Diagnosing and fixing common problems in Bayesian optimization for molecule design

Austin Tripp [1] **José Miguel Hernández-Lobato** [1]

Abstract

Bayesian optimization (BO) is a principled approach to molecular design tasks. In this paper we explain three pitfalls of BO which can cause poor empirical performance: an incorrect prior width, over-smoothing, and inadequate acquisition function maximization. We show that with these issues addressed, even a basic BO setup is able to achieve the highest overall performance on the PMO benchmark for molecule design (Gao et al., 2022). These results suggest that BO may benefit from more attention in the machine learning for molecules community.

1. Introduction

Many problems in drug discovery can be summarized as finding molecules with desirable properties. This is often formalized as maximizing a property function $f : \mathcal{M} \mapsto \mathbb{R}$, where $\mathcal{M}$ denotes the space of molecules. The challenge of this problem is the immense size of the search space $\mathcal{M}$: out of an estimated 10^{60} possible molecules (Bohacek et al., 1996), only a minuscule fraction can be tested experimentally (perhaps 10^2–10^4). Therefore, algorithms for molecule design must operate very efficiently, making the best use of their experimental budget.

Despite the need for efficiency, the current most popular algorithms for molecule design all seem to heavily rely on *random* exploration. Genetic algorithms (GAs) and their variants randomly mutate and combine known molecules (Jensen, 2019; Nigam et al., 2020). Algorithms based on reinforcement learning (RL) such as REINVENT (Olivecrona et al., 2017; Blaschke et al., 2020) and GFlowNets (Bengio et al., 2021; 2023) instead make random perturbations to a molecule generation policy. In both cases, because the exploration is random it is likely to be inefficient.

[1]Department of Engineering, University of Cambridge, Cambridge, UK. Correspondence to: Austin Tripp <ajt212[at]cam.ac.uk>.

Accepted at ICML 2024 AI for Science workshop.

In contrast, Bayesian optimization (BO) stands out as a principled alternative which performs *deliberate* exploration (Garnett, 2023). By explicitly using prior knowledge to model molecular properties, BO algorithms can make a precise trade-off between exploration (testing new molecules) and exploitation (testing molecules similar to the best known ones). Because of this, one might expect BO methods to be state-of-the-art in this field. Surprisingly however, prior work has shown that BO under-performs RL/GA methods (Gao et al., 2022).

In this short paper, we argue that poor BO performance in prior works may essentially be due to poor tuning of hyperparameters. To show this, we first introduce BO (§2) and explain several ways in which certain choices of hyperparameters can lead to *predictably* poor optimization performance (§3). Second, we show that with the right settings a basic BO setup achieves the best reported performance on the PMO benchmark for molecular optimization algorithms (Gao et al., 2022). We conclude with a brief evaluation of the pros and cons of BO, arguing that while it is not perfect, it should likely receive more attention from the community (§5).

2. Background on Bayesian optimization

Let $\mathcal{X}$ represent an input space. Let $\mathbb{P}$ denote the probability of an event, $\mathbb{E}$ denote expected value, and $\mathbb{V}$ denote variance. The most basic form of Bayesian optimization (BO) seeks

$$x^* = \arg\max_{x \in \mathcal{X}} f(x) \,, \tag{1}$$

namely an input which maximizes an *objective function* $f : \mathcal{X} \mapsto \mathbb{R}$. At the heart of BO is a *probabilistic surrogate model*, which specifies a *distribution* over surrogate models $\hat{f} : \mathcal{X} \mapsto \mathbb{R}$ for the objective function f. We will denote a general probabilistic surrogate model by $p(\hat{f})$.

BO uses $p(\hat{f})$ to choose inputs to evaluate, typically choosing an input x which maximizes an *acquisition function* α. An intuitive example of an acquisition function is the *probability of improvement* (PI) (Garnett, 2023, §7.5)

$$\alpha_{\text{PI}}\left(x; p(\hat{f}), y_{\text{best}}\right) = \mathbb{P}_{\hat{f} \sim p(\hat{f})}\left[\hat{f}(x) > y_{\text{best}}\right] \,, \tag{2}$$

which measures the probability that $f(x)$ will improve upon the incumbent best measurement y_{best}: an intuitively reasonable criterion to select points for evaluation.

