Biases in Evaluation of Molecular Optimization Methods and Bias Reduction Strategies

Hiroshi Kajino [1] **Kohei Miyaguchi** [1] **Takayuki Osogami** [1]

Abstract

We are interested in an evaluation methodology for molecular optimization. Given a sample of molecules and their properties of our interest, we wish not only to train a generator of molecules optimized with respect to a target property but also to evaluate its performance accurately. A common practice is to train a predictor of the target property using the sample and apply it to both training and evaluating the generator. However, little is known about its statistical properties, and thus, we are not certain about whether this performance estimate is reliable or not. We theoretically investigate this evaluation methodology and show that it potentially suffers from two biases; one is due to misspecification of the predictor and the other to reusing the same finite sample for training and evaluation. We discuss bias reduction methods for each of the biases, and empirically investigate their effectiveness.

1. Introduction

Molecular optimization aims to generate novel molecules with improved properties. This problem has been approached by a combination of a variational autoencoder and Bayesian optimization (Gómez-Bombarelli et al., 2018) or reinforcement learning (Olivecrona et al., 2017). Such an algorithm outputs a *generator*, which can generate desired molecules. The performance of each algorithm can be measured by the quality of molecules generated by the generator; some define the performance metric by top-k properties of the generated molecules, and others may define it by the average property of the generated molecules.

Let $\mathcal{M}$ denote the set of molecules. If we have access to the *property function* $f^\star\colon \mathcal{M} \to \mathbb{R}$, which outputs the true property of the input molecule, it is rather trivial to estimate the performance metric. However, when the target property is costly to evaluate, which may involve wet-lab experiments, we do not have access to $f^\star$, and the performance estimation becomes harder. In such a case, we instead assume that we have a dataset, $\mathcal{D} = \{(m_n, f^\star(m_n)) \in \mathcal{M} \times \mathbb{R}\}_{n=1}^N$, and we often substitute a predictor $\hat{f}\colon \mathcal{M} \to \mathbb{R}$ trained on $\mathcal{D}$ for $f^\star$, which allows us to estimate the performance metric. We call the resultant performance estimator a *plug-in performance estimator*, deriving from the plug-in principle in statistics. For example, Li et al. (2018) and Jin et al. (2020) use predictors for JNK3 and GSK-3β activities; many of the existing studies use the $\log \mathrm{P}$ predictor developed by Wildman & Crippen (1999). In contrast, little attention has been paid to how reliable such estimators are. A few exceptions are empirical studies on it (Renz et al., 2019; Langevin et al., 2022), although we have not reached any consensus. Therefore, in this paper, we investigate the reliability of the plug-in performance estimator from a theoretical perspective and discuss approaches to improve it.

Our first contribution is to show that the plug-in performance estimator is biased in two ways, indicating that it is not reliable in general (Section 3.1). The first bias called a *misspecification bias* comes from the deviation between $\hat{f}$ and $f^\star$ on the molecules discovered by the learned generator. This bias is closely related to the one encountered in covariate shift (Shimodaira, 2000). It grows as the molecules discovered by the generator become dissimilar to those in $\mathcal{D}$. The second bias called a *reuse-and-finiteness bias* is caused by reusing a finite dataset for training and testing the generator. In fact, $\hat{f}$, which is dependent on $\mathcal{D}$, is used to train the generator *and* to estimate the performance metrics. Due to these biases, the plug-in performance estimator is not necessarily a good estimator for the true performance.

Our second contribution is to systematically discuss how to reduce these two biases. Section 4.1 introduces three approaches to reducing the misspecification bias. Since the misspecification bias is caused by covariate shift, a natural idea is to reduce it by applying covariate shift adaptation methods when learning the predictor (Section 4.1.1). In addition to it, we find that we can reduce it by constraining the generator so that the generated molecules become similar

[1]IBM Research – Tokyo, Tokyo, Japan. Correspondence to: Hiroshi Kajino <kajino@jp.ibm.com>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

to those in the dataset $\mathcal{D}$ (Section 4.1.2) in exchange for the performance, which has not been discussed in the literature of covariate shift adaptation. Yet another approach is to use a more sophisticated estimator called a doubly-robust performance estimator (Section 4.1.3).

Our idea to correct the reuse-and-finiteness bias comes from the analogy to information criteria (Konishi & Kitagawa, 2007), whose main objective is to estimate the test performance by correcting the bias of the training performance, which is computed by reusing the same dataset for training and testing. Given the analogy, one may consider train-test split allows us to reduce the bias; we may split the data into training and test sets, learn two independent predictors from them, and use one for training a generator and the other for evaluating it. We however argue that it is not as effective as that applied to supervised learning due to the key difference between our setting and supervised learning; the test set in supervised learning is used to take expectation, while that in our setting is used to train another predictor, which is much more complex than expectation. This complexity introduces a non-negligible bias to the train-test split estimator, resulting in a less accurate bias estimation (Section 4.2.1). We instead propose to use a bootstrap method in Section 4.2.2, which is proven to estimate the bias more accurately than the train-test split method.

We empirically validate our theoretical findings in Section 5. First, we quantify the two biases, and confirm that both are non-negligible, and the reuse-and-finiteness bias increases as the sample size decreases, as predicted by our theory. Second, we assess the effectiveness of the bias reduction methods, and confirm that the reuse-and-finiteness bias can be corrected, while the reduction of the misspecification bias comes at the cost of performance degradation.

Notation. For any set $\mathcal{X}$, we let $\mathcal{P}(\mathcal{X})$ be a set of probability distributions defined over $\mathcal{X}$. For any distribution P, let $\hat{P} \sim P^N$ denote the empirical distribution of a sample of N items independently drawn from P. For a notational reason, we use the empirical distribution of the sample, $\hat{P} \in \mathcal{P}(\mathcal{M} \times \mathbb{R})$, instead of the sample $\mathcal{D}$ itself, and we call $\hat{P}$ an empirical distribution and a sample interchangeably. For a set $\mathcal{X}$, let δ_x be Dirac's delta distribution at $x \in \mathcal{X}$. For any integer $N \in \mathbb{N}$, let $[N] \coloneqq \{0, \dots, N-1\}$. We use uppercase letters for random variables and lowercase letters for their realizations.

2. Problem Formulation

We first present our abstract formulation of molecular optimization. Since there are a number of different problem settings with different objectives and algorithms, it is necessary to abstract them into a single mathematical formulation so that we can theoretically analyze it without much loss of generality. We define the molecular optimization problem as Problem 2.1. In the following, let us discuss problem settings and algorithms that our formulation covers.

Problem 2.1. *Let $u\colon \mathbb{R} \to \mathbb{R}$ be a utility function. The molecular optimization aims to find a* generator $G \in \mathcal{P}(\mathcal{M})$ *such that the following performance metric is maximized:*

$$J^\star(G) \coloneqq \mathbb{E}_{M \sim G}\left[u\left(f^\star(M)\right)\right]. \tag{1}$$

2.1. Molecular Optimization Algorithms

A molecular optimization algorithm is defined as Definition 2.2. In this section, we discuss two major approaches in molecular optimization fit in Definition 2.2.

Definition 2.2. A molecular optimization algorithm is defined as a mapping α_G that receives a sample $\hat{P} \in \mathcal{P}(\mathcal{M} \times \mathbb{R})$ and a property function $f^\star\colon \mathcal{M} \to \mathbb{R}$, and returns a generator $G \in \mathcal{P}(\mathcal{M})$.

2.1.1. Variational Autoencoder Approach

The molecular optimization problem has been tackled by a combination of variational autoencoder (VAE) and Bayesian optimization (Gómez-Bombarelli et al., 2018). It first trains VAE (Kingma & Welling, 2014) on a sample $\hat{P}$ to obtain a pair of encoder and decoder, which translates between a molecule $m \in \mathcal{M}$ and its latent vector $\mathbf{z} \in \mathbb{R}^{D_H}$. Then, it constructs a dataset $\{(\mathbf{z}_n, y_n) \in \mathbb{R}^{D_H} \times \mathbb{R}\}_{n=1}^N$ from a sample $\hat{P} = \{(m_n, y_n) \in \mathcal{M} \times \mathbb{R}\}_{n=1}^N$ using the encoder, and applies Bayesian optimization to obtain promising latent vectors $\mathbf{z}_k^\star \in \mathbb{R}^{D_H}$ by interacting with $f^\star$ for each iteration $k = 1, 2, \dots$. The latent vectors are converted into molecules $m_k \in \mathcal{M}$ by the decoder at each iteration of Bayesian optimization.

If we regard each molecule m_k as a sample from the generator, the VAE-based algorithm does not exactly fit in Definition 2.2, because each sampling is not independent; if its dependency is weak, then it approximately fits in Definition 2.2. Suppose that the algorithm outputs the best molecule at the final step of Bayesian optimization, and we repeat the procedure multiple times independently. Then, the algorithm exactly fits in Definition 2.2. Therefore, our abstraction can model the VAE approach.

2.1.2. Reinforcement Learning Approach

Another approach initiated by Olivecrona et al. (2017) and others uses reinforcement learning (RL) to generate and optimize molecules. Let us first introduce a typical Markov decision process (MDP) of length $H+1$ ($H \in \mathbb{N}$), used to construct a molecule. Let $\mathcal{S}$ be a set of states, and $s_\perp \in \mathcal{S}$ be the terminal state. Let $\mathcal{M} \subseteq \mathcal{S}$ be a subset of states that correspond to molecules and the rest of the states correspond to incomplete representations of molecules. Let $\mathcal{A}$ be a set of actions that transform a possibly incomplete molecule

into another one. There exists the terminal action $a_\perp \in \mathcal{A}$ that evaluates the property of the molecule at step H, after which the state transits to the terminal state $s_\perp$. We assume that the set of states at step H is limited to $\mathcal{M}$, indicating that the property evaluation is applied only to valid molecules. For $h \in [H+1]$, let $T_h \colon \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ be a state transition distribution at step h. Let $r_h \colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ be a reward function at step h. Let $\rho_0 \in \mathcal{P}(\mathcal{S})$ be the initial state distribution. Let Π be the set of policies and $\pi = \{\pi_h(a \mid s)\}_{h=0}^H \in \Pi$ be a policy modeled by a probability distribution over $\mathcal{A}$ conditioned on $s \in \mathcal{S}$. At each step $h \in [H+1]$, the agent takes action a_h sampled from $\pi_h(\cdot \mid s_h)$. Let $p_h^\pi \in \mathcal{P}(\mathcal{S})$ be the distribution of states reached by policy π at step $h \in [H+1]$. We assume that we know $\{\mathcal{S}, \mathcal{A}, \{T_h\}_{h=0}^H, \rho_0, H\}$.

The RL-based algorithm receives $f^\star$ and trains the policy so as to maximize the expected cumulative reward, where the reward is usually defined using $f^\star$, *e.g.*,

$$r_h(s,a) = \begin{cases} 0 & h = 0, 1, \ldots, H-1 \\ f^\star(s) & h = H, a = a_\perp, s \in \mathcal{M}. \end{cases}$$

The trained policy can be regarded as a generator, because each independent episode generated by interacting the policy with the environment yields one molecule.

2.2. Performance Metrics

Next, let us discuss that many existing performance metrics can be approximately or exactly represented as Equation (1).

The most naive metric is the average property of the generated molecules, which can be modeled by Equation (1) if we use the identity function for u. This metric can be, for example, used to obtain a generator of molecules that satisfy a certain set of conditions (*e.g.*, Lipinski's rule of five), by defining $f^\star$ as Equation (2). Such a generator is useful as an efficient alternative to a virtual library and virtual screening.

$$f^\star(m) = \begin{cases} 1 & \text{if } m \text{ satisfies the conditions,} \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Another metric used in the literature focuses on top-k scores of generated molecules. By setting $u(x) = \exp(Cx)$ for large $C > 0$, we obtain a soft top-k metric, because such a utility function puts more weights on top-performing molecules. In fact, in the limit of $C \to \infty$, the performance metric prefers the generator that can generate the best molecule, regardless of the quality of the rest of the generated molecules. In the following, we set u to be the identity function for simplicity.

