
$p_{\text{crossover}}^{(2)}(X_{\text{crossover}}^{\text{parent 2}} \mid X_{\text{crossover}}^{\text{parent 1}}, \mathcal{S}^{(t)})$	probability to select the second parent molecule in crossover
$p_{\text{crossover}}(X_{\text{crossover}}^{\text{child 1}}, X_{\text{crossover}}^{\text{child 2}} \mid \mathcal{S}^{(t)})$	probability of two generated child molecules in crossover (Eq 2)
$p_{\text{mutation}}^{(1)}(X_{\text{mutation}}^{\text{parent}} \mid \mathcal{S}^{(t)})$	probability to select the parent molecule in mutation
$p_{\text{mutation}}^{(2)}(\xi \mid X_{\text{mutation}}^{\text{parent}}, \mathcal{S}^{(t)})$	probability to select the reaction in mutation
$p_{\text{mutation}}(X_{\text{mutation}}^{\text{child}} \mid \mathcal{S}^{(t)})$	probability of generated child molecule in mutation (Eq 3)

B Illustration of Genetic Algorithm

Figure 3 provides two examples to illustrate crossover and mutation operations in genetic algorithm described in Section 3.1.

Crossover, also called recombination, combines the structure of two parents to generate new children. Following Autogrow 4.0 [16], as shown in Figure 3(a), we select two parents from the last generation and search for the largest common substructure shared between them. Then we generate two children by randomly switching their decorating moieties, i.e., the side chains attached to the common substructure.

Mutation operator performs an *in silico* chemical reaction to generate an altered child compound (i.e., product in the chemical reaction) derived from a parent (reactant in the chemical reaction), as shown in Figure 3(b). The chemical reaction here contains two reactants, one is parent molecule, another is from reaction set. The reaction set $\mathcal{R}$ is generated via merging two public reaction libraries: (1) the AutoClickChemRxn set (36 reactions) [46] and (2) RobustRxn set (58 reactions [49]). Each reaction ξ in reaction set $\mathcal{R}$ contains a SMARTS string based reaction template and a reactant. It uses SMARTS reaction template, together with RDKit [60], to perform chemical mutations efficiently. The process is written as $X^{\text{parent}}_{\text{mutation}} \xrightarrow{\text{mutated by } \xi} X^{\text{child}}_{\text{mutation}}$, where ξ is selected reaction (with a reaction template and another reactant). The ligand to be mutated and the reaction used for mutation are both randomly selected from previous generation and reaction set $\mathcal{R}$, respectively. Compared with the mutation operator in conventional GA that randomly flipping an arbitrary bit, the reaction-based mutation enhance synthesizability of the generated molecules [16]. Mutation operator performs an *in silico* chemical reaction to generate an altered child compound (i.e., product in the chemical reaction) derived from a parent (reactant in the chemical reaction). The chemical reaction here contains two reactants, one is parent molecule, another is from reaction set.



(a) crossover (b) mutation

Figure 3: Illustration of GA operations: (a) **crossover** finds the largest substructure that the two parent compounds share and generates a child by combining their decorating moieties. (b) **mutation**: given a reactant (i.e., parent), mutation operator uses SMARTS-reaction template (with another reactant) to performs an *in silico* chemical reaction to generate child compound (i.e., product).

C Baseline Setup

In this section, we describe the experimental setting for baseline methods. Most of the settings follow the original papers.

- **GA+D** (genetic algorithm enhanced by discriminator neural network) [26] utilizes SELFIES string as the representation of molecules, thus guaranteeing the 100% chemical validity of the generated molecules. Following their original paper, the discriminator neural network is a two-layer fully connected neural network with ReLU activation and sigmoid output layer. The hidden size is 100. the size of the output layer is 1. The input feature of discriminator neural network is a vector of chemical and geometrical properties characterizing the molecules. The population size is set to 300. Maximal generation number is set to 1000. The patience is set to 5. When the property does not improve when the patience exhausts, the process early stops. We used Adam optimizer with 1e-3 as the initial learning rate. beta (β) is the weight of discriminator neural network's score in fitness evaluation, which is used to select most promising molecules in each generation. We set $\beta = 1$.
- **Graph-GA** [15] uses molecular graph to represent molecules and uses crossover and mutation operations to edit the molecular graph. After tuning, the size of population is set to 120. The size of offspring is set to 70. The mutation rate is set to 0.067. Graph-GA do not have learnable parameters and is easy to implement. We use the implementation in GuacaMol [13].
- **MolDQN** (Molecule Deep Q-Networks) [30] uses molecular graph to represent molecules, formulate the molecule optimization process as a Markov Decision Process (MDP) and

utilize Deep Q-value learning to optimize it. It grows molecular graph atom-wise, that is, in each episode, it adds one atom to the partially generated molecular graph. The reward function is the negative value of the Vina score. Following the original paper, maximal step in each episode is 40. Each step calls oracle once. The discount factor is 0.9. Deep Q-network is a multilayer perceptron (MLP) whose hidden dimensions are 1024, 512, 128, 32, respectively. The model size is 6.4 M. The input of the Q-network is the concatenation of the molecule feature (2048-bit Morgan fingerprint, with a radius of 3) and the number of left steps. Adam is used as an optimizer with 1e-4 as the initial learning rate. Only rings with a size of 5 and 6 are allowed. It leverages ϵ-greedy together with randomized value functions (bootstrapped-DQN) as an exploration policy, ϵ is annealed from 1 to 0.01 in a piecewise linear way.

