Bencher: Simple and Reproducible Benchmarking for Black-Box Optimization

Leonard Papenmeier [1] **Luigi Nardi** [1 2]

Abstract

We present Bencher, a modular benchmarking framework for black-box optimization that fundamentally decouples benchmark execution from optimization logic. Unlike prior suites that focus on combining many benchmarks in a single project, Bencher introduces a clean abstraction boundary: each benchmark is isolated in its own virtual Python environment and accessed via a unified, version-agnostic RPC interface. This design eliminates dependency conflicts and simplifies the integration of diverse, real-world benchmarks, which often have complex and conflicting software requirements. Bencher can be deployed locally or remotely via Docker or on HPC clusters via Singularity, providing a containerized, reproducible runtime for any benchmark. Its lightweight client requires minimal setup and supports drop-in evaluation of 80 benchmarks across continuous, categorical, and binary domains.

1. Introduction

Black-box optimization refers to the problem of optimizing a function

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

of unknown form for which we can only observe the function value but no derivatives (Turner et al., 2021). In particular, f may be highly multimodal and/or noisy. Furthermore, f often is expensive to evaluate, and we can only afford to spend a limited number of function evaluations to find a good solution.

Problems of this type have received considerably interest due to their ubiquity in various field like chemical engineering (Hernández-Lobato et al., 2017; Burger et al., 2020), materials science (Frazier & Wang, 2016; Hughes et al., 2021), engineering (Lam et al., 2018; Maathuis et al., 2024), hyperparameter optimization (Snoek et al., 2012; Bergstra et al., 2011), neural architecture search (Ru et al., 2021), hardware design (Nardi et al., 2019; Ejjeh et al., 2022), drug discovery (Negoescu et al., 2011; Svensson et al., 2022), and life sciences (Tallorin et al., 2018; Cosenza et al., 2022).

The development of sample-efficient algorithms for black-box optimization is a highly active research field. To give a holistic picture of an algorithm's performance, it is usually evaluated on a wide range of benchmarks. Besides synthetic benchmarks of known form, methods are usually evaluated on various benchmarks reflecting real-world applications. We call these benchmarks *real-world benchmarks*. Running these benchmarks often requires significant effort for two reasons. First, many benchmarks have very specific software requirements and can have a complex setup, making it hard to set them up even in isolation. For example, the Mujoco benchmarks used in Wang et al. (2020) require several scientific libraries to be installed, setting environment variables, and the presence of the Mujoco executable in a specific location. Second, running multiple benchmarks in a single project can be time-consuming, as different benchmarks can have different, conflicting requirements. For instance, the benchmarks used by Wang et al. (2018) require Python version 3.8, which has reached end-of-life in 2024, conflicting with newer Python versions.

These difficulties in setting up benchmarks can lead to cases where results on the same benchmark are not comparable, arguably due to inconsistent setups. For example, the TuRBO baseline (Eriksson et al., 2019) in Fan et al. (2024, Fig. 3, Hopper benchmark) lies on a completely different scale than the same baseline on the same benchmark in Nguyen et al. (2022, Fig. 3, Hopper benchmark).

This paper introduces Bencher[1], a benchmarking framework that allows running benchmarks in a reproducible and simple way. Bencher isolates every benchmark (or set of compatible benchmarks) in a virtual Python environment. This allows each benchmark to run with different versions of Python and other external packages. To facilitate the setup of a benchmark, Bencher can run in a Docker container. We provide a simple package with minimal external dependencies that is responsible for the communication with

[*]Equal contribution [1]Department of Computer Science, Lund University, Sweden [2]DBtune. Correspondence to: Leonard Papenmeier <leonard.papenmeier@cs.lth.se>.

Proceedings of the ICML 2025 Workshop on Championing Open-source Development in Machine Learning (CODEML '25). Copyright 2025 by the author(s).

[1]https://github.com/LeoIV/bencher

the Bencher Docker container. Furthermore, we provide a Singularity container that can be used to run Bencher on a cluster and provide detailed instructions for doing so.