Pseudocode for a general BO loop is given in Algorithm 1. The key lines of this algorithm are line 2 (which defines the probabilistic surrogate model) and line 3 (which uses an acquisition function to select an input to evaluate).[1] The rest of this section will discuss these steps in more detail.

Algorithm 1 General Bayesian optimization loop.

Require: Input dataset $\mathcal{D}_0 = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, acquisition function α

1: **for** i in $1, 2, \ldots$ **do**
2: Fit $p_i(\hat{f})$ to dataset $\mathcal{D}_{i-1}$
3: Select $x_i = \arg\max_x \alpha_i(x; p_i(\hat{f}))$
4: Acquire label y_i for x_i
5: $\mathcal{D}_i \leftarrow \mathcal{D}_{i-1} \cup \{(x_i, y_i)\}$
6: **if** computational budget is exhausted **then**
7: **return** $\mathcal{D}_i$ {Terminate}
8: **end if**
9: **end for**

2.1. Gaussian process surrogate models

Gaussian processes (GPs) are the most commonly-used class of probabilistic surrogate models, and therefore we will introduce them briefly here. A GP assumes the that *joint* distribution of the observed data is Gaussian, whose mean is given by a mean function $\mu : \mathcal{X} \mapsto \mathbb{R}$, and whose covariance is given by a positive-definite *kernel function* $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$. When $\mathcal{X} = \mathbb{R}^n$, a common choice of kernel function is the RBF kernel, defined as

$$k_{\text{RBF}}(x, x') = \sigma^2 \exp\left(\frac{-\|x - x'\|^2}{2\ell^2}\right). \tag{3}$$

The hyperparameter σ is referred to as the *kernel amplitude* (because the marginal prior distribution for every input is a Gaussian with standard deviation σ), while ℓ is referred to as the *lengthscale*.

The primary appeal of GP models is that their posterior distribution has an analytic solution, evading the need for approximate inference techniques like MCMC. The formulas for the analytic solution can be found in numerous textbooks (Rasmussen & Williams, 2006; Garnett, 2023). This allows the model fitting step in line 2 to be performed efficiently and reliably.

GP surrogate models will be used in the remainder of this paper. However, we emphasize that BO does not *require* the use of GP models: Bayesian neural networks or ensembles are viable alternatives.

[1] To allow the acquisition function to vary over iterations, we use the notation α_i.

2.2. Acquisition functions

Despite its simplicity, the PI acquisition function in equation 2 is seldom used in practice, chiefly because it does not account for the *magnitude* of the improvement (so large improvements are treated the same as small improvements). Instead, many people use *expected improvement* (EI)

$$\alpha_{\text{EI}}\left(x; p(\hat{f}), y_{\text{best}}\right) = \mathbb{E}_{\hat{f} \sim p(\hat{f})}\left[\max\left(0, \hat{f}(x) - y_{\text{best}}\right)\right], \tag{4}$$

which measures the average *amount* by which $f(x)$ is predicted to improve over y_{best}. Another common acquisition is the *upper confidence bound* (UCB)

$$\alpha_{\text{UCB}}\left(x; p(\hat{f})\right) = \mathbb{E}_{\hat{f}}\left[\hat{f}(x)\right] + \beta\sqrt{\mathbb{V}_{\hat{f}}\left[\hat{f}(x)\right]}, \tag{5}$$

which is the mean prediction plus the standard deviation weighted by β. There are many other choices of acquisition function: Garnett (2023, Chapter 7) gives a good introduction to them.

Importantly, the acquisition function is *not* something which should be chosen arbitrarily. Because the acquisition function specifies (implicitly or explicitly) the explore-exploit trade-off, it should be chosen with that in mind. In general, EI tends to be *exploitative*, while UCB becomes more exploitative as $\beta \to 0$ and more exploratory as $\beta \to \infty$.

3. Common Bayesian optimization pitfalls

While there is are no universal rules to optimally tune all hyperparameters in BO, some hyperparameter settings have intuitive and predictable failure modes. We explain three possible failure modes with an illustrative example in 1D, shown in Figure 1. This setup is chosen to be vaguely analogous to molecule design: some molecules near a local optimum are known, but other more promising optima are unexplored. We use a GP with an RBF kernel as the surrogate model (typically the default choice in most GP libraries) with low observation noise.