- **RationaleRL** [31]. The architecture of the generator is a message-passing network (MPN) followed by MLPs applied in breadth-first order. The generator is pre-trained on general molecules combined with an encoder and then fine-tuned to maximize the reward function using policy gradient. The encoder and decoder MPNs both have hidden dimensions of 400. The dimension of the latent variable is 50. Adam optimizer is used on both pre-training and fine-tuning with initial learning rates of 1e-3, 5e-4, respectively. The annealing rate is 0.9. We pre-trained the model with 20 epochs.
- **MARS** [35] leverage Markov chain Monte Carlo sampling (MCMC) on molecules with an annealing scheme and an adaptive proposal. The proposal is parameterized by a graph neural network, which is trained on MCMC samples. We follow most of the settings in the original paper. The message passing network has six layers, where the node embedding size is set to 64. Adam is used as an optimizer with 3e-4 initial learning rate. To generate a basic unit, top-1000 frequent fragments are drawn from ZINC database [55] by enumerating single bonds to break. During the annealing process, the temperature $T = 0.95^{\lfloor t/5 \rfloor}$ would gradually decrease to 0.
- **Autogrow 4.0** [16] is the base model for `RGA` and have been briefly described in Section 3.1. The setup of Autogrow is the same as `RGA` for fair comparison, the only difference is that `RGA` use policy network to guide the selection of ligands and reaction for crossover and mutation while Autogrow randomly selects them. The reaction set $\mathcal{R}$ is generated via merging two public reaction libraries: (1) the AutoClickChemRxn set (36 reactions) [46] and (2) RobustRxn set (58 reactions [49]). In each generation, It generates 200 offspring (100 from crossover and 100 from mutation) and keep 50 most promising (with lowest Vina scores) ones for the next generation.
- **Screening** exhaustively searches the ZINC database [55] within oracle budget. It is the traditional high-throughput screening approach.
- **REINVENT** [28] is a reinforcement learning approach, represent molecule as SMILES string and uses recurrent neural network to model SMILES string. It pretrains a prior model using molecules on ZINC and finetune the model using the reward function. It uses REINFORCE to maximize the expected reward function. The learning rate is set to 0.0005; the batch size is set to 64; The hyperparameter σ weighs the pretrained prior model and the reward function, and is set to 60. The model size is 16.3M.
- **JTVAE** [20] build a junction tree to represent molecule via using substructure (either ring or atom) to represent molecule. It uses both molecular graph-leven and junction tree-level encoder and decoder. The VAE model is pretrained on ZINC databases. Then Bayesian Optimization is used to optimize the docking score on the continuous latent space. We use "botorch", the python's Bayesian optimization package, to implement the Bayesian optimization process. It has 703 substructures in vocabulary, extracted from ZINC. The hidden size is 450. The latent size of VAE is set to 56. The model size is 21.8 M.
- **Gen3D** [10] uses 3D deep generative models and grow the molecule via adding atoms auto-regressively. It train a universal model for all the targets. The number of message passing layers in context encoder is 6, and the hidden dimension is 256. We train the model using the Adam optimizer at learning rate 0.0001. The model size is 17.4 M.
- **GEGL** (Genetic Expert Guided Learning) [32] uses LSTM (guided by RL agent) to imitate GA process, however, it is unable to inherit the GA's flexible assembling manner due to the auto-regressive essence of LSTM. It use Adam as optimizer with initial learning rate 1e-3.

The batch size during sampling is 512, the batch size during optimization is 256. In GA, mutation rate is 0.01. The similarity threshold is 0.4, which constrain the similarity between the original molecule and the edited molecule. The maximal SMILES length is set to 120.

D Additional Experimental Setup

D.1 Docking Simulation

Molecular docking is a computational method which predicts the preferred orientation of one molecule to a second when a ligand and a target are bound to each other to form a stable complex. Knowledge of the preferred orientation in turn may be used to predict the strength of association or binding affinity between two molecules using, for example, scoring functions. We adopt AutoDock Vina [51] to evaluate the binding affinity. The docking score estimated by AutoDock Vina is called Vina score and roughly characterizes the free energy changes of binding processes in kcal/mol. Vina score is usually smaller than 0 and lower Vina score means a stronger binding affinity between the ligand and target. We leverage the negative value of the docking score as reward function (Equation 4).

D.2 Dataset

In this paper, we use ZINC [55] database and CrossDocked2020 [56] dataset. ZINC is a free database of 250 thousands commercially-available drug-like chemical compounds for virtual screening [55]. We randomly select molecules from ZINC [55] database (around 250 thousands drug-like molecules) as 0-th generation of the genetic algorithms (`RGA`, Autogrow 4.0, graph-GA GA+D). Other baseline methods also use ZINC to either pretrain the models, e.g., JTVAE, REINVENT, RationaleRL or provide searching database, e.g., screening. We adopt CrossDocked2020 [56] dataset that contains around 22 million ligand-protein complexes as the training data for pretraining the policy neural networks, as mentioned in Section 3.3.

Regarding the target proteins, we picked various disease-related proteins, including G-protein coupling receptors (GPCRs) and kinases from DUD-E [52] and the SARS-CoV-2 main protease [53] as targets. for all the selected target protein, the binding pocket size for all the targets are set to (15.0, 15.0, 15.0). The units of coordinate are Angstrom $\mathring{A}$ (10^{-10} m). Detailed descriptions of these targets are available at `https://www.rcsb.org/`.