In summary, we make the following contributions

- We introduce Bencher, a benchmarking framework that ensures reproducibility and simplicity by isolating each benchmark in a dedicated virtual Python environment.
- We design a server-based architecture that handles remote procedure calls (RPCs) and enables flexible, version-independent execution of benchmarks.
- We provide containerized solutions (Docker and Singularity) along with lightweight client packages to support easy deployment on both local machines and high-performance computing (HPC) clusters.

2. Related Work

Several benchmark suites for non-convex black-box optimization have been proposed in the literature. A recent survey by Sala & Müller (2020) provides a comprehensive overview of the state of the art in black-box optimization benchmarks.

Nevergrad (Bennet et al., 2021) is an open-source platform for black-box optimization that offers a wide portfolio of algorithms and benchmark problems, including synthetic, combinatorial, and real-world tasks. It features automatic algorithm selection, extensive parallelism, and a public leaderboard, but does not support environment isolation, limiting the simultaneous use of conflicting benchmarks. Many of the benchmarks in Nevergrad are undocumented, as acknowledged by the authors[2]. For this reason, Nevergrad is currently not included in Bencher but we aim to establish a collaboration with the authors to add the benchmarks in the future.

COCO (Hansen et al., 2019) is a long-standing and widely used benchmarking platform for zero-order black-box optimization, including noisy, multi-objective, and non-continuous problems. The single-objective, continuous, and noise-free benchmarks from the BBOB suite are especially popular and have been used in many studies. These methods are included in Bencher, using the implementation provided by de Nobel et al. (2024).

IOHexperimenter de Nobel et al. (2024) is the experimentation module of IOHprofiler Doerr et al. (2018), offering a wide range of continuous and pseudo-boolean benchmarks. Bencher implements most of the benchmarks from IOHexperimenter, including the BBOB, pseudo-boolean, W-model, and submodular benchmarks.

[2]See https://facebookresearch.github.io/nevergrad/benchmarks.html#list-of-benchmarks

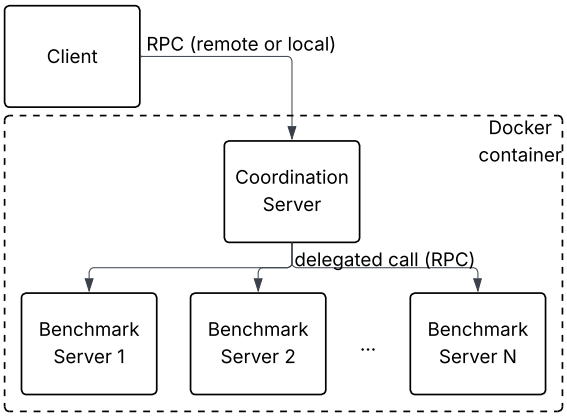

Figure 1. The Bencher architecture. The server runs in a Docker container and listens to RPCs from clients, which can be on the same or on a different machine. The server is composed of multiple Poetry environments, one for each benchmark.

While all of the aforementioned benchmark suites advance the state of the art in black-box optimization, they differ from Bencher in that they do not aim at decoupling the benchmarks but instead focus on providing a comprehensive set of benchmarks in a single package.

3. Bencher: Design and Implementation

Bencher follows a client-server architecture (see Figure 1). The server is responsible for running the benchmarks and listening to remote procedure calls (RPCs) from clients. It can be run in a Docker container, which allows running the server in an isolated environment with minimal setup. Furthermore, a Singularity container can inherit the Docker container to run Bencher on an HPC cluster.

The server is responsible for running the benchmarks and listening to remote procedure calls (RPCs) from clients. It is composed of multiple Poetry environments – one coordinator and multiple environments for sets of compatible benchmarks. The coordinator is responsible for listening to RPCs and delegating them to the appropriate benchmark environment. The client and the server, and the different benchmark environments among themselves, communicate via gRPC[3], a high-performance, open-source universal RPC framework. Each benchmark environment exposes a gRPC service on a specific port that listens for incoming RPCs. The clients exclusively communicate with the coordinator, which forwards the RPCs to the appropriate benchmark environment. This structure has the advantage that the Docker container only needs to expose a single port and that the clients only need to speak to the coordinator.