2.3. Plug-in Performance Estimator

Finally, let us formally define the plug-in performance estimator. Since the estimator is defined using a predictor, we first provide our definition of a learning algorithm for a predictor in Definition 2.3.

Definition 2.3. A learning algorithm for a predictor is represented as a mapping α_f that receives $\hat{P}$ and returns predictor $\hat{f} \colon \mathcal{M} \to \mathbb{R}$.

The plug-in performance estimator is derived by substituting $\hat{f}$ for $f^\star$ when learning and evaluating a generator. Let us define the predictor learned from $\hat{P}$ as $\hat{f} = \alpha_f(\hat{P})$ and the generator learned from $\hat{f}$ and $\hat{P}$ as $\hat{G} \coloneqq \alpha_G(\hat{P}, \alpha_f(\hat{P}))$. Let us also define the *plug-in performance function* J_{PI} as,

$$J_{\mathrm{PI}}(G, f) \coloneqq \mathbb{E}_{M \sim G} f(M), \tag{3}$$

for any generator G and predictor f. Then, the plug-in performance estimator[1] is $J_{\mathrm{PI}}(\hat{G}, \hat{f})$.

3. Biases of Plug-in Performance Estimator

Let us investigate the bias of $J_{\mathrm{PI}}(\hat{G}, \hat{f})$. We point out that it is biased in two ways (Section 3.1) and theoretically characterize these biases in Sections 3.2 and 3.3.

3.1. Bias Decomposition

We define the bias as $\mathbb{E}_{\hat{P} \sim P^N}[J_{\mathrm{PI}}(\hat{G}, \hat{f}) - J^\star(\hat{G})]$ and investigate its statistical properties. First, we will show that it can be decomposed into two terms as Theorem 3.1. The *reuse-and-finiteness bias* is caused by reusing the finite sample $\hat{P}$ to train both the generator and the predictor, and the *misspecification bias* is caused by model misspecification of the predictor. The latter appears when the predictor trained by the infinite sample, $f^\infty \coloneqq \alpha_f(P)$, does not coincide with $f^\star$.

Theorem 3.1. *The bias is decomposed as follows:*

$$\begin{aligned} &\mathbb{E}_{\hat{P} \sim P^N}\left[J_{\mathrm{PI}}(\hat{G}, \hat{f}) - J^\star(\hat{G})\right] \\ =& \underbrace{\mathbb{E}_{\hat{P} \sim P^N}\left[J_{\mathrm{PI}}(\hat{G}, \hat{f}) - J_{\mathrm{PI}}(\hat{G}, f^\infty)\right]}_{\textit{reuse-and-finiteness bias}} \\ &+ \underbrace{\mathbb{E}_{\hat{P} \sim P^N}\left[J_{\mathrm{PI}}(\hat{G}, f^\infty) - J^\star(\hat{G})\right]}_{\textit{Misspecification bias}}. \end{aligned} \tag{4}$$

3.2. Misspecification Bias

The inner term of the squared misspecification bias is upper-bounded by Jensen's inequality as,

$$\begin{aligned} &\left(J_{\mathrm{PI}}(\hat{G}, f^\infty) - J^\star(\hat{G})\right)^2 \\ =& \left(\mathbb{E}_{M \sim \hat{G}}\left[f^\infty(M) - f^\star(M)\right]\right)^2 \\ \leq& \mathbb{E}_{M \sim \hat{G}}\left(f^\infty(M) - f^\star(M)\right)^2. \end{aligned} \tag{5}$$

[1] An *estimator* is defined a function whose input is data, and a *performance function* is a tool to build an estimator.

To further understand the misspecification bias, let us assume that α_f is given by

$$\alpha_f(Q) = \operatorname*{argmin}_{f \in \mathcal{F}} \mathbb{E}_{M \sim Q} \left(f(M) - f^\star(M)\right)^2, \quad (6)$$

for any $Q \in \mathcal{P}(\mathcal{M})$ and model class $\mathcal{F}$. Then, we notice that f^∞ minimizes the mean squared error (MSE) over P, while the bias is upperbounded by MSE over $\hat{G}$, indicating that f^∞ does not necessarily minimize the upperbound of the bias. This is closely related to covariate shift (Shimodaira, 2000), and it is known that f^∞ does not minimize the upperbound (Equation (5)) when the model class is misspecified, *i.e.*, $f^\star \notin \mathcal{F}$. The bias tends to increase if $\hat{G}$ and P become largely deviated (*i.e.*, the discovered molecules are not similar to those in the sample).

3.3. Reuse-and-Finiteness Bias

The former term of Equation (4),

$$b^N_{\mathrm{PI}}(P) \coloneqq \mathbb{E}_{\hat{P} \sim P^N} \left[J_{\mathrm{PI}}(\hat{G}, \hat{f}) - J_{\mathrm{PI}}(\hat{G}, f^\infty) \right], \quad (7)$$

quantifies the bias caused by reusing $\hat{P}$ for training and testing a generator. We call it a reuse-and-finiteness bias[2].

Let us theoretically analyze this bias, assuming the sample size N is moderately large such that the asymptotic expansions are valid. Then, we have Corollary 3.2, whose proof is given in Appendix A.2.

Corollary 3.2. $b^N_{\mathrm{PI}}(P) = O(1/N)$ *holds.*

In particular, if the generator maximizes the performance metric and the predictor is unbiased, *i.e.*, $\mathbb{E}_{\hat{P} \sim P^N} \alpha_f(\hat{P}) = \alpha_f(P)$, we can prove that the bias is optimistic (Proposition 3.3). See Appendix B for its proof.

Proposition 3.3. *Assume that* $\mathbb{E}_{\hat{P} \sim P^N} \hat{f} = f^\infty$ *and* $\alpha_G(\hat{P}) = \operatorname{argmax}_{G \in \mathcal{G}} J_{\mathrm{PI}}(G, \alpha_f(\hat{P}))$ *hold. Then,* $b^N_{\mathrm{PI}}(P) \geq 0$ *holds.*

4. Bias Reduction Strategies

We have witnessed that the plug-in performance estimator is biased in two ways. In this section, we discuss how to reduce these biases to obtain reliable performance estimates.

4.1. Reducing Misspecification Bias

There are mainly three approaches to reducing the misspecification bias. The first one is to train the predictor with *covariate shift adaptation*, correcting the mismatch between training and testing distributions (Section 4.1.1). The second approach is to constrain a generator such that the molecules discovered by it become similar to those in the sample $\hat{P}$ (Section 4.1.2). These are mainly motivated by minimizing the right-hand side of Equation (5). The third one is motivated by a standard technique in contextual bandit, the *doubly-robust performance estimator* instead of the plug-in performance estimator (Section 4.1.3).

Before going into details, let us introduce the notion of importance weight, which is used extensively. For any generator G and any $Q \in \mathcal{P}(\mathcal{M})$ whose support is no smaller than that of G, let $(G/Q)(m) \coloneqq G(m)/Q(m)$ denote the importance weight between them.

4.1.1. Covariate Shift Adaptation

The misspecification bias will be reduced by minimizing its upperbound, *i.e.*, the right-hand side of Equation (5), which is MSE between f^∞ and $f^\star$ over $\hat{G}$. In contrast, f^∞ is usually defined by minimizing MSE over P, and does not necessarily minimize the upperbound. One approach suggested by Shimodaira (2000) to alleviating such a mismatch is to train the predictor as follows:

$$\min_{f \in \mathcal{F}} \mathbb{E}_{M \sim P} w(M)^\lambda (f(M) - f^\star(M))^2, \quad (8)$$

where P is the data distribution, w is a weight function approximating $\hat{G}/P$, and $\lambda \in [0, 1]$ controls the bias and variance of the estimated predictor. Noticing that the refined algorithm minimizes the upperbound if $\lambda = 1$ and $w = \hat{G}/P$, the misspecification bias will be reduced.

4.1.2. Constrain a Generator

The covariate shift adaptation does not always work. If $\hat{G}$ and P are not close enough, the effective sample size of Equation (8) decreases, leading to poor estimation of $\hat{f}$, and increasing the reuse-and-finiteness bias. This suggests that not all generators can be accurately evaluated; those largely deviated from P are difficult to be evaluated.

A straightforward idea to alleviate it is to constrain a divergence between $\hat{G}$ and P. In fact, a VAE approach naturally implements this idea by training a VAE on the data distribution, which constrains $\hat{G}$ so that it lies close to P. In contrast, an RL-based approach does not implement this idea. In the following, we will discuss how to implement this idea for the RL-based approach.

Let us assume that the policy is obtained by solving,

$$\min_{\pi \in \Pi} \ell(\pi; P). \quad (9)$$

While a natural approach is to add a divergence between the generator and the data distribution as a regularization term,

[2]The reuse-and-finiteness bias is caused by sample reuse as well as the finiteness of the sample, the latter of which is clear when the generator is independent from $\hat{P}$; the reuse-and-finiteness bias still exists in such a case if $\mathbb{E}_{\hat{P}}[\alpha_f(\hat{P})] \neq f^\infty$, namely, unless the predictor is unbiased.

it is computationally expensive, especially when the length of MDP is large.

We instead propose to regularize the policy, inspired by *behavior cloning* (Fujimoto & Gu, 2021). Behavior cloning regularizes the policy so that the policy imitates a *behavior policy* that generates the data. Assume that there exists a behavior policy π_{b} that induces the data distribution, *i.e.*, $p_H^{\pi_{\mathrm{b}}}(m) = P(m)$ for $m \in \mathcal{M}$. Note that the behavior policy may not be available in our setting. Behavior cloning employs the following regularized objective function: $\ell(\pi; G) - \frac{\nu}{H+1}\sum_{h=0}^{H}\mathbb{E}_{S_h\sim p_h^{\pi_{\mathrm{b}}}, A_h\sim\pi_{\mathrm{b}}(S_h)}\left[\log\pi(A_h \mid S_h)\right]$, where $\nu \geq 0$ is a hyperparameter controlling behavior cloning. The larger ν is, the more the learned policy resembles the behavior policy, which in turn will make p_H^{π} close to the data distribution, reducing the misspecification bias.

A technical challenge in applying behavior cloning to our setting is that π_{b} may not be available. Our observation is that while π_{b} is not available, a trajectory towards each molecule in the dataset can be often reconstructed. For example, in an MDP that constructs a molecule atom-wisely (You et al., 2018), such a trajectory is easily obtained by removing atoms one by one from the molecule; in another MDP that constructs a molecule by chemical reactions (Gottipati et al., 2020), since each molecule in the dataset should be synthesizable (molecules in the dataset do exist in reality and thus are synthesizable), such a trajectory is available for the molecules in the dataset. This observation is inspired by the expert imitation proposed by You et al. (2018), where expert actions are synthesized from molecules in the dataset.

Letting $\pi_{\mathrm{b}}^{-1}(m) = (s_0, a_0, s_1, a_1, \ldots, s_H = m)$ be a (potentially random) function to reconstruct a trajectory from a molecule, we propose to use a regularizer defined by $\Psi(\pi; P) = \frac{1}{H+1}\sum_{h=0}^{H}\mathbb{E}_{M\sim P}\mathbb{E}_{S_0,A_0,\ldots,S_H\sim\pi_{\mathrm{b}}^{-1}(M)}\left[\log\pi(A_h \mid S_h)\right]$, which leads to the following optimization problem:

$$\min_{\pi\in\Pi}\ell(\pi; P) - \nu\Psi(\pi; P). \tag{10}$$

Although this regularization is not sufficient to constrain the divergence between $p_H^{\hat{\pi}}$ and P (which has been discussed in the literature of imitation learning), we consider the idea of behavior cloning is a simple yet effective heuristic to reduce the misspecification bias, which will be investigated in the experiment.