D.3 Evaluation metrics

We leverage the following evaluation metrics to measure the optimization performance:

- **Novelty** is the fraction of the generated molecules that do not appear in the training set.
- **Diversity** of generated molecules is defined as the average pairwise Tanimoto distance between the Morgan fingerprints [29, 31, 35].

$$\text{diversity} = 1 - \frac{1}{|\mathcal{M}|(|\mathcal{M}|-1)} \sum_{m_1, m_2 \in \mathcal{M}} \text{sim}(m_1, m_2), \tag{5}$$

  where $\mathcal{M}$ is the set of generated molecules that we want to evaluate. $\text{sim}(m_1, m_2)$ is the Tanimoto similarity between molecule m_1 and m_2, where (Tanimoto) Similarity measures the similarity between the input molecule and generated molecules. It is defined as $\text{sim}(X, Y) = \frac{\text{FP}_X^\top \text{FP}_Y}{\|\text{FP}_X\|_2 \|\text{FP}_Y\|_2}$, FP_X is the binary Morgan fingerprint vector for the molecule X. In this paper, it is a 2048-bit binary vector.
- **QED** represents a quantitative estimate of drug-likeness. QED score ranges from 0 to 1. It can be evaluated by the RDKit package (`https://www.rdkit.org/`).
- **SA** (Synthetic Accessibility) score measures how hard it is to synthesize a given molecule, based on a combination of the molecule's fragments contributions [61]. It is evaluated via RDKit [60]. The raw SA score ranges from 1 to 10. A higher SA score means the molecule is hard to be synthesized and is not desirable.
- **Run Time**. Unlike optimizing some simple oracles such as QED and LogP scores, the docking simulation need to search 3D molecular conformation docked in the target, which is

computationally expensive. Thus run time is an important metric to measure the efficiency of the methods.

E Implementation Details

E.1 Software/Hardware Configuration

We implemented RGA using Pytorch 1.10.2, Python 3.7, RDKit v2020.09.1.0 on an Intel Xeon E5-2690 machine with 256G RAM and NVIDIA Pascal Titan X GPUs.

E.2 Hyperparameter Setup

The neural architectures of policy networks are E(3)-equivariant neural network (ENN) [18]. The vocabulary set $\mathcal{V} = \{C, N, O, S, H, \text{other}\}$. In ENN, the number of layers is set to 3, i.e., $L = 3$; the hidden dimension is set to 100, i.e., $d = 100$. In Equation 1, $\text{MLP}_e(\cdot), \text{MLP}_x(\cdot), \text{MLP}_h(\cdot)$ are two-layer MLP in ENN with Swish activation [50] in hidden layer. Summation function is used as aggregation function to aggregate the last-layer's node embedding into graph-level embedding. All the atoms that within the binding site are used as the input of ENN. REINFORCE is used to implement policy gradient [62, 28]. Adam is utilized as optimizer with learning rate 0.001 for both crossover and mutation policy networks. The reaction set $\mathcal{R}$ is generated via merging two public reaction libraries: (1) the AutoClickChemRxn set (36 reactions) [46] and (2) RobustRxn set (58 reactions [49]). We use RDKit [60] to perform *in silico* chemical reaction based on SMARTS reaction template, then we have $|\mathcal{R}| = 36 + 58 = 94$. In each generation, we generate up to 200 offspring (100 from crossover and 100 from mutation) and keep 50 most promising (with lowest Vina scores) ones for the next generation, i.e., $K = 50$.

E.3 Code Repository

The code repository is uploaded in supplementary material for reproducibility.

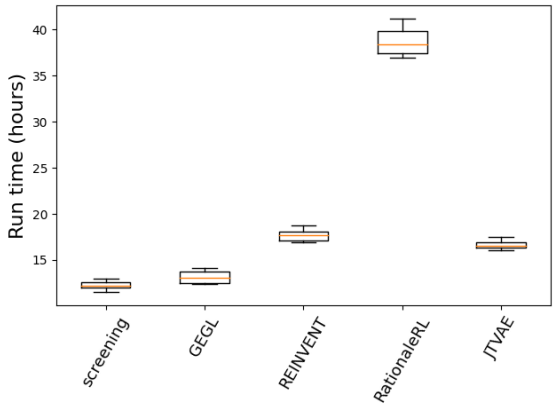
F Additional Experiment

F.1 Efficiency Study

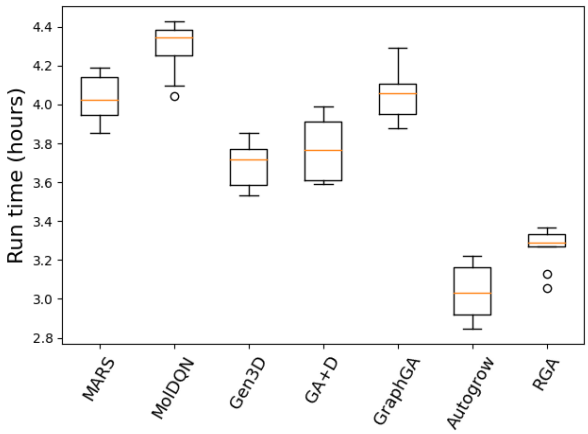
As mentioned before, unlike optimizing some simple oracles such as QED and LogP scores, the docking simulation need to search 3D molecular conformation docked in the target, which is time-consuming. Thus run time is an important measurement to evaluate the efficiency of the methods. We report the bar of run time over different targets for all the compared methods in Figure 4. The run times varies greatly over different methods. Thus, for ease of visualization, we divided all the methods into two groups. One is slow group, containing 5 methods: screening, GEGL, REINVENT, RationaleRL and JTVAE, where all the methods take more than 10 hours. Another is fast group (<10 hours), containing 7 methods, MARS, MolDQN, Gen3D, GA+D, GraphGA, Autogrow, and RGA. We find that both Autogrow and RGA are efficient compared with other methods. This attributes to the unique design of genetic algorithm (both crossover and mutation operations) and the usage of filter after GA operators, as described in Section 3.1. RGA is only slightly slower than Autogrow because it requires additional computation to pretrain/train the policy neural networks.