[3]https://grpc.io/

3.1. Client

The client is the interface to the server and the main entry point for users. It establishes a connection to the coordination server (see Figure 1) and provides methods to evaluate points on the benchmarks. The client is implemented in Python and is available on PyPI (https://pypi.org/project/bencherscaffold/). It is designed to have minimal external dependencies (only grpcio and protobuf are required) and can be installed by pip install bencherscaffold. Once the client successfully connects to the server, it can be used to evaluate points on the benchmarks as follows:

Listing 1. Exemplary client code for evaluating a benchmark (long lines are broken).

```
from bencherscaffold.client import ↙
    BencherClient
from bencherscaffold.protoclasses.↙
    bencher_pb2 import Value, ValueType
client = BencherClient()
benchmark_name = 'mopta08'
values = [Value(type=ValueType.CONTINUOUS,↙
     value=0.5) for _ in range(124)]
result: float = client.evaluate_point(↙
    benchmark_name, point)
```

This code runs the 124-dimensional Mopta08 vehicle mass optimization benchmark (Jones, 2008) in its soft-constrained version (Eriksson et al., 2019). All 124 parameters of this benchmark are continuous and normalized to the unit hypercube $[0, 1]^{124}$. An exemplary implementation running all available benchmarks is available on GitHub (https://github.com/LeoIV/bencherclient). This repository is also used during the testing of Bencher to ensure that all benchmarks are working correctly.

3.2. Docker Container

The server can be run in a Docker container, which allows running the server in an isolated environment with minimal setup. The Docker container can be built from source by cloning the repository and running docker build in the root directory of the repository. It is also available on Docker Hub (https://hub.docker.com/r/gaunab/bencher). Since all communication between the different benchmark environments happens internally, the Docker container only needs to expose a single port to the outside world. The container can be pulled and run in the background with the following command:

```
docker pull gaunab/bencher
docker run -d --name bencher -p 50051:50051
    ↪  gaunab/bencher
```

3.3. HPC Setup

Bencher can be run on an HPC cluster using Singularity with minimal manual setup by inheriting the Docker container. One constraint is that the Singularity container needs to be started as an instance in the background. To isolate instances running on the same cluster node, we recommend using a unique instance name for each HPC job. With slurm, this can be done by using the job ID as the instance name, e.g.,

```
singularity instance start 'INST_NAME'
```

where 'INST_NAME' is replaced with ${SLURM_ARRAY_JOB_ID}_${SLURM_ARRAY_TASK_ID}. A job can then be run with

```
singularity run instance://INST_NAME CMD
```

```
bencher
├── BencherServer
│   ├── bencherserver
│   │   ├── __init__.py
│   │   └── benchmark-registry.json
│   └── pyproject.toml
├── Benchmark 1
│   ├── benchmark1
│   │   └── __init__.py
│   ├── .python-version
│   └── pyproject.toml
├── ...
├── Benchmark N
│   ├── benchmarkN
│   │   └── __init__.py
│   └── pyproject.toml
├── Dockerfile
├── container.sdef
└── entrypoint.py
```

Figure 2. The Bencher directory structure.

3.4. Project Structure

The Bencher implementation follows a prespecified structure that allows for easy extendability. The directory structure is shown in Figure 2. All benchmarks, the server, and other relevant files are located in the bencher directory. We refer to each of these subdirectories as *subprojects*. Each subproject has its own pyproject.toml file, which defines the dependencies for the subproject and a starting script start-benchmark-service. The entrypoint.py, which is the entry point for the Docker container, goes into each subproject, activates the virtual environment, and starts the server using this script.

The benchmark-registry.json file is used to define the mapping between the benchmark names and the ports on which the benchmarks are running:

```
... "lasso-dna": {
    "port": 50053,
    "dimensions": 180,
```

```
    "type": "purely_continuous"
}, ...
```

where `lasso-dna` is the name of the 180-dimensional `Lasso-DNA` benchmark (Šehić et al., 2022b), `port` is the port on which the benchmark is running, `dimensions` is the number of dimensions of the benchmark, and `type` is the type of the benchmark. For instance, `purely_continuous` means that all dimensions of the benchmark are continuous and `binary` means that all dimensions of the benchmark are binary.