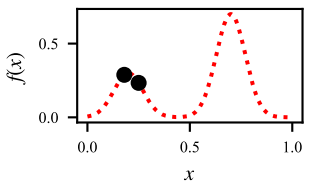

Figure 1. 1D optimization task meant to be qualitatively similar to molecular design tasks. Only a small number of data points are known (black dots), none of which are near the global optimum of the unknown function (red dashed line).

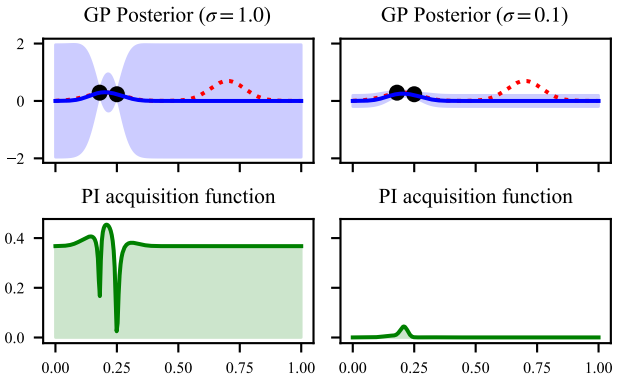

Figure 2. Effect of prior width parameter σ in a GP model, illustrating "prior width" pitfall (§3.1). Low values of σ cause the model to predict lower returns from exploration.

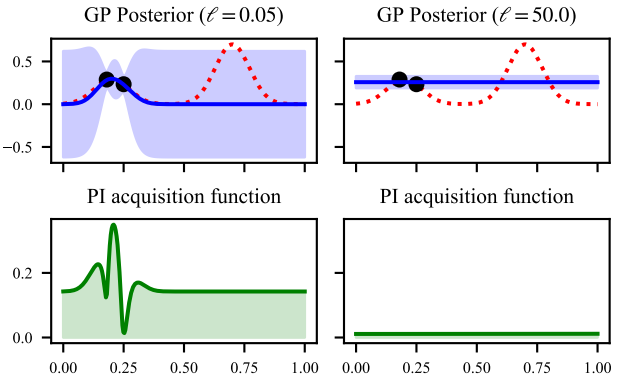

Figure 3. Effect of lengthscale parameter ℓ in a GP model, illustrating "over-smoothing" pitfall (§3.2). High values of ℓ also imply lower returns from exploring inputs near known inputs.

3.1. Pitfall #1: prior width

A model $p(\hat{f})$ will imply a range of values that f is likely to take, which we refer to as the *prior width*. For example, with a GP model, the predictive standard deviation can be interpreted as a prior width, and can be controlled by the parameter σ (equation 3). The prior width directly determines the predicted gains from exploring away from the training data. Figure 2 directly shows the consequence of this, using prior widths of 0.1 and 1.0. When $\sigma = 0.1$, the points near the left are predicted to be nearly optimal, and there is no predicted gain from exploring the right side of the space. In contrast, when $\sigma = 1.0$, the points near the right have a reasonably high predicted probability of being better than the points on the left.

It is straightforward to see that the same principles will also hold outside of 1D examples. A general guideline is that if σ is too high, then BO algorithms will anticipate large gains from exploration and tend to be too exploratory. Conversely, if σ is too low then BO algorithms will under-explore.

3.2. Pitfall #2: over-smoothing

The probabilistic surrogate model $p(\hat{f})$ essentially encodes how measurements of known input points influence those of unmeasured points. For GPs in 1D, each point can be thought of as having a "radius" of influence around it, which is determined by the lengthscale of the kernel function (e.g. ℓ in equation 3). If this radius is too high, it can lead to overconfident predictions. Figure 3 illustrates this by showing the GP posterior using an RBF lengthscale of $\ell = 0.05$ and $\ell = 5.0$. When $\ell = 50.0$, the measurements on the left suggest that the right side is not worth exploring, which does not happen when $\ell = 0.05$.

A general guideline is that over-smoothing will result in under-exploration, while under-smoothing will result in over-exploration.

3.3. Pitfall #3: inadequate search

Line 3 requires finding an input which maximizes the acquisition function. Although in 1D this can be accomplished via a comprehensive grid search, in combinatorially large spaces like molecules inevitably only a small fraction of all candidate points may be considered. Unfortunately, popular search methods like generative models and GAs tend to propose molecules similar to known molecules. In 1D, this is a bit like only searching in a narrow interval around the known points, akin to never considering inputs on the right side of Figure 1.