F.2 Pretraining Equivariant neural network

Pretraining equivariant neural network is a crucial step to RGA. We split the whole dataset into training/validation dataset with ratio of 9:1. Each data point is a target-ligand complex and the binding affinity (scalar). We report the validation loss of ENN on target-ligand binding affinity prediction task. The validation loss function is root mean square error (RMSE). We find the learning process converges rapidly when passing 150K data points (within an epoch) in terms of validation MSE loss.



(a) Slower Group (>10 hours)



(b) Fast Group (<10 hours)

Figure 4: Efficiency evaluation measured by run time for all the methods. The unit of run time is hours. Due to the big variance in run time, for ease of visualization, we divide all the methods into two groups. One is slow group, containing 5 methods: screening, GEGL, REINVENT, RationaleRL and JTVAE. Another is fast group, containing 7 methods, MARS, MolDQN, Gen3D, GA+D, GraphGA, Autogrow, and RGA.

F.3 Additional Ablation Study

To further understand our model and GA process, we conduct an ablation study to investigate the impact of each component/strategy to optimization performance. Specifically, we consider the following four variants of RGA. `RGA`-pretrain is a variant of `RGA` that does not pretrain the policy neural network. `RGA`-KT (Knowledge Transfer) is a variant of `RGA` that does not training policy neural network on different target proteins, i.e., optimizing ligand for one target at a time. `RGA`-MU (mutation) is a variant of `RGA` that does not involve mutation operation in GA. That is, all the ligands are generated via crossover operator. Correspondingly, `RGA`-CO (crossover) is a variant that does not use crossover operation in GA, which means no mutation operator. The results are reported in Table 3. We find that removing either component/strategy will cause a drop in optimization performance (i.e., increase in TOP-100/10/1 scores). Both crossover and mutation are critical to the optimization performance. Also, both pretraining the policy networks and knowledge transfer between different target have positive contribution to the performance. The ablation study furtherly validates the effectiveness of the proposed RGA method.

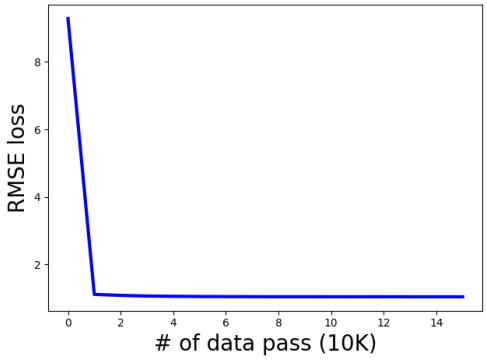


Figure 5: Learning curve of pretraining Equivariant neural network based on target-ligand binding affinity prediction. We plot the root-mean-square error (RMSE) loss as a function of number of passed data. We use early stop strategy to terminate the learning process earlier when the validation loss would not decrease to save computational resource and avoid overfitting. We found the learning process would converges when passing 150K data samples (within an epoch), RMSE loss decreases from more than 8 to less than 1.

Table 3: Ablation studies. Arrows (↑, ↓) indicate the direction of better performance. For each metric, the best method is underlined. RGA-pretrain is a variant of RGA that does not pretrain the policy neural network. RGA-KT (Knowledge Transfer) is a variant of RGA that does not training policy neural network on different target proteins, i.e., optimizing ligand for one target at a time. RGA-MU (mutation) is a variant of RGA that does not involve mutation operation in GA. That is, all the ligands are generated via crossover operator. Correspondingly, RGA-CO (crossover) is a variant that does not use crossover operation in GA, which means no mutation operator. Via comparing the results with RGA (full) in the first line, we observe that removing either component would cause a drop in optimization performance (i.e., increase in TOP-100/10/1 scores).

Method	TOP-100↓	TOP-10↓	TOP-1↓	Nov↑	Div↑	QED↑	SA↓
RGA (full)	-11.867±0.170	-12.564±0.287	-12.869±0.473	100.0±0.0%	0.857±0.020	0.742±0.036	2.473±0.048
RGA - pretrain	-11.443±0.219	-12.424±0.386	-12.435±0.654	100.0±0.0%	0.854±0.035	0.750±0.034	2.494±0.043
RGA - KT	-11.434±0.169	-12.437±0.354	-12.502±0.603	100.0±0.0%	0.853±0.028	0.738±0.034	2.501±0.050
RGA - MU	-10.919±0.166	-11.135±0.362	-11.747±0.455	100.0±0.0%	0.812±0.032	0.702±0.050	2.970±0.048
RGA - CO	-9.866±0.169	-10.320±0.296	-10.793±0.501	100.0±0.0%	0.737±0.048	0.748±0.067	2.467±0.034

F.4 Scalability

As mentioned before, the population size in RGA is K. Then we analyze the computational complexity within each generation. There are two operations, including crossover and mutation operations, as described in Section 3.3.

For crossover, the first step is to select the first parent molecule, we need to evaluate the probability over all the molecules in current population, whose complexity is $O(K)$. The second step is to select the second parent molecule based on the first parent molecule, we need to evaluate the probability over all the remaining molecules in current population, as shown in Equation (2), whose complexity is $O(K-1)$. The complexity of crossover operation is $O(K)$.

On the other hand, for mutation operation, the first step is to select the parent molecule (only one parent) via evaluating the probability over all the molecules in the current population, i.e., $p^{(1)}_{\text{mutation}}(X^{\text{parent}}_{\text{mutation}}|\mathcal{S}^{(t)})$, whose complexity is $O(K)$. The second step is to select a mutated molecules (generated by chemical reaction), as shown in Equation (3). The complexity is $O(|\mathcal{R}|)$, where $\mathcal{R}$ is the reaction set (see Table 2). The complexity of mutation operation is $O(|\mathcal{R}|+K)$.