Some subprojects may have the `.python-version` file. This file is used during the Docker installation of the project to set the correct `Python` version for the subprojects. Different Python versions are managed by `pyenv`[4] and are installed in the Docker container. Subprojects without this file will use the default `Python` version of the Docker container.

When using Docker, the `entrypoint.py` file is executed when the container is started. When using Singularity, the `entrypoint.py` file *must* be defined as the startscript container definition file:

```
%startscript
bash -c "python3.11␣/entrypoint.py"
```

An exemplary container definition file is provided in the repository.

New benchmarks can be added by creating a new subproject with the same structure as the existing ones. In particular, the project must define a `pyproject.toml` file, a `start-benchmark-service` script, and, optionally, a `.python-version` file.

3.5. Testing

`Bencher` is tested using integration testing. Triggered by a commit or a daily cron job, the full Docker container is built and started in the background. Then, the `bencherclient` (see Section 3.1) is used to run all benchmarks. If all benchmarks are running successfully, the Docker container is pushed to Docker Hub, and the Singularity container is built, inheriting from the previously pushed Docker image.

4. Benchmarks

`Bencher` currently supports 80 benchmarks, including 18 real-world benchmarks. We list all benchmarks in Appendix A and restrict this section to a general overview. `Bencher` currently is limited to unconstrained, single-objective optimization problems. Benchmarks can be continuous, ordinal, binary, or categorical. All continuous benchmarks are normalized to the unit hypercube $[0, 1]^d$. Categorical benchmarks, such as `pestcontrol`, expect an integer input for each dimension; the number of categories is documented in the `README.md` file in the repository.

Most of the benchmarks implemented in `Bencher` are well-known benchmarks from the literature. For instance, the soft-constrained version of the `Mopta08` benchmark (Jones, 2008) was originally introduced in Eriksson et al. (2019) but has found widespread adoption in the high-dimensional Bayesian optimization literature (Shen & Kingsford, 2021; Eriksson & Jankowiak, 2021; Papenmeier et al., 2022; 2023; Hellsten et al., 2023; Hvarfner et al., 2024; Xu et al., 2024; Papenmeier et al., 2025). However, the original link to the executables of the benchmark is dead, and they are currently only available since other researchers uploaded them, as acknowledged by Xu et al. (2024). Similarly, the Mujoco benchmarks used in various papers, including Wang et al. (2020); Papenmeier et al. (2022); Hvarfner et al. (2024), require installing additional software, setting environment variables, and downloading additional executables, potentially reducing adoption and reproducibility.

5. Conclusion and Future Work

We present `Bencher`, a benchmarking framework for black-box optimization that allows running benchmarks in a reproducible and simple way. It mostly decouples the benchmarking code from the optimization algorithm, allowing for more freedom in the dependency specification of the optimizer's code. `Bencher`'s client only requires minimal dependencies to communicate with the server and is easily installable via `pip`.

The server can be run in a Docker container and abstracts away the complexity of setting up the benchmarks for the user. The benchmarks are isolated in their own virtual environments, allowing for different versions of `Python` and other external packages. `Bencher` further provides a Singularity container that can be used to run `Bencher` on a cluster and provides detailed instructions for doing so.

We plan to extend `Bencher` in several ways. Currently, `Bencher` supports unconstrained, single-objective optimization problems. While this covers a wide range of applications, we aim for a more general framework that can also handle constrained, multi-fidelity, and multi-objective optimization problems. In the future, we will also support more diverse search domains, covering, for instance, graph-based benchmarks. Furthermore, we plan to gradually extend the set of benchmarks in `Bencher`, adding benchmark suites like the one in Nevergrad (Bennet et al., 2021), `CATBench` (Tørring et al., 2024), and `HPOBench` (Eggensperger et al., 2021) suite.

