MolPAL: A Software Tutorial for Sample Efficient High-Throughput Virtual Screening

David E. Graff
Department of Chemistry and Chemical Biology
Harvard University
Cambridge, MA 02138
&
Department of Chemical Engineering
MIT, Cambridge, MA 02142
deg711@g.harvard.edu

Connor W. Coley
Department of Chemical Engineering
Department of Electrical Engineering and Computer Science
MIT, Cambridge, MA 02139
ccoley@mit.edu

Abstract

Structure-based virtual screening (SBVS) of ultra-large chemical libraries has led to the discovery of novel inhibitors for challenging protein targets. However, screening campaigns of these magnitudes are expensive and thus impractical to employ in standard practice. As the broad goal of most SBVS workflows is the identification of the most potent molecules in the library, the task can be viewed as an optimization problem. Previous work has demonstrated the ability for Bayesian optimization to improve sample efficiency in SBVS using the MolPAL software. In this tutorial, we provide a broad algorithmic overview of the MolPAL software and a guide for its utilization in a prospective virtual screening task.

1 Introduction

Structure-based virtual screening (SBVS) is an important tool in drug discovery programs. Structure-based simulations, such as computational docking, enable chemists to quickly search large compound libraries for potential binders to a given protein. These simulations mimic the physical process of drug binding and estimate the binding potency with a "docking score". Early applications of so-called "high-throughput docking" screened libraries containing hundreds of thousands molecules [4]. Given the recent explosion in the sizes of publicly available enumerated compound libraries, there is renewed interest in high-throughput docking for drug discovery. Recent examples include the screening of more than 100M molecules to identify nano- and picomolar antibacterial and anti-psychotic compounds, respectively [18], and the screening of 1.4B molecules to discover a sub-micromolar anti-inflammatory compound [10].

Ultra-large SBVS is expensive to employ as its cost scales proportionally with the size of the library. The large resource demands of these ultra-large screens also make them practically challenging, requiring computational resources that are only accessible on select high-performance computing clusters. These docking campaigns screen a massive number of molecules, but the end goal is generally the identification of the top-k ($k \leq 100\text{k}$) molecules in the library by docking score. This view of SBVS allows it to be framed as an optimization problem, where the goal is to identify a subset of molecules $\{x_i\}_{i=1}^{k}$ in a library $\mathcal{X}$ that minimizes the summed docking scores, i.e.,

AI for Accelerated Materials Design, NeurIPS 2022 Workshop.

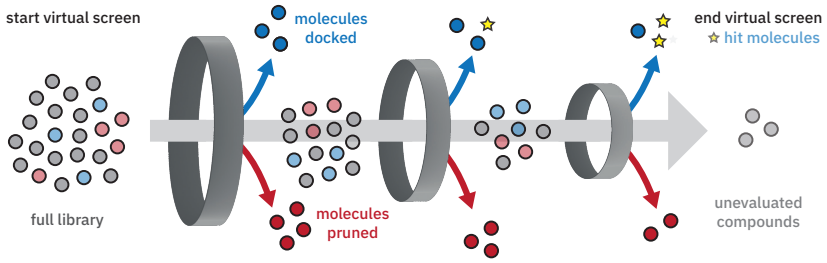

Figure 1: Graphical depiction of the MolPAL workflow. A virtual library (circles) will be subjected to multiple iterations of the optimization loop (grey ring), where molecules are selected for docking (blue circles) and, optionally, pruned (§3.1; red circles). Upon termination of the optimization loop, some number of molecules will remain undocked (grey circles).

$\{x_i\}_{i=1}^k = \text{argmin}_{\{x_i\}_{i=1}^k \subseteq \mathcal{X}} \sum_{i=1}^k f(x_i)$, where $f(\cdot)$ computes the docking score. We leveraged discrete optimization techniques in the development of the MolPAL software, which improves the sample efficiency of SBVS via model-guided optimization. This tutorial is intended to provide a general summary of how to employ MolPAL in a virtual screening scenario, starting from library preparation and setting up the distribution engine to choosing optimization parameters and analyzing the results.