Unlike the first two pitfalls, poor search should only ever result in *under*-exploration. However, longer searches will generally take more time.

4. Experiments: fixing these issues substantially improves performance

In this section we consider the application of BO to the PMO benchmark, which consists of 23 different objective functions $f : \mathcal{M} \mapsto [0, 1]$ over molecule space (Gao et al., 2022). Very few works have applied BO to this benchmark,[2] so we focus our attention to the "GP BO" baseline implemented by Gao et al. (2022). Their implementation used a basic Tanimoto kernel on molecular fingerprint features

$$k(x, x') = \sigma^2 T\left(\text{fp}(x), \text{fp}(x')\right) \ , \tag{6}$$

where T denotes the Tanimoto coefficient[3] function and fp is a function producing molecular fingerprints. They used a UCB acquisition function with random value of β

[2]Aside from Gao et al. (2022), we are only aware of Wang-Henderson et al. (2023).

[3]Also called Jaccard similarity

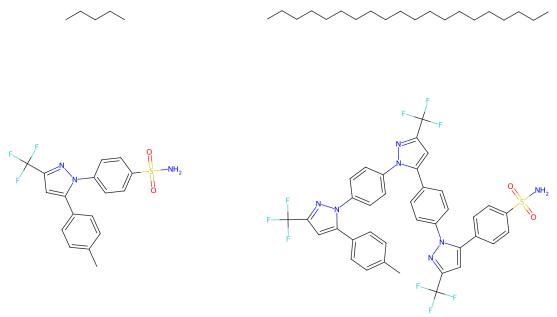

Figure 4. Two pairs of molecules whose *binary* Morgan fingerprints of radius 2 are identical. The top pair is two alkanes of different lengths, which only contain $-CH_3$ and $-CH_2-$ groups. The bottom pair is the anti-inflammatory drug molecule Celecoxib and a larger analogue with many repeated substructures. SMILES strings are given in Appendix C.

in each iteration, which was optimized using a Graph GA algorithm (Jensen, 2019). However, a close inspection of their implementation reveals potential signs of all 3 pitfalls from Section 3:

1. **Prior width**: the kernel hyperparameters are chosen by maximizing the marginal likelihood on the *starting* data, which mainly consists of molecules with poor scores. This is likely to select a lower value of σ.
2. **Over-smoothing**: a GP with a Tanimoto kernel over *binary* Morgan fingerprints is used. Since binary fingerprints track only the presence or absence of certain structures rather than their count, it is possible for molecules of vastly different sizes to be judged as highly similar. Figure 4 shows some examples.
3. **Inadequate search**: their Graph GA used a very small number of iterations compared to the batch size, such that for every molecule chosen only ≈ 6 molecules were proposed by the GA. This is a relatively low number, especially as GAs tend to propose molecules which are very similar to the starting molecules. Ultimately, this likely resulted in significant under-exploration.

To address these issues, we created a modified implementation of GP BO. To ensure a suitable prior width, we set $\sigma = 1.0$ for the GP kernel (equation 6) knowing that all objectives in the PMO benchmark lie in the interval $[0, 1]$. This ensures that the model assigns a reasonable probability to all possible values. To fix over-smoothing, we used *count* Morgan fingerprints instead of binary fingerprints. Finally, to improve the search, we tuned the genetic algorithm parameters to propose ≈ 1000 molecules per molecule chosen. We also decreased the batch size to 1 to allow for more iterations. To keep computational costs reasonable, we only ran BO for 1000 iterations (10% of the evaluation budget), and chose the remaining 9000 molecules in one large batch by maximizing the GP posterior mean. More details and a link to our code can be found in Appendix A.

As recommended by Gao et al. (2022), we report the AUC Top-10 metric, which is the normalized area under the curve of the 10th best molecule over time. The AUC Top-10 results from out experiments is shown in Table 1. The sum of AUC Top-10 scores for our GP BO method 16.303 which is not only higher than the best method from Gao et al. (2022) (REINVENT, with a score of 14.196), but also higher than subsequently reported results from Tripp & Hernández-Lobato (2023) and Kim et al. (2024). Importantly, our GP BO implementation improves upon the implementation from Gao et al. (2022) by over 3.0 points, which is about the same as the score difference between the best and 10th best methods from Gao et al. (2022). This suggests that our changes did have a significant impact.