[4] https://github.com/pyenv/pyenv

References

Bennet, P., Doerr, C., Moreau, A., Rapin, J., Teytaud, F., and Teytaud, O. Nevergrad: black-box optimization platform. *ACM SIGEVOlution*, 14(1):8–15, 2021.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 24. Curran Associates, Inc., 2011.

Burger, B., Maffettone, P. M., Gusev, V. V., Aitchison, C. M., Bai, Y., Wang, X., Li, X., Alston, B. M., Li, B., Clowes, R., et al. A mobile robotic chemist. *Nature*, 583(7815): 237–241, 2020.

Cosenza, Z., Astudillo, R., Frazier, P., Baar, K., and Block, D. E. Multi-Information Source Bayesian Optimization of Culture Media for Cellular Agriculture. *Biotechnology and Bioengineering*, 2022.

de Nobel, J., Ye, F., Vermetten, D., Wang, H., Doerr, C., and Bäck, T. Iohexperimenter: Benchmarking platform for iterative optimization heuristics. *Evolutionary Computation*, 32(3):205–210, 2024.

Deshwal, A., Ament, S., Balandat, M., Bakshy, E., Doppa, J. R., and Eriksson, D. Bayesian Optimization over High-Dimensional Combinatorial Spaces via Dictionary-based Embeddings. In *International Conference on Artificial Intelligence and Statistics*, pp. 7021–7039. PMLR, 2023.

Doerr, C., Wang, H., Ye, F., Van Rijn, S., and Bäck, T. Iohprofiler: A benchmarking and profiling tool for iterative optimization heuristics. *arXiv preprint arXiv:1810.05281*, 2018.

Eggensperger, K., Müller, P., Mallik, N., Feurer, M., Sass, R., Klein, A., Awad, N., Lindauer, M., and Hutter, F. Hpobench: A collection of reproducible multi-fidelity benchmark problems for hpo. *arXiv preprint arXiv:2109.06716*, 2021.

Ejjeh, A., Medvinsky, L., Councilman, A., Nehra, H., Sharma, S., Adve, V., Nardi, L., Nurvitadhi, E., and Rutenbar, R. A. HPVM2FPGA: Enabling True Hardware-Agnostic FPGA Programming. In *Proceedings of the 33rd IEEE International Conference on Application-specific Systems, Architectures, and Processors*, 2022.

Eriksson, D. and Jankowiak, M. High-dimensional bayesian optimization with sparse axis-aligned subspaces. In *Uncertainty in Artificial Intelligence*, pp. 493–503. PMLR, 2021.

Eriksson, D. and Poloczek, M. Scalable Constrained Bayesian Optimization. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pp. 730–738. PMLR, 13–15 Apr 2021.

Eriksson, D., Pearce, M., Gardner, J., Turner, R. D., and Poloczek, M. Scalable Global Optimization via Local Bayesian Optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 5496–5507, 2019.

Fan, Z., Wang, W., Ng, S. H., and Hu, Q. Minimizing ucb: a better local search strategy in local bayesian optimization. *Advances in Neural Information Processing Systems*, 37: 130602–130634, 2024.

Frazier, P. I. and Wang, J. *Bayesian Optimization for Materials Design*, pp. 45–75. Springer International Publishing, Cham, 2016. ISBN 978-3-319-23871-5.

Hansen, N., Brockhoff, D., Mersmann, O., Tusar, T., Tusar, D., ElHara, O. A., Sampaio, P. R., Atamna, A., Varelas, K., Batu, U., Nguyen, D. M., Matzner, F., and Auger, A. Comparing continuous optimizers: numbbo/coco on github, March 2019. URL https://doi.org/10.5281/zenodo.2594848.

Hellsten, E., Hvarfner, C., Papenmeier, L., and Nardi, L. High-dimensional Bayesian Optimization with Group Testing. *arXiv preprint arXiv:2310.03515*, 2023.