2 Software and Methods

MolPAL ("molecular pool-based active learning") is a software that accelerates SBVS using Bayesian optimization (BO) (Figure 1). Broadly, MolPAL accepts three main inputs: (1) a virtual library, (2) a docking protocol, and (3) optimization hyperparameters, such as the surrogate machine learning (ML) model architecture and acquisition function. As the MolPAL program runs, it will use the input optimization hyperparameters (described in further detail below) to adaptively select a batch of molecules from the library to dock. Upon completion, MolPAL outputs a CSV file containing the SMILES string of each compound docked and the docking score of its best pose. The software is open source and available under the MIT license at [11]. It is invoked from the command line like so:

```
$ molpal run --library LIBRARY -o docking \
    --objective-config OBJECTIVE_CONFIG [optimization hyperparameters]
```

At a high level, MolPAL operates via (1) docking a random batch of n molecules from the library, (2) training a surrogate machine learning model $\hat{f}$ on these docking data, (3) predicting the docking score and (optionally) the uncertainty for all remaining molecules in the library (4) calculating the "acquisition utility" of all undocked molecules using these predictions and an acquisition function α, (5) docking the top-n points by acquisition utility, and (6) repeating steps (2)–(5) for some number of iterations T. For a more detailed review of Bayesian optimization we refer a reader to [25] and [8]. MolPAL pipelines the entire optimization from model training and prediction to running the docking simulations and parsing the results. It requires no input from a user once the program starts. In addition, MolPAL is able to use distributed resource allocations to parallelize each of these steps across large resource allocations.

We have previously studied the ability of MolPAL to accelerate a variety of different SBVS tasks across various virtual chemical libraries, docking software, and protein targets, and we refer a reader to [14] for the complete results. In these studies, we found an improvement in sample efficiency between 40- and 80-fold over random selection on an ultra-large virtual screening task of 100M molecules docked against AmpC with DOCK3.7 from Lyu et al. [18] (Figure 2). While MolPAL is also effective in settings with smaller virtual libraries, the cost savings are both more pronounced and more impactful as the size of the library increases.

3 General workflow

A typical MolPAL run is broken up into five steps: (1) connecting compute resources with a Ray cluster, (2) preparing the virtual library of candidate ligands, (3) (optional) pre-computing

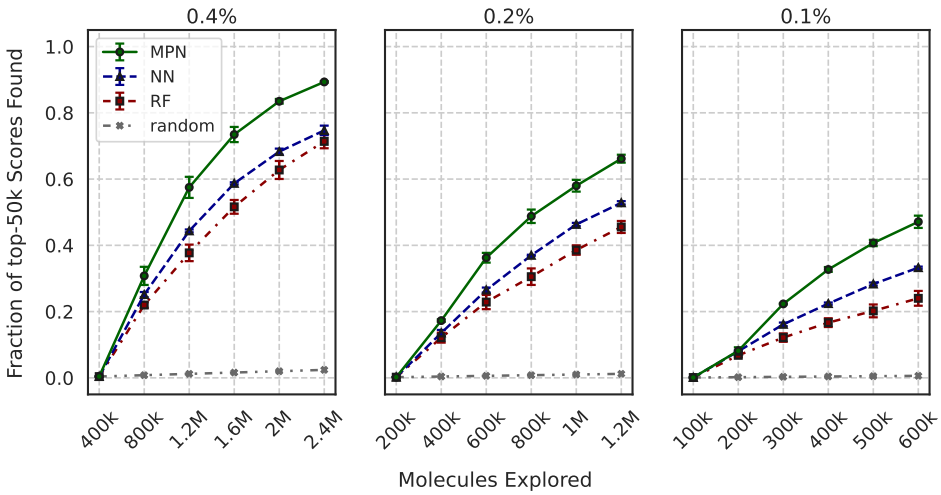

Figure 2: MolPAL performance on the AmpC library (100M) with greedy acquisition and the specified surrogate model architecture. Plot title represents the fraction of the pool acquired in each batch. *RF*: random forest; *NN*: feedforward neural network; *MPN*: directed message passing neural network; *random*: random acquisition from the library. Figure reproduced using data from [14].

featurizations of the library molecules, (4) defining parameters of the docking and optimization protocols, (5) running the MolPAL software, and (6) analyzing the results. In the following paragraphs, we will describe each of these steps in detail, and for an interactive version, a reader may utilize the tutorial notebook [11].