To summarize, the complexity of RGA is $O(K+|\mathcal{R}|)$. That is, RGA scales linearly in population size K and the size of the chemical reaction set $|\mathcal{R}|$. As mentioned in Section E.2, $|\mathcal{R}|=94$, $K=50$. Thus, RGA owns desired scalability and is not computational expensive.

Table 4: Results of hypothesis testing. We conduct hypothesis testing to show the statistical significance of our method over other GA-based methods. Specifically, we compare the significance of the improvement over GA methods (AutoGrow4, Graph-GA, GA+D, GEGL) via running 5 independent trials with different random seeds and then evaluating p-value. We consider top-100/10/1 scores, the most important metrics for optimization performance.

	p-value on TOP-100	p-value on TOP-10	p-value on TOP-1
RGA v.s. AutoGrow 4	0.002	0.005	0.07
RGA v.s. Graph-GA	0.003	0.010	0.046
RGA v.s. GA+D	1.0e-7	5.0e-5	3e-4
RGA v.s. GEGL	2.5e-4	3.7e-3	5.0e-3

F.5 Significance Studies

In this section, we present the significance studies. Specifically, we conduct the hypothesis testing to show the statistical significance of our method over the other GA methods. We compare the significance of the improvement over other GA methods (including AutoGrow4, Graph-GA, GA+D, and GEGL) via running 5 independent trials with different random seeds and then evaluating the **p-value**. We consider the top-100 and top-10 score as the major metrics, which are the most important metrics for optimization performance. We compare RGA versus AutoGrow 4, Graph-GA and GA+D. The results (p-value) are shown in Table 4. We find almost all the p-values are less than 0.05 (except one value), which indicates that the improvements of RGA over other GA methods are statistically significant.

F.6 Example of the generated ligand

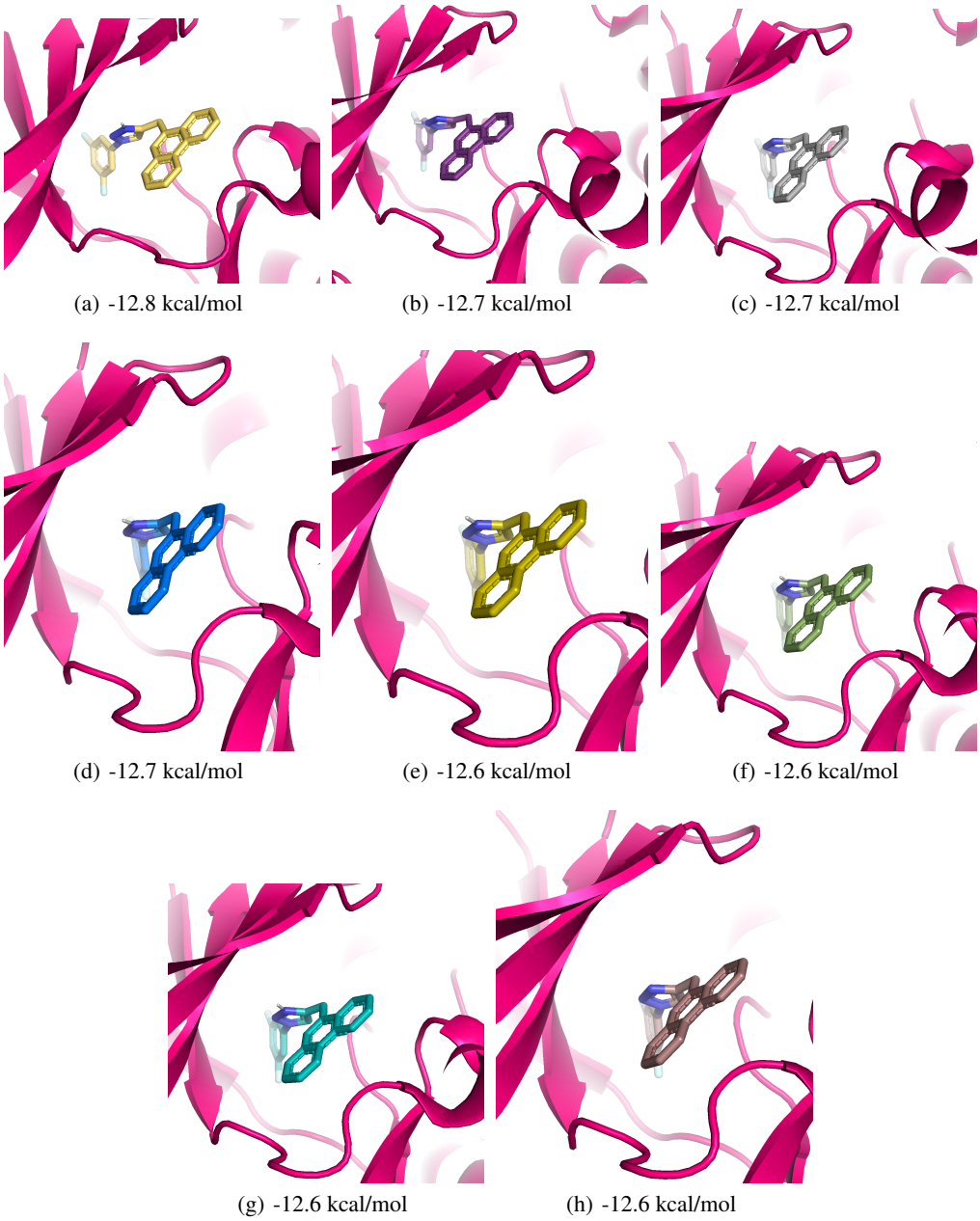
This section shows some examples of the generated ligands with desirable binding affinity for various target proteins in Figure 6, 7, 8 and 9, respectively. We observe that the generated ligands bind tightly with the target proteins.