5. Discussion

This short paper surveyed several potential failure modes of BO (§3) and showed empirically that a basic BO implementation with these issues resolved is able to achieve state-of-the-art performance on the PMO benchmark (Gao et al., 2022).

However, what this paper presents should best be thought of as a very limited pilot study, rather than a full diagnosis of potential issues in BO. Importantly, we *do not* claim that BO will work well if the three pitfalls we present are avoided. We also did not perform an ablation study, and therefore our results do not provide insight into how much each component of BO influences the overall result. Additionally, we did not experiment with changing the acquisition function, which in practice should significantly impact BO behavior. Finally, it is unclear whether results from single-task, noiseless, and unconstrained optimization will translate to real-world problems which tend to be multitask, noisy, and highly constrained.

Nevertheless, we think there are good reasons to continue research into BO algorithms for molecule design. Aside from empirical performance, the BO framework allows domain experts to incorporate their knowledge into the probabilistic surrogate model, and produces decisions which are interpretable and correctable.[4] These are highly desirable properties for practical molecule design problems. Improving surrogate models and extending BO to more complex opti-

[4]Specifically, the question of why one decision was made over another can be reduced to comparing the model's predictions for each decision, making them interpretable. If the user dislikes a decision, they can correct it by either changing the model (to change its predictions) or changing the acquisition function (to change how decisions are made from predictions).

mization settings are active research areas which plausibly still have a lot of low-hanging fruit left. Overall, we hope the reader concludes from this paper that BO is a promising technique for molecule design, and finds the explanations and fixes of common BO problems useful.

References

Bengio, E., Jain, M., Korablyov, M., Precup, D., and Bengio, Y. Flow network based generative models for non-iterative diverse candidate generation. *Advances in Neural Information Processing Systems*, 34:27381–27394, 2021.

Bengio, Y., Lahlou, S., Deleu, T., Hu, E. J., Tiwari, M., and Bengio, E. Gflownet foundations. *Journal of Machine Learning Research*, 24(210):1–55, 2023.

Blaschke, T., Arús-Pous, J., Chen, H., Margreitter, C., Tyrchan, C., Engkvist, O., Papadopoulos, K., and Patronov, A. Reinvent 2.0: an ai tool for de novo drug design. *Journal of chemical information and modeling*, 60(12): 5918–5922, 2020.

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

Gao, W., Fu, T., Sun, J., and Coley, C. Sample efficiency matters: a benchmark for practical molecular optimization. *Advances in Neural Information Processing Systems*, 35:21342–21357, 2022.

Garnett, R. *Bayesian Optimization*. Cambridge University Press, 2023.

Jensen, J. H. 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.

Kim, H., Kim, M., Choi, S., and Park, J. Genetic-guided gflownets: Advancing in practical molecular optimization benchmark. *arXiv preprint arXiv:2402.05961*, 2024.

Nigam, A., Friederich, P., Krenn, M., and Aspuru-Guzik, A. Augmenting genetic algorithms with deep neural networks for exploring the chemical space. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1lmyRNFvr.

Olivecrona, M., Blaschke, T., Engkvist, O., and Chen, H. Molecular de-novo design through deep reinforcement learning. *Journal of cheminformatics*, 9:1–14, 2017.

Rasmussen, C. E. and Williams, C. K. I. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, USA, January 2006.

Tripp, A. and Hernández-Lobato, J. M. Genetic algorithms are strong baselines for molecule generation. *arXiv preprint arXiv:2310.09267*, 2023.

Wang-Henderson, M., Soyuer, B., Kassraie, P., Krause, A., and Bogunovic, I. Graph neural bayesian optimization for virtual screening. In *NeurIPS 2023 Workshop on Adaptive Experimental Design and Active Learning in the Real World*, 2023.

A. Details of BO setup

Full code for our experiments is available at:

https://github.com/AustinT/basic-mol-bo-workshop2024

Our implementation used:

- An initial set of **10** molecules randomly sampled from the ZINC 250k dataset.
- A BO batch size of **1** (i.e. one molecule is selected every iteration)
- The default GA from the MOLGA package was used as the optimizer. It used a population size of 10^4, an offspring size of 200, and 5 generations.
- To prevent excessively large molecules from being produced, molecules were limited to have at most 100 heavy atoms.
- A UCB acquisition function with random β values, (logarithmically) evenly distributed in $[10^{-2}, 10^0]$.