1. Utilizing distributed resources MolPAL can utilize large and distributed hardware allocations to parallelize each step in the optimization loop (e.g., model training, inference, and docking) using the Ray [19] distributed computing framework. Before running any MolPAL commands or scripts below, a Ray cluster must be started: `ray start --head`. The previous command is sufficient for utilizing local resources (i.e., a single node), but for more involved settings, such as starting a Ray cluster over multiple nodes, the details will depend on a user's specific system. The MolPAL repository and supplemental notebook [11] both include an example using Slurm, and the Ray documentation includes examples for starting clusters on other HPC schedulers or cloud-based platforms like GCP. A typical use-case for MolPAL is to search a library of 50-100M molecules and docking approximately 2M of them. These runs typically finish within 48 hours across 400-600 CPUs. Because the vast majority of costs in a MolPAL run are docking-related, we generally find GPUs unnecessary to accelerate surrogate model training and inference in the case of NN and MPN models.

2. Preparing the virtual library A key input to a MolPAL run is the virtual chemical library to explore in the form of a text file containing the SMILES string of each compound. We recommend that the library contain no duplicated molecules to avoid one batch containing the same molecule multiple times. Note also that MolPAL accepts a library as-is and performs no further modifications. It is possible that some molecules in a library possess multiple states corresponding to the same SMILES string (e.g., tautomeric, ring-conformation, ionization states, etc.), some of which dock more favorably than others. Enumeration of these states with tools such as RDKit [1] or Gypsum-DL [23] during library preparation can avoid missing these molecules. Some ultra-large libraries, such as the Enamine REAL Database [7], are provided in several shards. Provided that each shard has the same format, simply pass each one to the `--library` argument to mimic searching the full, unsharded library.

3. Featurizing the library In cases where the surrogate model expects vector inputs, such as random forest (RF) or feedforward neural network models (NN), the vector representations of library members can be precalculated to save time during the actual run. To do so, use the `fingerprints.py` script to generate an HDF5 file containing these vectors:

```
$ python fingerprints.py -o OUTPUT -l LIBRARY \
    --fingerprint FINGERPRINT --radius RADIUS --length LENGTH
```

Note that the fingerprint, radius, and length values must match those supplied to MolPAL. Typical values are `pair`, 2, and 2048, respectively, and these produce an output file that is approximately 200G per 100M SMILES strings. Next, supply this HDF5 file, `OUTPUT`, to MolPAL via the `--fps` argument. This script requires a Ray cluster to be started before running, so please refer to the 1. Utilizing distributed resources paragraph above for details on setting one up. This script also outputs the indices of any invalid SMILES strings in the provided library. Providing these indices to MolPAL via the `--invalid-idxs` argument will skip the library validation step during startup.