G Additional Discussion on Related Work

Methodology. The molecule generations methods can be divided into two categories. The first one is deep generative models (DGMs), which leverage the continuous representation to estimate the data distribution using various kinds of deep neural networks, including variational autoencoder (VAE) [19, 20], generative adversarial network (GAN) [21, 22], normalizing flow model [23, 63, 24], energy based model [25] and diffusion model [64], etc. The second one is combinatorial optimization methods, which directly search the discrete chemical space, including genetic algorithm (GA) [15, 26, 27], reinforcement learning approaches (RL) [28, 29, 30, 31], Bayesian Optimization (BO) [33, 65], Monte Carlo Tree Search (MCTS) [66, 67, 31] and Markov Chain Monte Carlo (MCMC) [34, 35, 36]. Deep generative models (DGMs) usually require a large amount of data to fit, which impedes their usage in low data regime. In contrast, combinatorial optimization methods require less training data, while the trade-off is the need to call the optimization oracles during the exploration in the chemical space [30, 37, 48, 27, 13].

Among all the machine learning methods, Genetic algorithm (GA) exhibits superior performance in some standard benchmarks [13, 14]. The key reason is GA's global assembling strategy. Specifically, in each generation (iteration), GA maintains a population of molecule candidates (a.k.a. parents), and conducts the crossover operation between two (random-selected) parent candidates, which enables relatively large exchanges on molecular sub-graph between molecular graphs. However, GA is leveraging random-walk based mutation and crossover operations [15, 16] and is essentially based on brute-force trial and error.

On the other hand, Reinforcement learning (RL) methods are good at navigating the discrete space via prioritizing the promising searching branches and circumventing brute-force search. The current RL-based methods [28, 29, 30, 31] slightly left behind other state-of-the-art combinatorial optimization methods [37, 14]. The main reason is that current RL based molecule optimization approaches are based on auto-regressive assembling strategy, i.e., growing molecules iteratively via adding a basic



(a) -12.8 kcal/mol (b) -12.7 kcal/mol (c) -12.7 kcal/mol

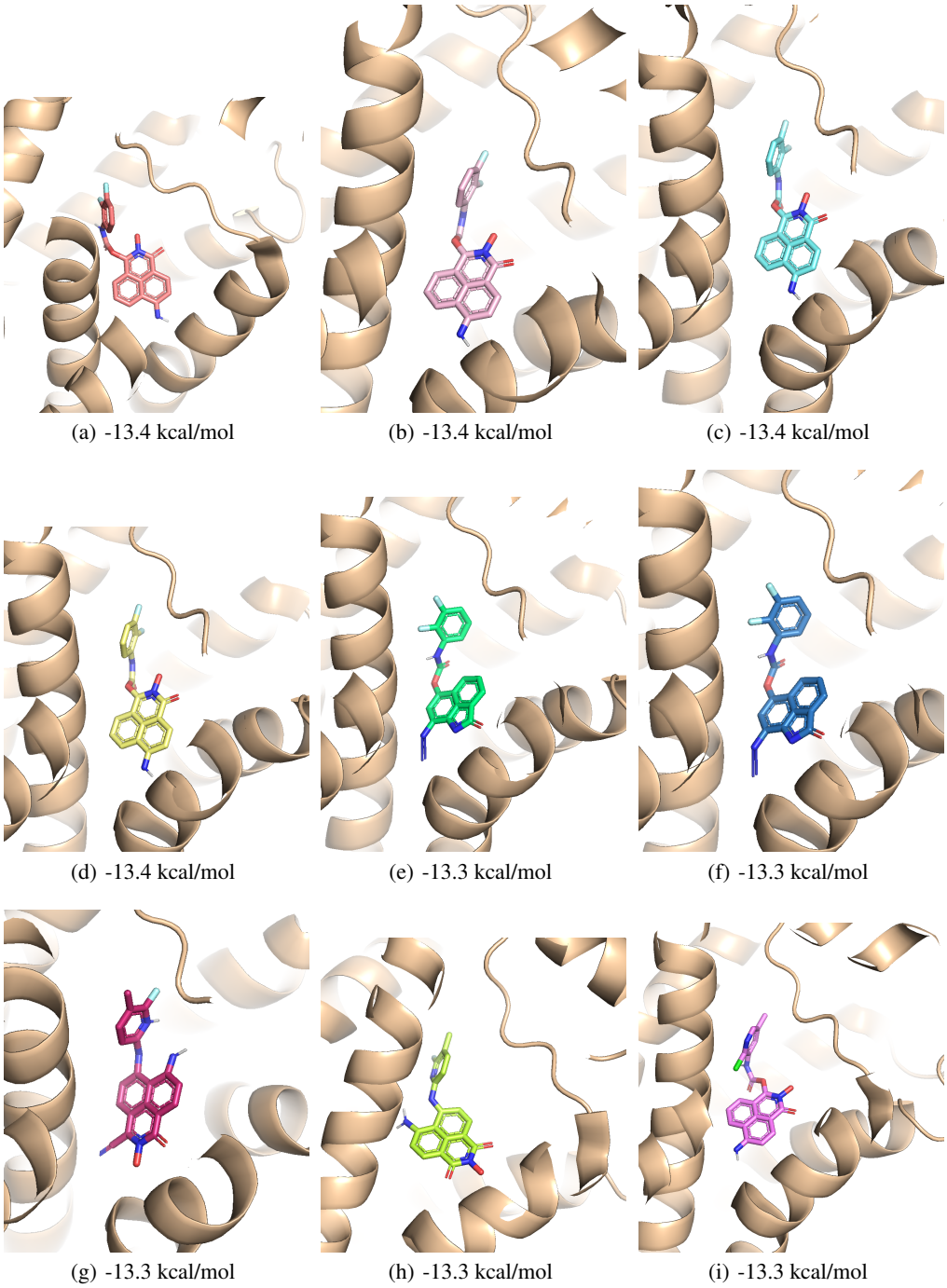
(d) -12.7 kcal/mol (e) -12.6 kcal/mol (f) -12.6 kcal/mol