4.1.3. Doubly-Robust Performance Estimator

The third approach to reducing the misspecification bias is a *doubly-robust performance estimator*, which has been applied in contextual bandit (Dudík et al., 2014) and offline reinforcement learning (Tang et al., 2020) as an alternative to the plug-in performance estimator. As we will see later, the performance can also be estimated via importance sampling. The doubly-robust performance estimator combines these two estimators so as to inherit their benefits.

Importance-Sampling Performance Estimator. Given the following change-of-measure,

$$J^\star(G) = \mathbb{E}_{M\sim G}f^\star(M) = \mathbb{E}_{M\sim P}(G/P)(M)f^\star(M),$$

we obtain the following *importance-sampling performance function* by substituting an importance weight model w for the true importance weight,

$$J_{\mathrm{IS}}(w, P) \coloneqq \mathbb{E}_{M\sim P}w(M)f^\star(M).$$

This coincides with $J^\star(G)$ if $w = G/P$. We obtain an importance-sampling performance estimator by substituting any weight estimator for w and $\hat{P}$ for P.

Doubly-Robust Performance Estimator. The *doubly-robust performance function* is defined as follows:

$$\begin{aligned}&J_{\mathrm{DR}}(G, w, f, P)\\ \coloneqq&\mathbb{E}_{M\sim P}\left[w(M)(f^\star(M) - f(M))\right] + \mathbb{E}_{M\sim G}f(M).\end{aligned} \tag{11}$$

It combines the plug-in and importance-sampling performance functions in that $J_{\mathrm{DR}}(G, 0, f, P) = J_{\mathrm{PI}}(G, f)$ and $J_{\mathrm{DR}}(G, w, 0, P) = J_{\mathrm{IS}}(G, w, P)$ hold.

This is called doubly-robust because its misspecification bias is expressed as,

$$\begin{aligned}&J_{\mathrm{DR}}(\hat{G}, w^\infty, f^\infty, P) - J^\star(\hat{G})\\ =&\mathbb{E}_{M\sim P}(w^\infty(M) - (\hat{G}/P)(M))(f^\star(M) - f^\infty(M)),\end{aligned}$$

where let w^∞ be the weight model trained on P. The above expression suggests that the bias disappears if either the predictor *or* the weight model is correctly specified.

Discussion. Notice that the misspecification biases of J_{PI} and J_{IS} are given as follows:

$$\mathbb{E}_{M\sim P}\left[(\hat{G}/P)(M)(f^\infty(M) - f^\star(M))\right],$$
$$\mathbb{E}_{M\sim P}\left[(w^\infty(M) - (\hat{G}/P)(M))f^\star(M)\right].$$

We can deduce that for $M \sim P$ (i) if $|f^\star(M) - f^\infty(M)| \ll |f^\star(M)|$ holds, the misspecification bias of J_{DR} will be smaller than that of J_{IS}, and (ii) if $|(\hat{G}/P)(M) - w^\infty(M)| \ll |(\hat{G}/P)(M)|$ holds, the misspecification bias of J_{DR} will be smaller than that of J_{PI}. Therefore, if we can learn both of the predictor and the weight model well, the doubly-robust performance estimator is preferred to the other estimators. Otherwise, the doubly-robust performance estimator can be worse than the others.

4.1.4. Summary

We have introduced three approaches to reducing misspecification bias. One can find the best combination of them depending on the closeness between $\hat{G}$ and P. If they are close enough that it is easy to estimate the importance weight between them, the first and third approaches will work, while the second approach may not be necessary. Otherwise, since the importance weight cannot be estimated well, only the second approach will work.

4.2. Reducing Reuse-and-Finiteness Bias

We investigate how to reduce the reuse-and-finiteness bias. Our approach is to estimate the bias and substract it from the estimator. Such a bias reduction has been extensively discussed in the literature of information criteria (Konishi & Kitagawa, 2007), which aim to estimate the test performance of a predictor by correcting the bias of its training performance. There are mainly two approaches: train-test split method and bootstrap method. In the following, we focus on the naive plug-in performance estimator for simplicity. However, similar results can be derived for the refined estimators presented in the previous sections.

4.2.1. Bias Estimation by Train-test Split

The first approach estimates the bias via train-test split of the sample. The sample $\mathcal{D}$ is randomly split into $\mathcal{D}_{\mathrm{train}}$ and $\mathcal{D}_{\mathrm{test}}$ such that $\mathcal{D}_{\mathrm{train}} \cap \mathcal{D}_{\mathrm{test}} = \emptyset$ and $\mathcal{D}_{\mathrm{train}} \cup \mathcal{D}_{\mathrm{test}} = \mathcal{D}$. Let $\hat{G}_{\mathrm{train}}$ and $\hat{f}_{\mathrm{train}}$ be the generator and predictor trained using $\mathcal{D}_{\mathrm{train}}$, and $\hat{f}_{\mathrm{test}}$ be the predictor trained using $\mathcal{D}_{\mathrm{test}}$. Then, the reuse-and-finiteness bias is estimated by

$$b^N_{\mathrm{split}}(\hat{P}) = \mathbb{E}[J_{\mathrm{PI}}(\hat{G}_{\mathrm{train}}, \hat{f}_{\mathrm{train}}) - J_{\mathrm{PI}}(\hat{G}_{\mathrm{train}}, \hat{f}_{\mathrm{test}})], \tag{12}$$

where the expectation is taken over random train-test split.

While this estimator seems to be reasonable, it is not recommended due to the bias of the bias estimator. As demonstrated in Corollary 4.1, the train-test split estimator has $O(1/N)$ bias, the same order as the bias $b^N(P)$ itself, and we cannot distinguish between the bias and the bias of the bias. Such a bias is due to the non-linearlity of $J_{\mathrm{PI}}(\hat{G}_{\mathrm{train}}, \hat{f}_{\mathrm{test}})$ with respect to $\hat{P}_{\mathrm{test}}$, the distribution used for testing. See Appendix A.3 for its proof and Appendix C for in-depth discussions.

Corollary 4.1. *Suppose we randomly divide the sample such that* $|\mathcal{D}_{\mathrm{train}}| : |\mathcal{D}_{\mathrm{test}}| = \lambda : (1-\lambda)$ *for some* $\lambda \in (0, 1)$. *Then,* $\mathbb{E}_{\hat{P}\sim P^N}[b^N_{\mathrm{split}}(\hat{P})] = b^N(P) + O(1/N)$ *holds.*

Note that direct estimation of test performance by $J_{\mathrm{PI}}(\hat{G}_{\mathrm{train}}, \hat{f}_{\mathrm{test}})$ is not recommended similarly, unless the size of the test sample is sufficiently large. See Appendix C for detailed discussion.

4.2.2. Bootstrap Bias Estimation

An alternative approach is bootstrap (Efron & Tibshirani, 1994). A bootstrap estimator of the reuse-and-finiteness bias $b^N(P)$ is obtained by plugging $\hat{P}$ into P. Letting $\hat{P}^\star$ be a resampled sample from $\hat{P}$, and letting $\hat{G}^\star$ and $\hat{f}^\star$ be the generator and predictor trained using $\hat{P}^\star$, the boostrap estimator is defined as,

$$b^N(\hat{P}) = \mathbb{E}_{\hat{P}^\star\sim\hat{P}^N}[J_{\mathrm{PI}}(\hat{G}^\star, \hat{f}^\star) - J_{\mathrm{PI}}(\hat{G}^\star, \hat{f})], \tag{13}$$

which can be computed by Monte-Carlo approximation.

In contrast to the train-test split method, the bootstrap bias estimation suffers less from the bias of the estimator, as stated in Corollary 4.2. See Appendix A.4 for its proof.

Corollary 4.2. $\mathbb{E}_{\hat{P}\sim P^N}[b^N(\hat{P})] = b^N(P) + O(1/N^2)$.

4.2.3. Summary

We have introduced two reusing-bias estimators, referring to the literature of information criteria. We have found that the train-test split estimator, one of the most popular estimators, cannot reliably estimate the bias in our problem setting, although it works in supervised learning. In contrast, the bootstrap bias estimator is shown to be less biased than the train-test split estimator and can estimate the reuse-and-finiteness bias more reliably. Therefore, we conclude that the bootstrap bias estimator is preferable to the train-test split estimator.

From computational point of view, the bootstrap bias estimator requires us to train M generators and $M + 1$ predictors. We set $M = 20$ in the experiments given the result of a preliminary experiment. Since the bootstrap procedure can be easily parallelized with low overhead, its wall-clock time can be reduced in proportion to the computational resource.

5. Empirical Studies

Let us empirically quantify the two biases as well as the effectiveness of the bias reduction methods. We employ a reinforcement learning setting as a case study. The code used in our empirical studies will be available in `https://github.com/kanojikajino/biases-in-mol-opt`.

5.1. Setup

Let us explain our experimental setup. See Appendix D for full details to ensure reproduciability.

Molecular Representation. Unless otherwise indicated, all of the functions defined over molecules use the 1024-bit Morgan fingerprint (Morgan, 1965; Rogers & Hahn, 2010) with radius 2 as a feature extractor.

Environment and Agent. We employ the environment and

the agent by Gottipati et al. (2020) with minor modifications. The agent receives a molecule as the current state, and outputs an action consisting of a reaction template and a reactant. The environment, receiving the action, applies the chemical reaction defined by the action to the current molecule to generate a product, which is then set as the next state. This procedure is repeated for H times, and lastly the agent takes action $a_\perp$ to be rewarded by the property of the final product. We set $H = 1$ to reduce the variance in the estimated performance and better highlight the biases and their reduction. The agent is implemented by actor-critic using fully-connected neural networks.

We use the reaction templates curated by Button et al. (2019) and prepare the reactants from the set of commercially available substances in the same way as the original environment. The number of reaction templates is 64, 15 of which require one reactant, and 49 of which require two reactants. The number of reactants is 150,560.

Predictor and Importance Weight Model. As a predictor, we use a fully-connected neural network with one hidden layer of 96 units with softplus activations except for the last layer. It is trained by minimizing MSE defined over $\hat{P}$.

The importance weight model is learned by the kernel unconstrained least-squares importance fitting (KuLSIF) (Kanamori et al., 2012). In particular, the last layer of the trained predictor is used as a feature extractor, and we compute the linear kernel using it.

Evaluation framework. To evaluate the biases, we need $f^\star$, which however is not available in general. We thus design a semi-synthetic experiment using a real-world dataset $\mathcal{D}_0 = \{(m_n, f^\star(m_n)) \in \mathcal{M} \times \mathbb{R}\}_{n=1}^{N_0}$. We regard a predictor trained on $\mathcal{D}_0$ as the true property function $f^\star$. In specific, we used the predictor provided by Gottipati et al. (2020), which was trained on the ChEMBL database (Gaulton et al., 2017) to predict pIC_{50} value associated with C-C chemokine receptor type 5 (CCR5). With this property function, we have full access to the environment, and we can construct an offline dataset $\mathcal{D}$ of an arbitrary sample size by running a random policy.