4. Defining the the run configuration MolPAL has two broad sets of parameters: (1) optimization hyperparameters: surrogate model architecture (D-MPNN, RF, NN), molecular representation (graph, fingerprint), acquisition function (greedy, UCB: upper confidence bound; TS: Thompson sampling; PI: probability of improvement; and EI: expected improvement), batch size (expressed as either a fraction of the pool or absolute amount), and stopping criteria (maximum number of iterations, total sample count, minimum improvement); and (2) docking task parameters (chemical library, protein target, and docking parameters). Given that each optimization task is unique, it is impossible to definitively identify the optimal set of optimization hyperparameters for every scenario. In previous studies, we experimented broadly among these choices for several tasks and found that the combination of a directed message passing neural network (D-MPNN) with mean-variance estimation (MVE) uncertainty quantification [21] for the surrogate model combined with an upper confidence bound (UCB) acquisition function to be robust in terms of optimization performance (e.g., `--model=mpn --conf-method=mve --metric=ucb`). We also found that smaller batch sizes led to generally higher enrichment compared to larger batch sizes for a similar sample count, but this shifts the computational budget to spend more time on training and prediction costs as the surrogate model is updated more frequently. It is an ongoing challenge in the field to identify the optimal batch size in a principled manner. Early stopping experiments showed that most optimizations converged between 8-10 batches with a 0.2% batch size on a 2M member library, so we generally recommend deciding on a total sample count first and then determining the batch size such that 6-10 batches are selected. For convenience, all of these arguments may be stored in a YAML-format configuration file and supplied via the `--config` argument.

MolPAL uses the pyscreener library [13] to perform the docking calculations, so a pyscreener configuration file is passed to the `--objective-config` argument. This file requires only the filepath of a receptor in PDB format, the desired docking software (Vina [26], Smina [17], PSOVina [20], QVina2 [2], and DOCK6 [3]), and docking box parameters. For more details on this file, we refer a reader to the documentation [12], and for a general guide on determining the appropriate docking protocol, we refer a reader to [5].

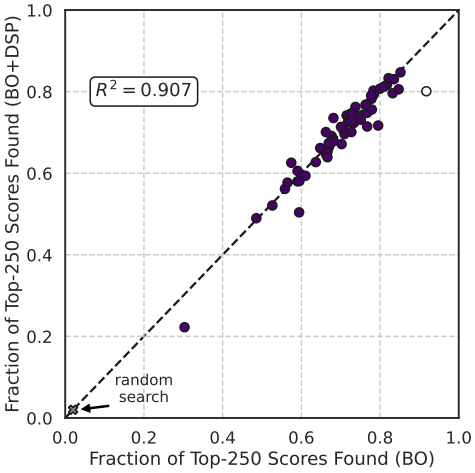

Figure 3: Parity plot of final MolPAL performance without DSP ("BO", x-axis) vs. with DSP ("BO+DSP", y-axis) on each DOCKSTRING task. Each point represents the mean performance of five independent runs. Runs were conducted with a D-MPNN surrogate model, UCB acquisition and six batches (one initialization plus five acquisition) of ca. 1,000 molecules. White circle indicates $p < 0.05$ (Bonferroni-corrected) that BO+DSP performance is worse than BO. Figure reproduced using data from [15].

6. Analyzing the results The outputs of a completed MolPAL run are are (1) a CSV containing all of the molecules and the docking score of their top-ranked pose and (2) a series of zipped tarballs containing all input and output files generated from docking. Typically, pose analysis is necessary before selecting molecules for further simulation or experimentation. To access the docking poses of the top-k molecules, run `molpal extract OUTPUT_DIR K` to extract the poses of the best `K` molecules from their respective tarballs to the `poses` subdirectory. Poses will be stored in the native output format of the selected docking software (e.g., PDBQT for Vina-type software and SDF for DOCK6). Once extracted, the poses may then

be examined either manually using programs such as PyMol [24] and Chimera [22] or automatically using Python libraries such as ProLIF [6].

3.1 Limiting overhead costs with pruning

In the regime of ultra-large chemical libraries, the surrogate model training and inference steps (steps (2) and (3) from Software and Methods) can become quite costly. When using a D-MPNN model on a library of 100M molecules with a batch size of 400k, these two steps can comprise more than 15% of the cost of a single iteration of the optimization loop. To mitigate these overhead costs, we developed design space pruning (DSP) as an extension to the framework of model-guided optimization [15]. At each iteration of the optimization, DSP *removes* molecules from the library based on the probability that a given molecule is a hit (as calculated by the surrogate model). This reduces the number of inference calls made in the prediction step of the next iteration and thus its overall cost. Across 58 different docking tasks of 260k molecules from the DOCKSTRING benchmark [9], BO+DSP resulted in comparable performance to baseline BO with significantly reduced inference costs (Figure 3).