To reduce computational requirements, we ran the above procedure for 990 iterations, then selected the remaining 9000 allowable molecules randomly. This means that our results are likely an *underestimate* of BO's potential.

B. Full results

See Table 1. The full results (including dis-aggregated AUC values and log files) are available at:

https://github.com/AustinT/basic-mol-bo-workshop2024

Table 1. AUC top-10 scores on PMO benchmark (Gao et al., 2022). *Taken from Gao et al. (2022). **Taken from Tripp & Hernández-Lobato (2023). †Taken from Kim et al. (2024).

Method	REINVENT*	MolGA**	Genetic GFN†	Our GP BO
albuterol_similarity	0.882 ± 0.006	0.896 ± 0.035	0.949 ± 0.010	0.964 ± 0.050
amlodipine_mpo	0.635 ± 0.035	0.688 ± 0.039	0.761 ± 0.019	0.720 ± 0.061
celecoxib_rediscovery	0.713 ± 0.067	0.567 ± 0.083	0.802 ± 0.029	0.860 ± 0.002
deco_hop	0.666 ± 0.044	0.649 ± 0.025	0.733 ± 0.109	0.672 ± 0.118
drd2	0.945 ± 0.007	0.936 ± 0.016	0.974 ± 0.006	0.902 ± 0.117
fexofenadine_mpo	0.784 ± 0.006	0.825 ± 0.019	0.856 ± 0.039	0.806 ± 0.006
gsk3b	0.865 ± 0.043	0.843 ± 0.039	0.881 ± 0.042	0.877 ± 0.055
isomers_c7h8n2o2	0.852 ± 0.036	0.878 ± 0.026	0.969 ± 0.003	0.911 ± 0.031
isomers_c9h10n2o2pf2cl	0.642 ± 0.054	0.865 ± 0.012	0.897 ± 0.007	0.828 ± 0.126
jnk3	0.783 ± 0.023	0.702 ± 0.123	0.764 ± 0.069	0.785 ± 0.072
median1	0.356 ± 0.009	0.257 ± 0.009	0.379 ± 0.010	0.415 ± 0.001
median2	0.276 ± 0.008	0.301 ± 0.021	0.294 ± 0.007	0.408 ± 0.003
mestranol_similarity	0.618 ± 0.048	0.591 ± 0.053	0.708 ± 0.057	0.930 ± 0.106
osimertinib_mpo	0.837 ± 0.009	0.844 ± 0.015	0.860 ± 0.008	0.833 ± 0.011
perindopril_mpo	0.537 ± 0.016	0.547 ± 0.022	0.595 ± 0.014	0.651 ± 0.030
qed	0.941 ± 0.000	0.941 ± 0.001	0.942 ± 0.000	0.947 ± 0.000
ranolazine_mpo	0.760 ± 0.009	0.804 ± 0.011	0.819 ± 0.018	0.810 ± 0.011
scaffold_hop	0.560 ± 0.019	0.527 ± 0.025	0.615 ± 0.100	0.529 ± 0.020
sitagliptin_mpo	0.021 ± 0.003	0.582 ± 0.040	0.634 ± 0.039	0.474 ± 0.085
thiothixene_rediscovery	0.534 ± 0.013	0.519 ± 0.041	0.583 ± 0.034	0.727 ± 0.089
troglitazone_rediscovery	0.441 ± 0.032	0.427 ± 0.031	0.511 ± 0.054	0.756 ± 0.141
valsartan_smarts	0.178 ± 0.358	0.000 ± 0.000	0.135 ± 0.271	0.000 ± 0.000
zaleplon_mpo	0.358 ± 0.062	0.519 ± 0.029	0.552 ± 0.033	0.499 ± 0.025
Sum	14.196	14.708	16.213	16.303

C. SMILES from Figure 4

Top pair:

```
CCCCC
```

```
CCCCCCCCCCCCCCCCCCCC
```

Bottom pair:

```
CC1=CC=C(C=C1)C1=CC(=NN1C1=CC=C(C=C1)S(N)(=O)=O)C(F)(F)F
```

```
Cc1ccc(-c2cc(C(F)(F)F)nn2-c2ccc(-n3nc(C(F)(F)F)cc3-c3ccc(-n4nc(C(F)(F)F)cc4-c4ccc
(S(N)(=O)=O)cc4)cc3)cc2)cc1
```