To decompose the bias into the misspecification bias and the reuse-and-finiteness bias, we also need f^∞, the predictor obtained with full access to the data-generating distribution P. We approximate it by $\hat{f}_{\mathrm{test}}$, which is a predictor trained by a large sample $\mathcal{D}_{\mathrm{test}}$ of size 10^5, constructed independently of $\mathcal{D}$. This approximation is valid if $|\mathcal{D}_{\mathrm{test}}|$ is sufficiently large (see Proposition C.2). Then, the misspecification bias can be estimated by $J_{\mathrm{PI}}(\hat{G}, \hat{f}_{\mathrm{test}}) - J^\star(\hat{G})$ and the reuse-and-finiteness bias by $J_{\mathrm{PI}}(\hat{G}, \hat{f}) - J_{\mathrm{PI}}(\hat{G}, \hat{f}_{\mathrm{test}})$. The performance estimators are defined by the expectation with respect to a trajectory of a policy, and we estimate them by Monte-Carlo approximation with 1,000 trajectories.

5.2. Quantifying the Two Biases

First, we aim to quantify the misspecification and reuse-and-finiteness biases. We design an experiment to investigate the relationship between these biases and the sample size.

We vary the training sample size N in $\{2^6, 2^7, \ldots, 2^{13}\}$. For each N, we repeatedly generate a pair of train and test sets for five times, and evaluate the biases as indicated above. We report the means and standard deviations.

Figure 1 (left) illustrates the result. We have three observations. First, when $N = 2^7$, the misspecification bias, $J_{\mathrm{PI}}(\hat{G}, f^\infty) - J^\star(\hat{G})$, was roughly twice as large as the reuse-and-finiteness bias, $J_{\mathrm{PI}}(\hat{G}, \hat{f}) - J_{\mathrm{PI}}(\hat{G}, f^\infty)$, demonstrating that both are non-negligible. Second, for $N \geq 2^7$, the reuse-and-finiteness bias increased as the size of the training sample decreased, which coincides with Corollary 3.2. The results for $N < 2^7$ did not coincide with our theoretical result because the sample size is not large enough for asymptotic expansion to be justified. Third, the ground-truth performance of the policies were rather stable across different training sample sizes. We found that the policies were similar to each other, suggesting that this environment has a local optimum with a reasonably good performance[3]. This also suggests that the policy learner was insensitive to the particular sample, and the reuse-and-finiteness bias in this case is mainly caused by the finiteness of the sample to train the predictor, not by reusing the same sample.

5.3. Quantifying Bias Reduction Methods

We then study the effectiveness of the bias reduction methods presented in Section 4. Since the behavior cloning coefficient ν will control the trade-off between the misspecification bias and the performance of the learned policy, it should be determined according to the user's requirement, *i.e.*, whether the accuracy of performance estimation or the actual performance is prioritized. Therefore, we design an experiment to evaluate the effectiveness of the bias reduction methods, varying ν in the range of $\{2^{-4}, \ldots, 2^4\}$.

To investigate the effectiveness of the bias reduction methods, we estimate the performance by i) the vanilla plug-in performance estimator (Section 2.3), ii) that enhanced by covariate shift adaptation (Section 4.1.1), iii) that enhanced by bootstrap bias estiomation (Section 4.2.2), and iv) the doubly-robust performance estimator (Section 4.1.3).

Figure 1 (middle) illustrates the performance estimates for $N = 10^3$. Since the doubly-robust performance estimator J_{DR} performs worse than the vanilla plug-in performance estimator J_{PI}, we omit it from the figure. See Ap-

[3]The score 6.8 is better than the score of a random policy, 5.8, and thus, we cocnclude the performance is *reasonably good*.

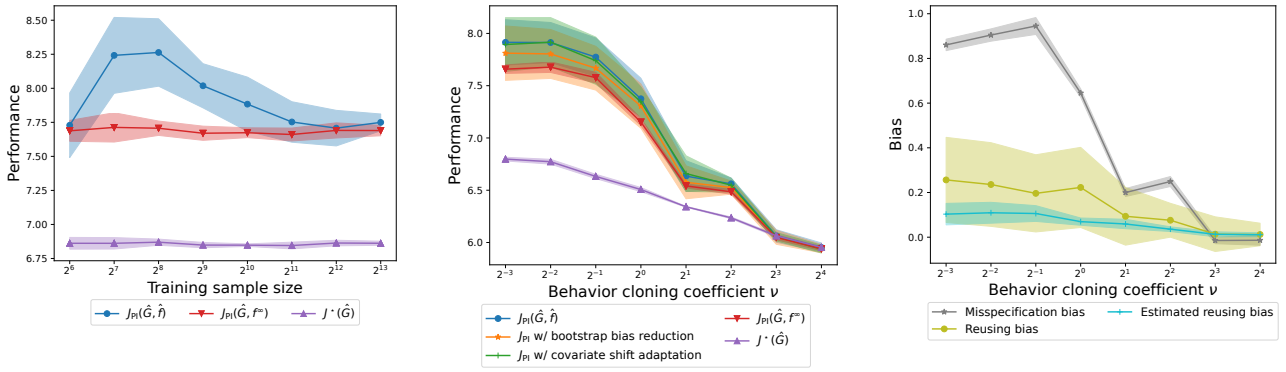

Figure 1. Lines show means and shaded areas show standard deviations. **(Left)** Relationship between the biases and the sample size. $J_{\mathrm{PI}}(\hat{G}, \hat{f}) - J_{\mathrm{PI}}(\hat{G}, f^{\infty})$ corresponds to the reuse-and-finiteness bias and $J_{\mathrm{PI}}(\hat{G}, f^{\infty}) - J^{\star}(\hat{G})$ to the misspecification bias. **(Middle)** Comparison between bias reduction methods. **(Right)** Comparison between the misspecification bias, reuse-and-finiteness bias, and the estimated reuse-and-finiteness bias.

pendix E for the full result. We observe that the bootstrap bias reduction worked well, while the benefit of the covariate shift adaptation is marginal. This indicates that the importance weight estimation did not work well in this setting.

Figure 1 (right) illustrates the biases in the vanilla plug-in performance estimate and the reuse-and-finiteness bias estimated by bootstrap. As we expected, the misspecification bias tends to decrease as we increase ν. The reuse-and-finiteness bias is under-estimated, but the bias estimation contributes to bias correction.

In summary, we confirm that (i) behavior cloning can reduce the misspecification bias at the expense of performance degradation, (ii) the reuse-and-finiteness bias can be estimated and corrected by bootstrap, and (iii) the methods using weight models did not perform well in our setting.

6. Related Work

Our primary contribution is the comprehensive study of theoretically-sound evaluation methodology for molecular optimization algorithms. Since the pioneering work by Gómez-Bombarelli et al. (2018), a number of studies on this topic have been published in the communities of machine learning and cheminformatics to advance the state-of-the-art. While some of them (Takeda et al., 2020; Das et al., 2021) have been validated on physical experiments, many others have been evaluated in computer simulation.

Early studies (Kusner et al., 2017) adopted the octanol-water partition coefficient, $\log P$, penalized by the synthetic accessibility score (Ertl & Schuffenhauer, 2009) and the number of long rings as the target property to be maximized. The score can be easily computed by RDKit, and is often implicitly regarded as a reliable score computed by an accurate simulator. Some recently consider that the $\log P$ optimization is not appropriate as a benchmark task because it is easy to optimize (Brown et al., 2019) or its prediction can be inaccurate (Yang et al., 2021), and alternative benchmark tasks have been investigated; some of them propose a suite of benchmark tasks (Brown et al., 2019; Polykovskiy et al., 2020) and the others use other property functions trained by real-world data (Olivecrona et al., 2017; Li et al., 2018; Jin et al., 2020; Gottipati et al., 2020; Xie et al., 2021).

As far as we are aware of, there are at least two empirical studies concerning about potential biases in the plug-in performance estimator. Renz et al. (2019) pointed out that the plug-in performance estimator is biased due to the divergence between the predictor and the true objective function and overfitting of the predictor, while a follow-up study by Langevin et al. (2022) attributed the bias to the train-test split used by Renz et al. (2019); the train and test sets were far from being identically distributed. While these two pioneering studies shed light on the potential flaw in the plug-in performance estimator, they do not offer full explanations due to being solely empirical.

Our contribution to this line of studies is that we empirically and theoretically demonstrate potential biases in the current evaluation methodology using real-world data and present bias reduction methods. In specific, we theoretically analyze the bias pointed out by Renz et al. (2019) and Langevin et al. (2022) to identify two sources of the bias (Theorem 3.1), which allows us not only to understand the bias clearly but also to devise bias reduction methods.

Our analysis also unveils why the $\log P$ optimization task has been hacked, and warns that the alternative benchmark tasks will be hacked as long as no bias reduction method is applied. The $\log P$ function implemented in RDKit (Wildman & Crippen, 1999) is obtained by fitting a linear model to a dataset of experimental $\log P$ values, and is in fact a predictor. Our theory suggests that unless the bias reduc-

tion methods are applied, the learned generator can generate unrealistic molecules that are far from those in the dataset (which has been often reported in $\log P$ optimization), and the resultant performance estimates are biased. It also suggests that by incorporating bias reduction methods, we can reliably estimate the performance and therefore can safely compare different methods even when using the $\log P$ optimization task.

A related but different literature is the applicability domain (AD) of a quantitative structure–activity relationship (QSAR) model (Gadaleta et al., 2016). In short, AD is a subspace of $\mathcal{M}$ where every prediction is reliable. In other words, it aims to ensure the reliability of prediction for a *single* molecule. In contrast, we aim to ensure the reliability of the performance of a generator, defined by predictions for *multiple* molecules. This indicates that our problem is easier than setting AD; a method to define AD can solve our problem, but a solution to our problem cannot define AD.

7. Conclusion and Future Work

We have discussed that the plug-in performance estimator is biased in two ways; one is due to model misspecification and the other is due to reusing the same finite sample for training and testing. In order to reduce these biases to obtain more accurate estimates, we recommend to (i) add a constraint to the generator so that it stays close to the data distribution and (ii) correct the bias by bootstrapping if it is non-negligible and we can afford to do it.

Since the present paper focuses on analysing the biases and puts less focus on bias reduction methods, an interesting research direction is to develop more sophisticated bias reduction methods. One important step is to improve the importance weight estimation so that the bias reduction methods using importance weights work. Another direction is to constrain a generator with less performance degradation. Adversarial training as done by You et al. (2018) may be helpful to better constrain the generator.

Another interesting direction is to study the exploration-exploitation dilemma when we are granted a limited access to the true property function $f^\star$. The present paper focused on the situation where no data acquisition is allowed, and therefore, it is recommended to learn a conservative generator. However, if a limited number of data can be acquired, we may first learn a generator without any constraint for exploration, and gradually turn to more conservative generators so that we can accurately estimate the performance. It is valuable to study how to balance exploration and exploitation.

References

Brown, N., Fiscato, M., Segler, M. H. S., and Vaucher, A. C. Guacamol: Benchmarking models for de novo molecular design. *Journal of Chemical Information and Modeling*, 59:1096–1108, 2019.

Button, A., Merk, D., Hiss, J. A., and Schneider, G. Automated de novo molecular design by hybrid machine intelligence and rule-driven chemical synthesis. *Nature Machine Intelligence*, 1:307–315, 2019. ISSN 2522-5839. doi: 10.1038/s42256-019-0067-7. URL https://doi.org/10.1038/s42256-019-0067-7.

Das, P., Sercu, T., Wadhawan, K., Padhi, I., Gehrmann, S., Cipcigan, F., Chenthamarakshan, V., Strobelt, H., dos Santos, C., Chen, P.-Y., Yang, Y. Y., Tan, J. P. K., Hedrick, J., Crain, J., and Mojsilovic, A. Accelerated antimicrobial discovery via deep generative models and molecular dynamics simulations. *Nature Biomedical Engineering*, 5:613–623, 2021. ISSN 2157-846X. doi: 10.1038/s41551-021-00689-x. URL https://doi.org/10.1038/s41551-021-00689-x.

Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011. ISSN 15324435. URL http://jmlr.org/papers/v12/duchi11a.html.