To use DSP during a MolPAL run, all that is required is the `--prune` flag. Because DSP introduces the possibility removing "good" molecules from the library before they are selected for docking, it is best suited for situations where the reduced inference costs outweigh the risk of removing active molecules from the library.

4 Conclusion

Model-guided optimization is an attractive choice for SBVS when the goal is to identify a broadly potent set of molecules. Exhaustive search of ultra-large chemical libraries with SBVS is expensive, but the optimization approach of MolPAL can drastically lower these costs while stile identifying the large majority of potent molecules. The approach is not limited to SBVS and is effective in other virtual screening objectives, such as the identification of organic photoelectric materials, as shown in a retrospective experiment on the Harvard Clean Energy Project dataset [14, 16].

References

[1] RDKit. URL `http://rdkit.org/`.

[2] A. Alhossary, S. D. Handoko, Y. Mu, and C.-K. Kwoh. Fast, accurate, and reliable molecular docking with QuickVina 2. *Bioinformatics*, 31(13):2214–2216, July 2015. doi: 10.1093/bioinformatics/btv082.

[3] W. J. Allen, T. E. Balius, S. Mukherjee, S. R. Brozell, D. T. Moustakas, P. T. Lang, D. A. Case, I. D. Kuntz, and R. C. Rizzo. DOCK 6: Impact of new features and current docking performance. *Journal of Computational Chemistry*, 36(15):1132–1156, 2015. doi: 10.1002/jcc.23905.

[4] J. C. Alvarez. High-throughput docking as a source of novel drug leads. *Current Opinion in Chemical Biology*, 8(4):365–370, Aug. 2004. doi: 10.1016/j.cbpa.2004.05.001.

[5] B. J. Bender, S. Gahbauer, A. Luttens, J. Lyu, C. M. Webb, R. M. Stein, E. A. Fink, T. E. Balius, J. Carlsson, J. J. Irwin, and B. K. Shoichet. A practical guide to large-scale docking. *Nature Protocols*, 16 (10):4799–4832, Oct. 2021. ISSN 1750-2799.

[6] C. Bouysset and S. Fiorucci. ProLIF: a library to encode molecular interactions as fingerprints. *Journal of Cheminformatics*, 13(1):72, Sept. 2021. ISSN 1758-2946.

[7] Enamine, LTD. REAL Database, 2022. URL `https://enamine.net/compound-collections/real-compounds/real-database`.

[8] P. I. Frazier. A Tutorial on Bayesian Optimization. *arXiv:1807.02811 [cs, math, stat]*, July 2018.

[9] M. García-Ortegón, G. N. C. Simm, A. J. Tripp, J. M. Hernández-Lobato, A. Bender, and S. Bacallado. DOCKSTRING: easy molecular docking yields better benchmarks for ligand design. *arXiv:2110.15486 [cs, q-bio, stat]*, Oct. 2021.

[10] C. Gorgulla, A. Boeszoermenyi, Z.-F. Wang, P. D. Fischer, P. W. Coote, K. M. Padmanabha Das, Y. S. Malets, D. S. Radchenko, Y. S. Moroz, D. A. Scott, K. Fackeldey, M. Hoffmann, I. Iavniuk, G. Wagner, and H. Arthanari. An open-source drug discovery platform enables ultra-large virtual screens. *Nature*, 580 (7805):663–668, Apr. 2020. doi: 10.1038/s41586-020-2117-z.

[11] D. Graff and C. Coley. MolPAL: Molecular Pool-based Active Learning, Sept. 2022. URL https://github.com/coleygroup/molpal. original-date: 2020-08-11T14:03:00Z.

[12] D. E. Graff. pyscreener, 2022. URL https://github.com/coleygroup/pyscreener.