(g) -12.6 kcal/mol (h) -12.6 kcal/mol

Figure 6: Example of ligand poses (generated by RGA) and binding sites of target structures "2rgp".

building block one time, where the building block can be either a token in SMILES representation [28] or a substructure in molecular graph representation [29, 30, 31]. Such assembling strategy are essentially local search methods, which hinders the algorithm's ability to overcome the rough optimization landscape (or energy barrier) and is easy to be stuck in the local optimum [68, 69].

Discussion. Among all the machine learning methods, molecular graph level genetic algorithm (GA) exhibits state-of-the-art performance in some standard molecule optimization benchmarks [13, 14]. The key reason is GA's assembling manner. Specifically, in each generation (iteration), GA maintains a population of possible candidates (a.k.a. parents), and conducts the crossover operation between two candidates to generate new offspring, which enables thorough exploration to the chemical space. However, there is still improvement space for GA. GA are leveraging random-walk based mutation and crossover operations [15] and suffers from brute-force trial and error strategy.



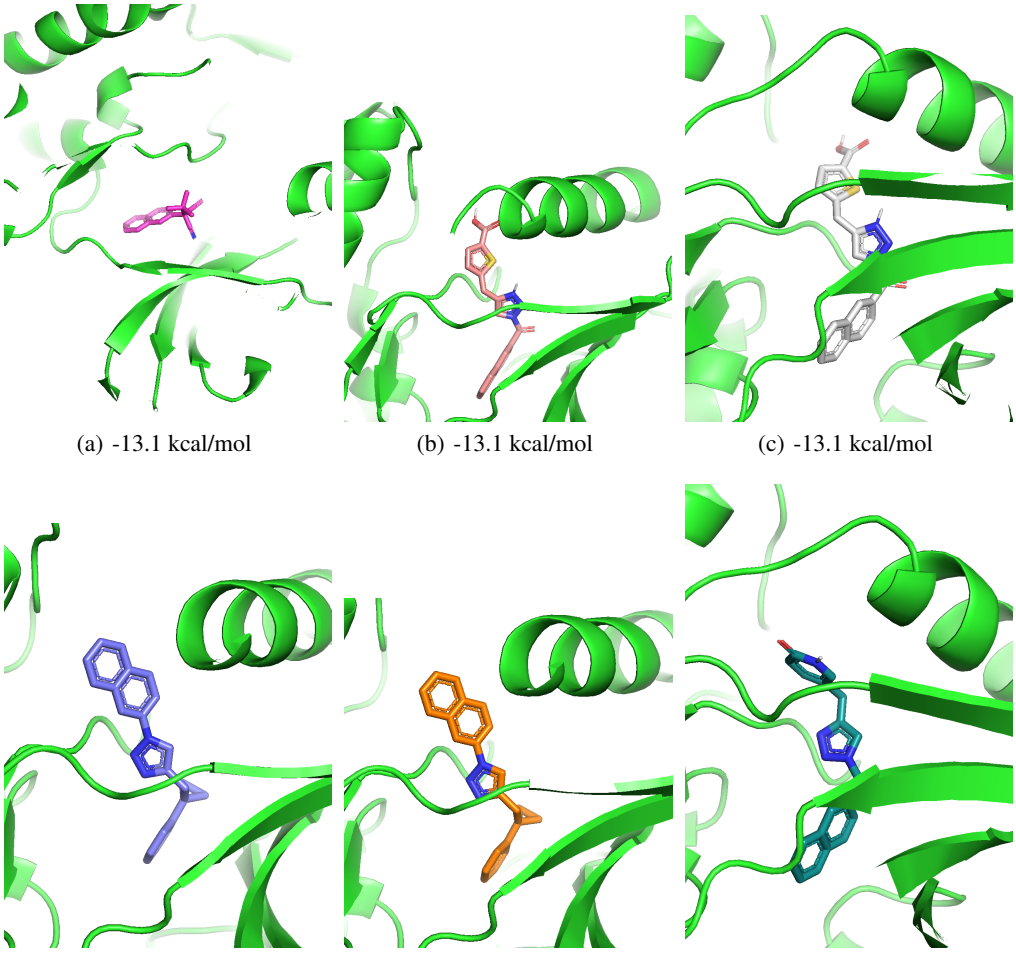
(a) -13.4 kcal/mol (b) -13.4 kcal/mol (c) -13.4 kcal/mol

(d) -13.4 kcal/mol (e) -13.3 kcal/mol (f) -13.3 kcal/mol

(g) -13.3 kcal/mol (h) -13.3 kcal/mol (i) -13.3 kcal/mol

Figure 7: Example of ligand poses (generated by RGA) and binding sites of target structures "3ny8".

On the other hand, reinforcement learning approaches are good at navigating the discrete space via prioritizing the promising decisions that are worth investigating, for example, AlphaGo successfully applied RL to defeat a professional human Go player [70]. However, the current RL based drug design methods [28, 29, 30] slightly left behind other state-of-the-art combinatorial optimization methods. The main reason lies at the inferior assembling strategy, which grows molecule in an auto-regressive fashion. It is hard for this kind of local search strategy to overcome the barrier of the objective, so it is easy to be trapped into the local optimum.