Dudík, M., Erhan, D., Langford, J., and Li, L. Doubly robust policy evaluation and optimization. *Statistical Science*, 29:485–511, 12 2014. ISSN 08834237, 21688745. URL http://www.jstor.org/stable/43288496.

Efron, B. and Tibshirani, R. J. *An introduction to the bootstrap*. CRC press, 1994.

Ertl, P. and Schuffenhauer, A. Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. *Journal of Cheminformatics*, 1:8, 2009. ISSN 1758-2946. doi: 10.1186/1758-2946-1-8. URL https://doi.org/10.1186/1758-2946-1-8.

Fujimoto, S. and Gu, S. A minimalist approach to offline reinforcement learning. 2021. URL https://openreview.net/forum?id=Q32U7dzWXpc.

Gadaleta, D., Mangiatordi, G. F., Catto, M., Carotti, A., and Nicolotti, O. Applicability domain for qsar models: where theory meets reality. *International Journal of Quantitative Structure-Property Relationships (IJQSPR)*, 1:45–63, 2016.

Gaulton, A., Hersey, A., Nowotka, M., Bento, A. P., Chambers, J., Mendez, D., Mutowo, P., Atkinson, F., Bellis, L. J., Cibrián-Uhalte, E., Davies, M., Dedman, N., Karlsson, A., Magariños, M. P., Overington, J. P., Papadatos, G., Smit, I., and Leach, A. R. The ChEMBL database in 2017. *Nucleic acids research*, 45:D945–D954, 1 2017. ISSN 1362-4962 (Electronic). doi: 10.1093/nar/gkw1074.

Gottipati, S. K., Sattarov, B., Niu, S., Pathak, Y., Wei, H., Liu, S., Liu, S., Blackburn, S., Thomas, K., Coley, C., Tang, J., Chandar, S., and Bengio, Y. Learning to navigate the synthetically accessible chemical space using reinforcement learning. volume 119, pp. 3668–3679. PMLR, 2020. URL http://proceedings.mlr.press/v119/gottipati20a.html.

Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., and Aspuru-Guzik, A. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 2018. doi: 10.1021/acscentsci.7b00572.

Jin, W., Barzilay, D., and Jaakkola, T. Multi-objective molecule generation using interpretable substructures. volume 119, pp. 4849–4859. PMLR, 2020. URL https://proceedings.mlr.press/v119/jin20b.html.

Kanamori, T., Suzuki, T., and Sugiyama, M. Statistical analysis of kernel-based least-squares density-ratio estimation. *Machine Learning*, 86:335–367, 2012. ISSN 1573-0565. doi: 10.1007/s10994-011-5266-3. URL https://doi.org/10.1007/s10994-011-5266-3.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. 2014.

Konishi, S. and Kitagawa, G. *Information Criteria and Statistical Modeling*. Springer Publishing Company, Incorporated, 1st edition, 2007. ISBN 0387718869.

Kusner, M. J., Paige, B., and Hernández-Lobato, J. M. Grammar variational autoencoder. volume 70, pp. 1945–1954. PMLR, 2017. URL https://proceedings.mlr.press/v70/kusner17a.html.

Langevin, M., Vuilleumier, R., and Bianciotto, M. Explaining and avoiding failure modes in goal-directed generation of small molecules. *Journal of Cheminformatics*, 14:20, 2022. ISSN 1758-2946. doi: 10.1186/s13321-022-00601-y. URL https://doi.org/10.1186/s13321-022-00601-y.

Li, Y., Zhang, L., and Liu, Z. Multi-objective de novo drug design with conditional graph generative model. *Journal of Cheminformatics*, 10:33, 2018. ISSN 1758-2946. doi: 10.1186/s13321-018-0287-6. URL https://doi.org/10.1186/s13321-018-0287-6.

Morgan, H. L. The generation of a unique machine description for chemical structures-a technique developed at chemical abstracts service. *Journal of Chemical Documentation*, 5:107–113, 5 1965. ISSN 0021-9576. doi: 10.1021/c160017a018. URL https://doi.org/10.1021/c160017a018. doi: 10.1021/c160017a018.

Olivecrona, M., Blaschke, T., Engkvist, O., and Chen, H. Molecular de-novo design through deep reinforcement learning. *Journal of Cheminformatics*, 9:48, 2017. ISSN 1758-2946. doi: 10.1186/s13321-017-0235-x. URL https://doi.org/10.1186/s13321-017-0235-x.

Polykovskiy, D., Zhebrak, A., Sanchez-Lengeling, B., Golovanov, S., Tatanov, O., Belyaev, S., Kurbanov, R., Artamonov, A., Aladinskiy, V., Veselov, M., Kadurin, A., Johansson, S., Chen, H., Nikolenko, S., Aspuru-Guzik, A., and Zhavoronkov, A. Molecular sets (moses): A benchmarking platform for molecular generation models. *Frontiers in Pharmacology*, 11:1931, 2020. ISSN 1663-9812. doi: 10.3389/fphar.2020.565644. URL https://www.frontiersin.org/article/10.3389/fphar.2020.565644.

Renz, P., Rompaey, D. V., Wegner, J. K., Hochreiter, S., and Klambauer, G. On failure modes in molecule generation and optimization. *Drug Discovery Today: Technologies*, 32-33:55–63, 2019. ISSN 1740-6749. doi: https://doi.org/10.1016/j.ddtec.2020.09.003. URL https://www.sciencedirect.com/science/article/pii/S1740674920300159. Artificial Intelligence.

Rogers, D. and Hahn, M. Extended-connectivity fingerprints. *Journal of Chemical Information and Modeling*, 50:742–754, 5 2010. ISSN 1549-9596. doi: 10.1021/ci100050t. URL https://doi.org/10.1021/ci100050t. doi: 10.1021/ci100050t.

Shimodaira, H. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90:227–244, 2000. ISSN 0378-3758. doi: https://doi.org/10.1016/S0378-3758(00)00115-4. URL https://www.sciencedirect.com/science/article/pii/S0378375800001154.

Takeda, S., Hama, T., Hsu, H.-H., Piunova, V. A., Zubarev, D., Sanders, D. P., Pitera, J. W., Kogoh, M., Hongo, T., Cheng, Y., Bocanett, W., Nakashika, H., Fujita, A., Tsuchiya, Y., Hino, K., Yano, K., Hirose, S., Toda, H., Orii, Y., and Nakano, D. Molecular inverse-design platform for material industries, 2020. URL https://doi.org/10.1145/3394486.3403346.

Tang, Z., Feng, Y., Li, L., Zhou, D., and Liu, Q. Doubly robust bias reduction in infinite horizon off-policy estimation. 2020. URL https://openreview.net/forum?id=S1glGANtDr.

Wildman, S. A. and Crippen, G. M. Prediction of physicochemical parameters by atomic contributions. *Journal of Chemical Information and Computer Sciences*, 39:868–873, 9 1999. ISSN 0095-2338. doi: 10.1021/ci990307l. URL https://doi.org/10.1021/ci990307l. doi: 10.1021/ci990307l.

Xie, Y., Shi, C., Zhou, H., Yang, Y., Zhang, W., Yu, Y., and Li, L. MARS: Markov molecular sampling for multi-objective drug discovery. 2021. URL https://openreview.net/forum?id=kHSu4ebxFXY.

Yang, X., Aasawat, T., and Yoshizoe, K. Practical massively parallel Monte-Carlo tree search applied to molecular design. 2021. URL https://openreview.net/forum?id=6k7VdojAIK.

You, J., Liu, B., Ying, Z., Pande, V., and Leskovec, J. Graph convolutional policy network for goal-directed molecular graph generation. pp. 6412–6422, 2018.

A. Asymptotic Analysis on the Reuse-and-Finiteness Bias

In this section, we analyze the reuse-and-finiteness bias by stochastic expansion to prove Corollary 3.2. We review the technical background of Taylor expansion using Fréchet derivative (Appendix A.1.1) and stochastic expansion of an estimator (Appendix A.1.2).

A.1. Technical Background

For completeness, let us first define the Fréchet derivative and Taylor expansion using it in Appendix A.1.1.

A.1.1. Taylor Expansion using Fréchet Derivative

In this paper, we mainly analyze a function between Banach spaces by its Taylor expansion. To do so, it is necessary to introduce the Fréchet derivative, which is a generalization of the total derivative on the space of real numbers to that on Banach spaces. In this section, we provide a brief introduction to the Fréchet derivative and Taylor expansion.

Let V and W be Banach spaces, $U \subset V$, and $f\colon U \to W$ be a function. If a bounded linear mapping $A\colon V \to W$ such that

$$\lim_{\|h\|\to 0} \frac{\|f(x+h) - f(x) - A_x(h)\|}{\|h\|} = 0 \tag{14}$$

exists, $f_x^{(1)} \coloneqq A_x$ is called the Fréchet derivative of f at $x \in U$. Let D be the Fréchet differential operator and we express $Df_x \coloneqq f_x^{(1)}$ when emphasizing the operator. Equation (14) implies that,

$$f(x+h) = f(x) + f_x^{(1)}(h) + o(\|h\|), \tag{15}$$

holds. Similarly, we can define a higher-order Fréchet derivative $f_x^{(k)}$ for $k \geq 0$, and it is a symmetric multilinear map from V^k to W when fixing x. The Taylor expansion of f is obtained as,

$$f(x+h) = \sum_{k=0}^{\infty} \frac{1}{k!} f_x^{(k)}(h^{\otimes k}), \tag{16}$$

where $h^{\otimes k}$ represents k repetitions of h.

For a bivariate function $f(x,y)\colon U^2 \to W$, let us introduce partial Fréchet derivatives. If a bounded linear mapping $A\colon V \to W$ such that,

$$\lim_{\|h_x\|\to 0} \frac{\|f(x+h_x, y) - f(x,y) - A_{x,y}(h_x)\|}{\|h_x\|} = 0 \tag{17}$$

exists, $f_{x,y}^{(1,0)} \coloneqq A_{x,y}$ is called the $(1,0)$-th Fréchet derivative of f at $(x,y) \in U$ with respect to x. Similarly, we can define the (k,l)-th Fréchet derivative $f_{x,y}^{(k,l)}$ as a multilinear map from $V^k \times V^l$ to W, when fixing $(x,y) \in U^2$. Let $D^{(k,l)}$ be the Fréchet differential operator and we express $D^{(k,l)} f_{x,y} \coloneqq f_{x,y}^{(k,l)}$ when putting emphasis on the operator. Then, the Taylor expansion of f is obtained as,

$$f(x+h_x, y+h_y) = \sum_{k=0}^{\infty} \frac{1}{k!} \sum_{l=0}^{k} \binom{k}{l} f_{x,y}^{(l,k-l)}(h_x^{\otimes l}, h_y^{\otimes (l-k)}). \tag{18}$$

A.1.2. Stochastic Expansion

Stochastic expansion is a mathematical tool to expand an estimator $\theta(P)$ with respect to its input distribution P. We are often interested in the estimator averaged over the possible sample space, $\mathbb{E}_{\hat{P}\sim P^N}\theta(\hat{P})$, and we often expand $\theta(\hat{P})$ around P to understand the averaged estimator. This section provides a useful formula to compute it.