[13] D. E. Graff and C. W. Coley. pyscreener: A Python Wrapper for Computational Docking Software. *Journal of Open Source Software*, 7(71):3950, Mar. 2022. ISSN 2475-9066. doi: 10.21105/joss.03950.

[14] D. E. Graff, E. I. Shakhnovich, and C. W. Coley. Accelerating high-throughput virtual screening through molecular pool-based active learning. *Chem. Sci.*, 12(22):7866–7881, June 2021. doi: 10.1039/D0SC06805E.

[15] D. E. Graff, M. Aldeghi, J. A. Morrone, K. E. Jordan, E. O. Pyzer-Knapp, and C. W. Coley. Self-Focusing Virtual Screening with Active Design Space Pruning. *J. Chem. Inf. Model.*, 62(16):3854–3862, Aug. 2022. doi: 10.1021/acs.jcim.2c00554.

[16] J. Hachmann, R. Olivares-Amaya, S. Atahan-Evrenk, C. Amador-Bedolla, R. S. Sánchez-Carrera, A. Gold-Parker, L. Vogt, A. M. Brockway, and A. Aspuru-Guzik. The Harvard Clean Energy Project: Large-Scale Computational Screening and Design of Organic Photovoltaics on the World Community Grid. *The Journal of Physical Chemistry Letters*, 2(17):2241–2251, Sept. 2011.

[17] D. R. Koes, M. P. Baumgartner, and C. J. Camacho. Lessons Learned in Empirical Scoring with smina from the CSAR 2011 Benchmarking Exercise. *J. Chem. Inf. Model.*, 53(8):1893–1904, Aug. 2013. doi: 10.1021/ci300604z.

[18] J. Lyu, S. Wang, T. E. Balius, I. Singh, A. Levit, Y. S. Moroz, M. J. O'Meara, T. Che, E. Algaa, K. Tolmachova, A. A. Tolmachev, B. K. Shoichet, B. L. Roth, and J. J. Irwin. Ultra-large library docking for discovering new chemotypes. *Nature*, 566(7743):224–229, Feb. 2019. doi: 10.1038/s41586-019-0917-9.

[19] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica. Ray: A Distributed Framework for Emerging AI Applications. *arXiv:1712.05889 [cs, stat]*, Sept. 2018.

[20] M. C. K. Ng, S. Fong, and S. W. I. Siu. PSOVina: The hybrid particle swarm optimization algorithm for protein-ligand docking. *Journal of Bioinformatics and Computational Biology*, 13(3):1541007, June 2015. ISSN 1757-6334. doi: 10.1142/S0219720015410073.

[21] D. A. Nix and A. S. Weigend. Estimating the mean and variance of the target probability distribution. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 1, pages 55–60 vol.1, June 1994. doi: 10.1109/ICNN.1994.374138.

[22] E. F. Pettersen, T. D. Goddard, C. C. Huang, G. S. Couch, D. M. Greenblatt, E. C. Meng, and T. E. Ferrin. UCSF Chimera–a visualization system for exploratory research and analysis. *Journal of Computational Chemistry*, 25(13):1605–1612, Oct. 2004.

[23] P. J. Ropp, J. O. Spiegel, J. L. Walker, H. Green, G. A. Morales, K. A. Milliken, J. J. Ringe, and J. D. Durrant. Gypsum-DL: an open-source program for preparing small-molecule libraries for structure-based virtual screening. *Journal of Cheminformatics*, 11(1):34, May 2019. ISSN 1758-2946. doi: 10.1186/s13321-019-0358-3. URL https://doi.org/10.1186/s13321-019-0358-3.

[24] Schrödinger, LLC. The PyMOL Molecular Graphics System, Version 1.8. Nov. 2015.

[25] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proc. IEEE*, 104(1):148–175, Jan. 2016. doi: 10.1109/JPROC.2015.2494218.

[26] O. Trott and A. J. Olson. AutoDock Vina: Improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. *Journal of Computational Chemistry*, 31(2):455–461, 2010. doi: 10.1002/jcc.21334.