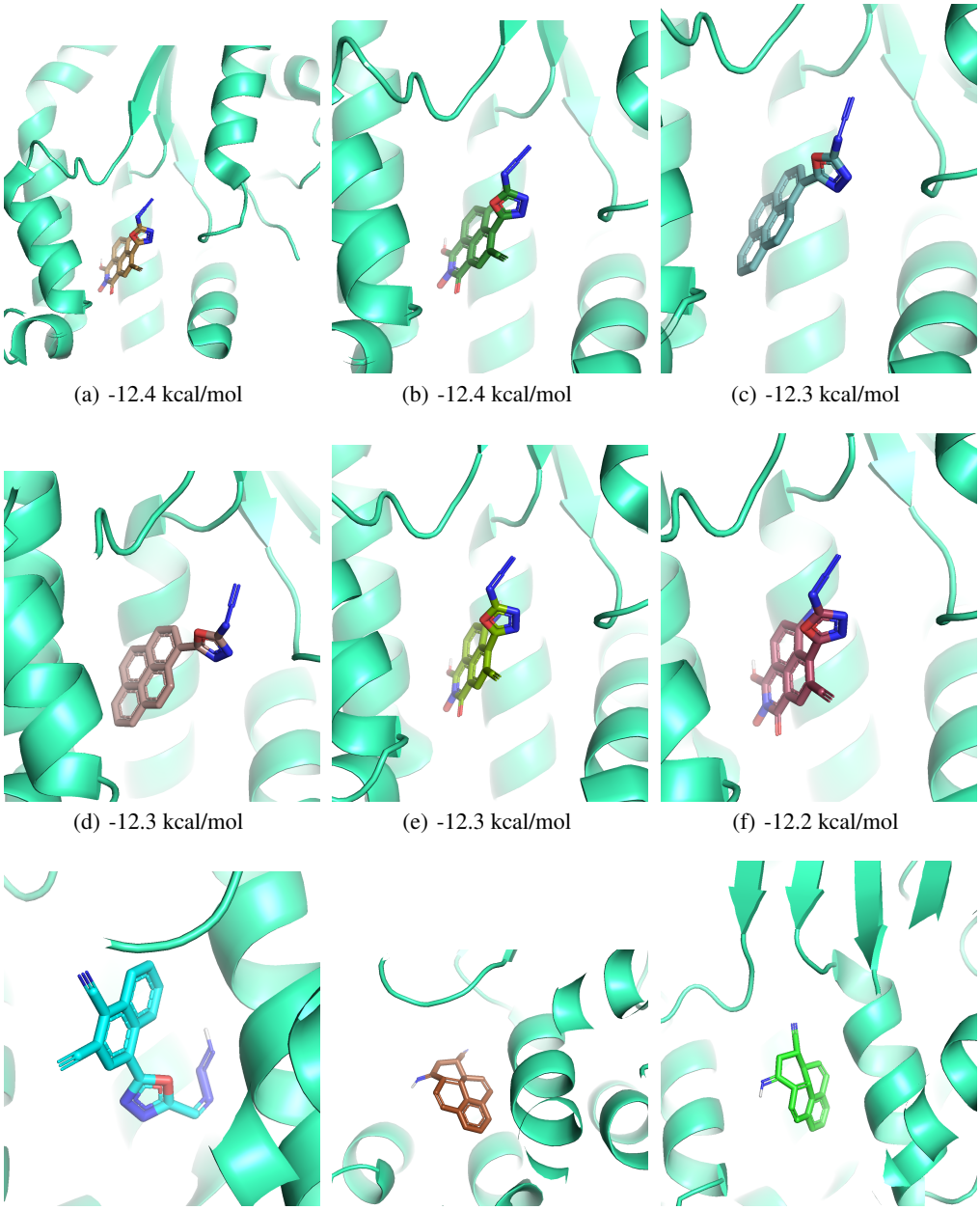
(a) -13.1 kcal/mol (b) -13.1 kcal/mol (c) -13.1 kcal/mol

(d) -13.0 kcal/mol (e) -12.9 kcal/mol (f) -12.8 kcal/mol

Figure 8: Example of ligand poses (generated by RGA) and binding sites of target structures "1iep".

Deep learning methods can also enhance genetic algorithm. Due to the random selection used in genetic algorithm, it is challenging to apply deep learning methods in the generation of new candidates (molecules in this paper). Deep learning can be used to compose fitness evaluation to select the offspring. For example, GA+D [26] leverages deep neural network as a discriminator to measure the drug's proximity to the training data, which is incorporated as a scorer in fitness evaluation. [71] train a deep neural network-based property predictor and leverage it to enhance the evolutionary algorithm.

In this paper, we attempt to enhance genetic algorithm using reinforcement learning technique. Specifically, we propose Reinforced Genetic Algorithm (RGA), which inherits the assembling manner from genetic algorithm and use reinforcement learning to guide the search over the chemical space. [32] also combine RL and GA, which uses LSTM (guided by RL agent) to imitate GA process, however, it is unable to inherit the GA's flexible assembling manner due to the auto-regressive essence of LSTM.



(a) -12.4 kcal/mol (b) -12.4 kcal/mol (c) -12.3 kcal/mol

(d) -12.3 kcal/mol (e) -12.3 kcal/mol (f) -12.2 kcal/mol

(g) -12.2 kcal/mol (h) -12.1 kcal/mol (i) -12.1 kcal/mol

Figure 9: Example of ligand poses (generated by RGA) and binding sites of target structures "4unn".