Lemma A.1 (Stochastic expansion formula)**.** *Let $\mathcal{X}$ be a set and let $\mathcal{P}(\mathcal{X})$ be the set of probability measures on $\mathcal{X}$. Let V be a Banach space, and let $f\colon \mathcal{P}(\mathcal{X}) \to V$ be a function. Let $\mathcal{D} = \{X_n\}_{n=1}^N$ be an i.i.d. sample from $P \in \mathcal{P}(\mathcal{X})$.*

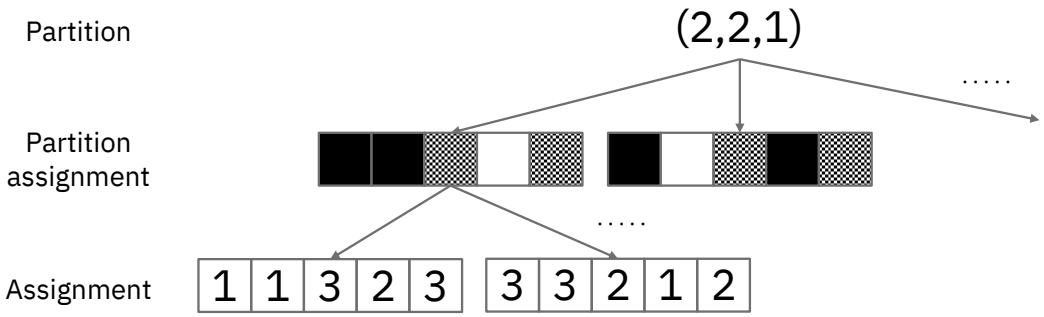

Figure 2. Assignment in $N = 3$, $k = 5$ can be abstracted into a *partition assignment*, where blocks with the same pattern fill will have the same index. A partition assignment is further abstracted into a *partition*, a sequence of non-increasing integers, each of which indicates the size of each pattern.

Let $n_1, \dots, n_k \in [N]$. If the k-th Fréchet derivative of f exists at P and there exists $i \in [k]$ such that $n_i \neq n_j$ for all $j \in [k] \backslash \{i\}$ (such an index i is called singular*), then*

$$\mathbb{E}_{\mathcal{D}}[f_P^{(k)}(\delta_{X_{n_1}} - P, \dots, \delta_{X_{n_k}} - P)] = 0, \tag{19}$$

holds. Moreover, the number of assignments $(n_1, \dots, n_k) \in [N]^k$ such that there does not *exist singular indices is $O(N^{\lfloor k/2 \rfloor})$ regarding k as a constant.*

Proof. The expectation can be calculated as,

$$\begin{aligned}
&\mathbb{E}_{\mathcal{D}}[f_P^{(k)}(\delta_{X_{n_1}} - P, \dots, \delta_{X_{n_k}} - P)] \\
=&\mathbb{E}_{\mathcal{D}\backslash X_i}[\mathbb{E}_{X_i}[f_P^{(k)}(\delta_{X_{n_1}} - P, \dots, \delta_{X_{n_i}} - P, \dots, \delta_{X_{n_k}} - P)]] && (\because \{X_n\}_{n=1}^N \text{are independent}) \\
=&\mathbb{E}_{\mathcal{D}\backslash X_i}[f_P^{(k)}(\delta_{X_{n_1}} - P, \dots, \mathbb{E}_{X_i}[\delta_{X_{n_i}} - P], \dots, \delta_{X_{n_k}} - P)] && (\because f_P^{(k)} \text{is multilinear}) \\
=&\mathbb{E}_{\mathcal{D}\backslash X_i}[f_P^{(k)}(\delta_{X_{n_1}} - P, \dots, 0, \dots, \delta_{X_{n_k}} - P)] \\
=&0. && (\because f_P^{(k)} \text{is multilinear})
\end{aligned}$$

Then, let us count the number of non-singular assignments $(n_1, \dots, n_k) \in [N]^k$ by using the abstraction illustrated in Figure 2. Assignments can be abstracted into *partition assignments*, and they are further abstracted into *partitions*, as explained in the caption of Figure 2.

A sequence of integers $p = (p_1, \dots, p_L)$ is a partition if and only if $p_1 \geq \cdots \geq p_L \geq 1$ and $\sum_{l=1}^L p_l = k$. The number of partitions depends only on k, not on N. For each partition, the number of associated partition assignments depends only on k, not on N. For each partition assignment with partition $p = (p_1, \dots, p_L)$, the number of associated assignments is at most N^L. Therefore, the number of assignments associated with partition $p = (p_1, \dots, p_L)$ is at most $C(k)N^L$. Since for a partition to be non-singular, it must not contain 1, *i.e.*, $p_L \geq 2$, $L \leq \lfloor \frac{k}{2} \rfloor$ holds. Therefore, the number of non-singular assignments is at most $C(k)N^{\lfloor \frac{k}{2} \rfloor}$. □

A.2. Result 1: Asymptotic Analysis of the Reuse-and-Finiteness Bias

This section analyze the reuse-and-finiteness bias in the asymptotic case (Corollary 3.2). For notational simplicity, we provide a more general statement in Proposition A.2, and we position Corollary 3.2 as a corollary of Proposition A.2; we can prove Corollary 3.2 by setting $\tau(P_1, P_2) = J_{\mathrm{PI}}(\alpha_G(P_1, \alpha_f(P_1)), \alpha_f(P_2))$. In the following, we often use τ instead of the performance estimator of our interest for notational simplicity.

Proposition A.2. *Let $\mathcal{X}$ be a set and let $\tau \colon \mathcal{P}(\mathcal{X})^2 \to \mathbb{R}$ be a bivariate function. Let us define a bias of τ by,*

$$b^N(P; \tau) \coloneqq \mathbb{E}_{\hat{P} \sim P^N}[\tau(\hat{P}, \hat{P}) - \tau(\hat{P}, P)], \quad N \geq 1, \quad P \in \mathcal{P}(\mathcal{X}), \tag{20}$$

where $\hat{P} \coloneqq \frac{1}{N}\sum_{n=1}^{N}\delta_{X_n}$ *denotes the empirical distribution of an i.i.d. sample* $\mathcal{D} = \{X_n\}_{n=1}^N \sim P$*. Then, we have*

$$b^N(P;\tau) = \frac{1}{2N}\mathbb{E}_{X\sim P}\left[2\tau_{P,P}^{(1,1)}(\delta_X - P, \delta_X - P) + \tau_{P,P}^{(0,2)}(\delta_X - P, \delta_X - P)\right] + O(1/N^2).$$

Proof of Proposition A.2. Assume that τ admits the Taylor expansion, i.e.,

$$\tau(P_1, P_2) = \sum_{k=0}^{\infty}\frac{1}{k!}\sum_{\ell=0}^{k}\binom{k}{\ell}\tau_{P,P}^{(\ell,k-\ell)}((P_1 - P)^{\otimes\ell}, (P_2 - P)^{\otimes(k-\ell)})$$

for all $P, P_1, P_2 \in \mathcal{P}(\mathcal{X})$. Thus, substituting $\hat{P}$ for P_1 and P_2 and taking the expectation with respect to $\hat{P}$, we have

$$\begin{aligned}
&\mathbb{E}[\tau(\hat{P}, \hat{P})] \\
=&\sum_{k=0}^{\infty}\frac{1}{k!}\sum_{\ell=0}^{k}\binom{k}{\ell}\mathbb{E}[\tau_{P,P}^{(\ell,k-\ell)}((\hat{P} - P)^{\otimes l}, (\hat{P} - P)^{\otimes(k-l)})] \\
=&\sum_{k=0}^{\infty}\frac{1}{k!}\sum_{\ell=0}^{k}\binom{k}{\ell}\frac{1}{N^k}\sum_{n_1,\ldots,n_k=1}^{N}\mathbb{E}\left[\tau_{P,P}^{(\ell,k-\ell)}\left(\bigotimes_{i=1}^{l}(\delta_{X_{n_i}} - P)), \bigotimes_{j=l+1}^{k}(\delta_{X_{n_j}} - P))\right)\right],
\end{aligned}$$

where the first equality is owing to the entirety of τ and the last equality follows from the multilinearity of the Fréchet derivatives.

Let us calculate each of the summands using the stochastic expansion formula (Lemma A.1) in the following. The summand of $k = 0$ is $\tau(P, P)$. The summand of $k = 1$ is calculated as,

$$\begin{aligned}
&\frac{1}{1!}\sum_{\ell=0}^{1}\binom{1}{\ell}\frac{1}{N}\sum_{n_1=1}^{N}\mathbb{E}\left[\tau_{P,P}^{(\ell,1-\ell)}\left(\bigotimes_{i=1}^{l}(\delta_{X_{n_i}} - P)), \bigotimes_{j=l+1}^{1}(\delta_{X_{n_j}} - P))\right)\right] \\
=&\frac{1}{N}\sum_{n_1=1}^{N}\mathbb{E}\left[\tau_{P,P}^{(0,1)}(\delta_{X_{n_1}} - P) + \tau_{P,P}^{(1,0)}(\delta_{X_{n_1}} - P)\right] \\
=&0.
\end{aligned}$$

The summand of $k = 2$ is calculated as

$$\begin{aligned}
&\frac{1}{2!}\sum_{\ell=0}^{2}\binom{2}{\ell}\frac{1}{N^2}\sum_{n_1,n_2=1}^{N}\mathbb{E}\left[\tau_{P,P}^{(\ell,2-\ell)}\left(\bigotimes_{i=1}^{l}(\delta_{X_{n_i}} - P)), \bigotimes_{j=l+1}^{2}(\delta_{X_{n_j}} - P))\right)\right] \\
=&\frac{1}{2N^2}\sum_{\ell=0}^{2}\binom{2}{\ell}\left(\sum_{n_1=n_2} + \sum_{n_1\neq n_2}\right)\mathbb{E}\left[\tau_{P,P}^{(\ell,2-\ell)}\left(\bigotimes_{i=1}^{l}(\delta_{X_{n_i}} - P)), \bigotimes_{j=l+1}^{2}(\delta_{X_{n_j}} - P))\right)\right] \\
=&\frac{1}{2N^2}\sum_{\ell=0}^{2}\binom{2}{\ell}\sum_{n=1}^{N}\mathbb{E}\left[\tau_{P,P}^{(\ell,2-\ell)}(\delta_{X_n} - P, \delta_{X_n} - P)\right] \\
=&\frac{1}{2N}\sum_{\ell=0}^{2}\binom{2}{\ell}\mathbb{E}_{X\sim P}\left[\tau_{P,P}^{(\ell,2-\ell)}(\delta_X - P, \delta_X - P)\right].
\end{aligned}$$

It is also shown that the summands of $k \geq 3$ is $O(1/N^{\lceil\frac{k}{2}\rceil})$ by Lemma A.1. Summing up, we have

$$\mathbb{E}[\tau(\hat{P}, \hat{P})] = \tau(P, P) + \frac{1}{2N}\sum_{\ell=0}^{2}\binom{2}{\ell}\mathbb{E}_{X\sim P}\left[\tau_{P,P}^{(\ell,2-\ell)}(\delta_X - P, \delta_X - P)\right] + O(1/N^2).$$

The other term, $\tau(\hat{P}, P)$, can be expanded in a similar way, and we have,

$$\mathbb{E}[\tau(\hat{P}, P)] = \tau(P, P) + \frac{1}{2N}\mathbb{E}_{X\sim P}[\tau_{P,P}^{(2,0)}(\delta_X - P, \delta_X - P)] + O(1/N^2).$$

Combining these two expansions yields the desired result. □

A.3. Result 2: Asymptotic Analysis of the Train-Test-Split Estimator for the Reuse-and-Finiteness Bias

Following the previous section, this section analyzes the train-test-split estimator for the reuse-and-finiteness bias discussed in Section 4.2.1. We provde a more general statement in Proposition A.3 in the same way as the previous section, and prove Corollary 4.1 as a corollary.

Proposition A.3. *Suppose we randomly divide the sample such that* $|\mathcal{D}_{\text{train}}| : |\mathcal{D}_{\text{test}}| = \lambda : (1-\lambda)$ *for some* $\lambda \in (0,1)$*, and let* $\hat{P}_{\text{train}}$ *and* $\hat{P}_{\text{test}}$ *be the corresponding empirical distributions. Let us define,*

$$b^N_{split}(\hat{P};\tau) = \mathbb{E}\left[\tau(\hat{P}_{\text{train}},\hat{P}_{\text{train}}) - \tau(\hat{P}_{\text{train}},\hat{P}_{\text{test}})\right]. \tag{21}$$

Then, the following holds:

$$\mathbb{E}_{\hat{P}\sim P^N}[b^N_{\text{split}}(\hat{P};\tau)] = b^N(P;\tau) + O(1/N). \tag{22}$$

Proof. By expanding τ around $(P_1,P_2)=(P,P)$, we have,

$$\tau(P_1,P_2) = \sum_{k=0}^{\infty}\frac{1}{k!}\sum_{\ell=0}^{k}\binom{k}{\ell}\tau^{(\ell,k-\ell)}_{P,P}((P_1-P)^{\otimes\ell},(P_2-P)^{\otimes(k-\ell)}).$$

Suppose we split the sample $\mathcal{D}$ into $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$ whose sample sizes are N_{train} and N_{test} respectively such that $\mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{test}} = \mathcal{D}$, $\mathcal{D}_{\text{train}} \cap \mathcal{D}_{\text{test}} = \emptyset$, and $N_{\text{train}} : N_{\text{test}} = \lambda : (1-\lambda)$ $(0<\lambda<1)$. Then, we have,

$$\mathbb{E}_{\hat{P}\sim P^N}[b^N_{\text{split}}(\hat{P};\tau)] = \mathbb{E}_{\substack{\hat{P}_{\text{train}}\sim P^{N_{\text{train}}}\\ \hat{P}_{\text{test}}\sim P^{N_{\text{test}}}}}\left[\tau(\hat{P}_{\text{train}},\hat{P}_{\text{train}}) - \tau(\hat{P}_{\text{train}},\hat{P}_{\text{test}})\right].$$

Each of the terms in the right-hand side can be expanded as follows:

$$\mathbb{E}_{\hat{P}_{\text{train}}\sim P^{N_{\text{train}}}}\tau(\hat{P}_{\text{train}},\hat{P}_{\text{train}}) = \tau(P,P) + \frac{1}{2N_{\text{train}}}\sum_{l=0}^{2}\binom{2}{l}\mathbb{E}_{X\sim P}\left[\tau^{(l,2-l)}_{P,P}(\delta_X - P, \delta_X - P)\right] + O(1/N^2_{\text{train}}), \tag{23}$$

$$\begin{aligned}\mathbb{E}_{\substack{\hat{P}_{\text{train}}\sim P^{N_{\text{train}}}\\ \hat{P}_{\text{test}}\sim P^{N_{\text{test}}}}}\tau(\hat{P}_{\text{train}},\hat{P}_{\text{test}}) =& \tau(P,P) + \frac{1}{2N_{\text{train}}}\mathbb{E}_{X\sim P}\left[\tau^{(2,0)}_{P,P}(\delta_X - P, \delta_X - P)\right]\\ &+ \frac{1}{2N_{\text{test}}}\mathbb{E}_{X\sim P}\left[\tau^{(0,2)}_{P,P}(\delta_X - P, \delta_X - P)\right] + O(1/N^2_{\text{test}}).\end{aligned} \tag{24}$$

By combining the above expansions, we have,

$$\begin{aligned}\mathbb{E}_{\hat{P}\sim P^N}[b^N_{\text{split}}(\hat{P};\tau)] =& \frac{1}{\lambda N}\mathbb{E}_{X\sim P}\left[\tau^{(1,1)}_{P,P}(\delta_X - P, \delta_X - P)\right] + \frac{1}{2N}\left(\frac{1}{\lambda} - \frac{1}{1-\lambda}\right)\mathbb{E}_{X\sim P}\left[\tau^{(0,2)}_{P,P}(\delta_X - P, \delta_X - P)\right]\\ &+ O(1/N^2) = b^N(P;\tau) + O(1/N).\end{aligned}$$

□

A.4. Result 3: Asymptotic Analysis of the Bootstrap Estimator for the Reuse-and-Finiteness Bias

In Section 4.2.2, we propose to use a bootstrap method to estimate the reuse-and-finiteness bias. The following proposition shows that $\tau(\hat{P},\hat{P})$ with bias correction by the bootstrap estimator $b^N(\hat{P};\tau)$ is the second-order biased estimator of $\tau(\hat{P},P)$, which is better than the plug-in estimator $\tau(\hat{P},\hat{P})$.

Proposition A.4 (Bias of a bootstrap estimator of the reuse-and-finiteness bias)**.** *Take* P*,* $\hat{P}$ *and* $b^N(P;\tau)$ *as in Proposition A.2. Then, we have*

$$\mathbb{E}[b^N(\hat{P};\tau)] = b^N(P;\tau) + O(1/N^2),$$

which implies,

$$\mathbb{E}[\tau(\hat{P},\hat{P}) - b^N(\hat{P};\tau)] = \mathbb{E}[\tau(\hat{P},P)] + O(1/N^2). \tag{25}$$

Moreover,

$$\sqrt{\mathbb{E}\{b^N(\hat{P};\tau) - b^N(P;\tau)\}^2} = O(1/N^{1.5}).$$

Proof. Define the coefficient of the $O(1/N)$ term of the bias by,

$$\beta(P;\tau) \coloneqq \mathbb{E}_{X\sim P}\left[2\tau^{(1,1)}_{P,P}(\delta_X - P, \delta_X - P) + \tau^{(0,2)}_{P,P}(\delta_X - P, \delta_X - P)\right],$$

so that $b^N(P;\tau) = \frac{1}{2N}\beta(P;\tau) + O(1/N^2)$. Note that $\beta(P;\tau)$ is independent of N. Thus, the stochastic expansion of $\beta(\hat{P};\tau)$ at P gives

$$\begin{aligned}
\mathbb{E}[\beta(\hat{P};\tau)] &= \sum_{k=0}^{\infty}\frac{1}{k!}\mathbb{E}\left[\beta^{(k)}_P((\hat{P}-P)^{\otimes k};\tau)\right] \\
&= \sum_{k=0}^{\infty}\frac{1}{k!}\frac{1}{N^k}\sum_{n_1,\dots,n_k=1}^{N}\mathbb{E}\left[\beta^{(k)}_P(\delta_{X_{n_1}} - P, \dots, \delta_{X_{n_k}} - P;\tau)\right] \\
&= \beta(P;\tau) + \frac{1}{2N}\mathbb{E}_{X\sim P}\left[\beta^{(2)}_P((\delta_X - P)^{\otimes 2};\tau)\right] + O(1/N^2).
\end{aligned}$$

Substituting this back to $\mathbb{E}[b^N(\hat{P};\tau)] = \frac{1}{2N}\mathbb{E}[\beta(\hat{P};\tau)] + O(1/N^2)$, we have

$$\begin{aligned}
\mathbb{E}[b^N(\hat{P};\tau)] &= \frac{1}{2N}\left[\beta(P;\tau) + \frac{1}{2N}\mathbb{E}_{X\sim P}\left[\beta^{(2)}_P((\delta_X - P)^{\otimes 2};\tau)\right] + O(1/N^2)\right] + O(1/N^2) \\
&= \frac{1}{2N}\beta(P;\tau) + O(N^{-2}) = b^N(P;\tau) + O(1/N^2),
\end{aligned}$$

which is the first desired result.

Similarly, the stochastic expansion gives

$$\begin{aligned}
&\mathbb{E}\{\beta(\hat{P};\tau) - \beta(P;\tau)\}^2 \\
&= \mathbb{E}\left\{\sum_{k=1}^{\infty}\frac{1}{k!}\beta^{(k)}_P((\hat{P}-P)^{\otimes k};\tau)\right\}^2 \\
&= \sum_{k=1}^{\infty}\sum_{\ell=1}^{\infty}\frac{1}{k!}\frac{1}{\ell!}\mathbb{E}\left[\beta^{(k)}_P((\hat{P}-P)^{\otimes k};\tau)\beta^{(\ell)}_P((\hat{P}-P)^{\otimes \ell};\tau)\right] \\
&= \sum_{k=1}^{\infty}\sum_{\ell=1}^{\infty}\frac{1}{k!}\frac{1}{\ell!}\frac{1}{N^k}\frac{1}{N^\ell}\sum_{n_1,\dots,n_k=1}^{N}\sum_{n'_1,\dots,n'_\ell=1}^{N}\mathbb{E}\left[\beta^{(k)}_P(\delta_{X_{n_1}} - P, \dots, \delta_{X_{n_k}} - P;\tau)\beta^{(\ell)}_P(\delta_{X_{n'_1}} - P, \dots, \delta_{X_{n'_\ell}} - P;\tau)\right] \\
&= \frac{1}{N}\mathbb{E}_{X\sim P}\left[\beta^{(1)}_P(\delta_X - P;\tau)\beta^{(1)}_P(\delta_X - P;\tau)\right] + O(1/N^2) \\
&= O(1/N),
\end{aligned}$$

and thus

$$\sqrt{\mathbb{E}\{b^N(\hat{P};\tau) - b^N(P;\tau)\}^2} = \sqrt{\frac{1}{4N^2}\mathbb{E}\{\beta(\hat{P};\tau) - \beta(P;\tau)\}^2 + O\left(1/N^4\right)} = O\left(1/N^{1.5}\right).$$

□

B. Reuse-and-Finiteness Bias for Optimal Generators

This section proves that the reuse-and-finiteness bias is non-negative for optimal generators under certain assumptions (Proposition 3.3).

Proof of Proposition 3.3. Let $\alpha_G(Q) \coloneqq \operatorname{argmax}_{G\in\mathcal{G}} J_{\mathrm{PI}}(G, \alpha_f(Q))$ for any distribution $Q \in \mathcal{P}(\mathcal{M}\times\mathbb{R})$. Since $J_{\mathrm{PI}}(\alpha_G(\hat{P}), \alpha_f(\hat{P})) \geq J_{\mathrm{PI}}(\alpha_G(P), \alpha_f(\hat{P}))$ holds for any empirical distribution $\hat{P}$, taking the expectations of $\hat{P}\sim P^N$ yields,

$$\mathbb{E}_{\hat{P}\sim P^N} J_{\mathrm{PI}}(\alpha_G(\hat{P}), \alpha_f(\hat{P})) \geq \mathbb{E}_{\hat{P}\sim P^N} J_{\mathrm{PI}}(\alpha_G(P), \alpha_f(\hat{P})) = J_{\mathrm{PI}}(\alpha_G(P), \alpha_f(P)). \tag{26}$$

Since $J_{\mathrm{PI}}(\alpha_G(P), \alpha_f(P)) \geq J_{\mathrm{PI}}(\alpha_G(\hat{P}), \alpha_f(P))$ holds for any $\hat{P}$, taking the expectations of $\hat{P} \sim P^N$ yields,

$$J_{\mathrm{PI}}(\alpha_G(P), \alpha_f(P)) \geq \mathbb{E}_{\hat{P}\sim P^N} J_{\mathrm{PI}}(\alpha_G(\hat{P}), \alpha_f(P)). \tag{27}$$

By combining these, we obtain the inequality. □

C. More Insights into the Train-Test Split Method

This section investigates more on the train-test split method. First, Proposition C.1 shows that the test performance estimated by the train-test split method is biased by $O(1/N)$, which is the same order as the difference between $\mathbb{E}\tau(\hat{P}, P)$ and $\tau(P, P)$. In contrast, the test performance estimated by bootstrap (*e.g.*, Equation (25)) is biased by $O(1/N^2)$, which is better than the train-test split method.

Proposition C.1. *Test performance estimation by train-test split also fails:*

$$\mathbb{E}\tau(\hat{P}_{\mathrm{train}}, \hat{P}_{\mathrm{test}}) = \mathbb{E}_{\hat{P}\sim P^N}\tau(\hat{P}, P) + O(1/N),$$

where $\mathbb{E}_{\hat{P}\sim P^N}\tau(\hat{P}, P) = \tau(P, P) + O(1/N)$ *holds.*

Proof. Recall that we split the sample $\mathcal{D}$ into $\mathcal{D}_{\mathrm{train}}$ and $\mathcal{D}_{\mathrm{test}}$ with sample sizes N_{train} and N_{test} such that $N_{\mathrm{train}} : N_{\mathrm{test}} = \lambda : (1-\lambda)$. Equation (24) suggests that,

$$\begin{aligned}\mathbb{E}_{\substack{\hat{P}_{\mathrm{train}}\sim P^{N_{\mathrm{train}}} \\ \hat{P}_{\mathrm{test}}\sim P^{N_{\mathrm{test}}}}}\tau(\hat{P}_{\mathrm{train}}, \hat{P}_{\mathrm{test}}) =& \tau(P, P) + \frac{1}{2\lambda N}\mathbb{E}_{X\sim P}\left[\tau^{(2,0)}_{P,P}(\delta_X - P, \delta_X - P)\right] \\ &+ \frac{1}{2(1-\lambda)N}\mathbb{E}_{X\sim P}\left[\tau^{(0,2)}_{P,P}(\delta_X - P, \delta_X - P)\right] + O(1/N^2),\end{aligned} \tag{28}$$

whereas the following holds:

$$\mathbb{E}_{\hat{P}\sim P^N}\tau(\hat{P}, P) = \tau(P, P) + \frac{1}{2N}\mathbb{E}_{X\sim P}\left[\tau^{(2,0)}_{P,P}(\delta_X - P, \delta_X - P)\right] + O(1/N^2). \tag{29}$$

By comparing the equations above, the coefficients of $O(1/N)$ terms do not coincide, and therefore, the test performance estimated by the train-test split is biased by $O(1/N)$. □

The train-test split method is less biased if the test set is sufficiently large (Proposition C.2). This proposition is not useful in practice, but it guarantees that we can estimate $\tau(\hat{P}_{\mathrm{train}}, P)$ by using a sufficiently large test sample.

Proposition C.2. *In the limit of* $N_{\mathrm{test}} \to \infty$, $\tau(\hat{P}_{\mathrm{train}}, \hat{P}_{\mathrm{test}})$ *coincides with* $\tau(\hat{P}_{\mathrm{train}}, P)$ *in expectation up to* $O(1/N_{\mathrm{train}})$*-term.*

Proof of Proposition C.2. Equation (24) suggests that,

$$\lim_{N_{\mathrm{test}}\to\infty} \mathbb{E}_{\substack{\hat{P}_{\mathrm{train}}\sim P^{N_{\mathrm{train}}} \\ \hat{P}_{\mathrm{test}}\sim P^{N_{\mathrm{test}}}}}\tau(\hat{P}_{\mathrm{train}}, \hat{P}_{\mathrm{test}}) = \tau(P, P) + \frac{1}{2N_{\mathrm{train}}}\mathbb{E}_{X\sim P}\left[\tau^{(2,0)}_{P,P}(\delta_X - P, \delta_X - P)\right] + O(1/N^2_{\mathrm{train}}), \tag{30}$$

which coincides with $\mathbb{E}_{\hat{P}_{\mathrm{train}}\sim P^{N_{\mathrm{train}}}}\tau(\hat{P}_{\mathrm{train}}, P)$ up to $O(1/N_{\mathrm{train}})$ term. □

Let us finally discuss why the train-test split method is less accurate in our setting, whereas it is a common practice in supervised learning. The key difference is that $\tau(P_1, P_2)$ is non-linear with respect to P_2 in our setting, while it is linear in the setting of supervised learning. Let us re-define $\tau(P_1, P_2)$ as an abstract performance estimator of a data-dependent algorithm using P_1 evaluated by another data-dependent algorithm using P_2. The evaluation of a supervised learning algorithm f can be instantiated as follows:

$$\tau(P_1, P_2) = \mathbb{E}_{Z\sim P_2}\ell(f(P_1), Z), \tag{31}$$

where $Z = (X, Y)$ is an example and ℓ denotes a loss function. The key observation is that Equation (31) is linear in P_2, whereas the performance estimator in our setting is in general non-linear in P_2. If $\tau(P_1, P_2)$ is linear in P_2, $\tau^{(0,2)}_{P,P}(\delta_X - P, \delta_X - P) = 0$ holds, and therefore, the train-test split estimator coincides with the true bias up to $O(1/N)$ and is biased by $O(1/N^2)$.

D. Experimental Settings

In this section, we introduce the details of our experimental settings. While we follow the environment and agent developed by Gottipati et al. (2020) as much as possible, we made some modifications for consistency with our setting. The main difference from the original environment is that we employ a finite-horizon reinforcement learning rather than an infinite-horizon RL formulation. The environment and agent are modified accordingly.

D.1. Environment

The state space $\mathcal{S}$ is the set of molecules. The action space $\mathcal{A}$ is the direct product of the set of reaction templates and that of reactants. In particular, let us represent the set of reactants by their feature vectors consisting of 35 molecular descriptors used by Gottipati et al. (2020), $\mathcal{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_L\} \subset \mathbb{R}^{35}$, and the set of reaction templates by $\Omega = \{\omega_1, \ldots, \omega_W\}$, and the action space is defined as $\mathcal{A} = \mathbb{R}^{35} \times \Omega$. Given an action $(\mathbf{v}, \omega_w) \in \mathcal{A}$ at the current state s_h, the environment transits to the next state as follows.

1. If the reaction template ω_w requires one reactant, the reaction template is applied to the current mol s_h.
   (a) If the reaction succeeds, one of the possible products is randomly selected as the next state.
   (b) If it fails, the current molecule is set as the next state.
2. If the reaction template ω_w requires two reactants, the next state is defined as follows.
   (a) Assume that the current molecule is used as the first reactant. The set of reactants is sorted by the distance to the query vector $\mathbf{v}$ in ascending order, and the second reactant is selected by the one with the smallest distance of those in the set which can be reacted with the first reactant using the reaction template ω_w. As a result, a set of possible products is obtained.
   (b) Assuming that the current molecules is used as the second reactant, the first reactant is selected in the same way as the previous procedure, and another set of possible products is obtained.
   (c) One of these possible products is set as the next state.

Note that all of the reaction templates require no more than two reactants, and the above two cases cover all.

D.2. Agent

We employ an actor-critic architecture following the existing work (Gottipati et al., 2020). To adapt to the finite-horizon setting, we prepare $H+1$ copies of actors and critics, and use each copy for each step. Let $\pi_h(a \mid s; \theta_h)$ be the actor and $Q_h(s, a; \phi_h)$ be the critic at step $h \in [H+1]$. The learning algorithm repeatedly obtains pairs of an actor and a critic for each $h = H, H-1, \ldots, 0$ backwardly. At each step, the parameters of the actor and critic are updated as follows for $t = 0, 1, 2, \ldots, T-1$:

$$\theta_h^{(t+1)} \leftarrow \theta_h^{(t)} + \alpha^{(t)} \frac{\partial}{\partial \theta_h} \mathbb{E}_{(s,a)\sim\bar{\mathcal{D}}_h^{(t)}} \left[Q_h(s, \pi_h(s; \theta_h^{(t)}); \phi_h^{(t)}) \right], \tag{32}$$

$$\phi_h^{(t+1)} \leftarrow \phi_h^{(t)} - \beta^{(t)} \frac{\partial}{\partial \phi_h} \mathbb{E}_{(s,a)\sim\bar{\mathcal{D}}_h^{(t)}} \left[\left(Q_h(s, a; \phi_h^{(t)}) - r - Q_{h+1}(s', \pi_{h+1}(s'; \theta_{h+1}); \phi_{h+1}) \right)^2 \right], \tag{33}$$

where $\alpha^{(t)}$ and $\beta^{(t)}$ are learning rates and $\bar{\mathcal{D}}_h^{(t)}$ is a mini-batch of state-action pairs at step h drawn from a sample of trajectories $\bar{\mathcal{D}}$.

The actor $\pi_h(a \mid s; \theta_h)$ consists of a *template selector*, which receives the current state and outputs a reaction template to be aplied at the next step, followed by a *reactant selector*, which receives the current state and the output of the template selector and outputs a query vector of a reactant, $\mathbf{v} \in \mathbb{R}^{35}$.

The template selector consists of a Morgan fingerprint module with radius 2 and 1024 bits, followed by a fully-connected neural network with one hidden layer with 256 units, interleaved with a softplus activation except for the last layer. The reactant selector consists of a Morgan fingerprint module with radius 2 and 1024 bits, which is then combined with the output of the template selector and is fed into a fully-connected neural network with one hidden layer with 256 units, interleaved with a softplus activation except for the last layer.

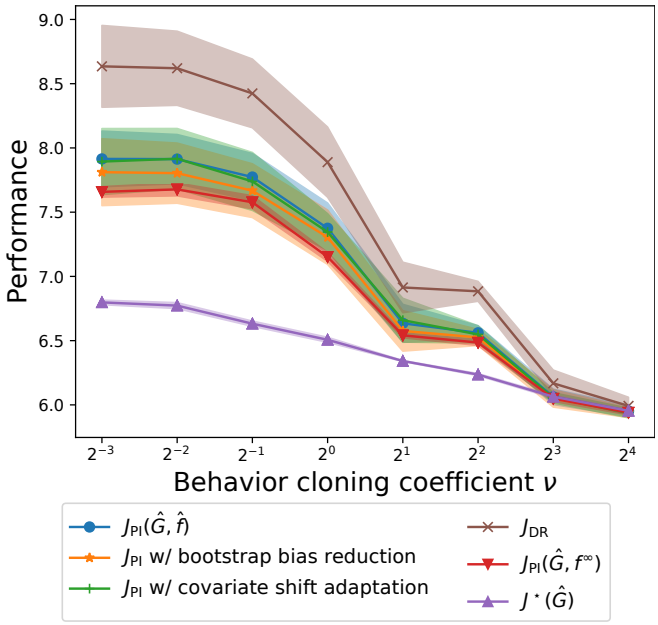

Figure 3. Experimental result with the doubly-robust performance estimator J_{DR}.

The critic network is a mapping from the current state and the outputs of the template selector and reactant selector to the estimated value of the current state and the action generated by the actor. The current molecule is converted into a continuous vector using the Morgan fingerprint with radius 2 and 1024 bits, which are concatenated with the outputs and fed into a fully-connected neural network with one hidden layer with 256 units, interleaved with a softplus activation except for the last layer.

For the first 500 steps, we only update the parameters of the critic, fixing those of the actor, and after that, both of them are updated for another 1,500 steps. They are optimized by AdaGrad (Duchi et al., 2011) with initial learning rate 4×10^{-4} and batch size 64.

D.3. Evaluators

The reward model $\hat{f}$ is a fully-connected neural network with one hidden layer of 96 units with softplus activations except for the last layer. It is trained by minimizing the risk defined over $S \sim P$ by AdaGrad for 10^4 steps with initial learning rate 10^{-3} and batch size 128.

The importance weight model, KuLSIF, has a regularization hyperparameter λ_w. We chose it from $\{2^{-20}, \ldots, 2^0\}$ by leave-one-out cross validation.

D.4. Computational Environment

We implement the whole simulation in Python 3.9.0. All of the chemistry-related operations including the template-based chemical reaction is implemented by RDKit (2021.09.3). We used an IBM Cloud with 16×2.10GHz CPU cores, 128GB memory, and two NVIDIA Tesla P100 GPUs.

E. Full Experimental Result

We report the performance estimates by the doubly-robust performance estimator J_{DR} in Figure 3. As evident from it, the scores are over-estimated primarily due to the over-estimation by the importance sampling performance estimator.