Hernández-Lobato, J. M., Requeima, J., Pyzer-Knapp, E. O., and Aspuru-Guzik, A. Parallel and Distributed Thompson Sampling for Large-scale Accelerated Exploration of Chemical Space. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pp. 1470–1479. PMLR, 06–11 Aug 2017.

Hughes, Z. E., Nguyen, M. A., Wang, J., Liu, Y., Swihart, M. T., Poloczek, M., Frazier, P. I., Knecht, M. R., and Walsh, T. R. Tuning Materials-Binding Peptide Sequences toward Gold-and Silver-Binding Selectivity with Bayesian Optimization. *ACS nano*, 15(11):18260–18269, 2021.

Hvarfner, C., Hellsten, E. O., and Nardi, L. Vanilla Bayesian Optimization Performs Great in High Dimensions. In Salakhutdinov, R., Kolter, Z., Heller, K., Weller, A., Oliver, N., Scarlett, J., and Berkenkamp, F. (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 20793–20817. PMLR, 21–27 Jul 2024.

Jones, D. R. Large-scale multi-disciplinary mass optimization in the auto industry. In *MOPTA 2008 Conference (20 August 2008)*, 2008.

Lam, R., Poloczek, M., Frazier, P., and Willcox, K. E. Advances in Bayesian optimization with applications in aerospace engineering. In *2018 AIAA Non-Deterministic Approaches Conference*, pp. 1656, 2018.

Maathuis, H. F., De Breuker, R., and Castro, S. G. High-Dimensional Bayesian Optimisation with Large-Scale Constraints-An Application to Aeroelastic Tailoring. In *AIAA SCITECH 2024 Forum*, pp. 2012, 2024.

Nardi, L., Koeplinger, D., and Olukotun, K. Practical design space exploration. In *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 347–358. IEEE, 2019.

Negoescu, D. M., Frazier, P. I., and Powell, W. B. The Knowledge-Gradient Algorithm for Sequencing Experiments in Drug Discovery. *INFORMS Journal on Computing*, 23(3):346–363, 2011.

Nguyen, Q., Wu, K., Gardner, J., and Garnett, R. Local bayesian optimization via maximizing probability of descent. *Advances in neural information processing systems*, 35:13190–13202, 2022.

Oh, C., Tomczak, J., Gavves, E., and Welling, M. Combinatorial Bayesian Optimization using the Graph Cartesian Product. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.

Papenmeier, L., Nardi, L., and Poloczek, M. Increasing the Scope as You Learn: Adaptive Bayesian Optimization in Nested Subspaces. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, 2022.

Papenmeier, L., Nardi, L., and Poloczek, M. Bounce: Reliable high-dimensional bayesian optimization for combinatorial and mixed spaces. *Advances in Neural Information Processing Systems*, 36:1764–1793, 2023.

Papenmeier, L., Poloczek, M., and Nardi, L. Understanding high-dimensional bayesian optimization. *arXiv preprint arXiv:2502.09198*, 2025.

Ru, B., Wan, X., Dong, X., and Osborne, M. Interpretable Neural Architecture Search via Bayesian Optimisation with Weisfeiler-Lehman Kernels. In *International Conference on Learning Representations*, 2021.

Sala, R. and Müller, R. Benchmarking for metaheuristic black-box optimization: perspectives and open challenges. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8. IEEE, 2020.

Šehić, K., Gramfort, A., Salmon, J., and Nardi, L. LassoBench: A High-Dimensional Hyperparameter Optimization Benchmark Suite for Lasso. In *First Conference on Automated Machine Learning (Main Track)*, 2022a.

Šehić, K., Gramfort, A., Salmon, J., and Nardi, L. LassoBench: A High-Dimensional Hyperparameter Optimization Benchmark Suite for Lasso. In *International Conference on Automated Machine Learning*, pp. 2–1. PMLR, 2022b.

Shen, Y. and Kingsford, C. Computationally efficient high-dimensional bayesian optimization via variable selection. *arXiv preprint arXiv:2109.09264*, 2021.

Snoek, J., Larochelle, H., and Adams, R. P. Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 25, 2012.

Svensson, H. G., Bjerrum, E. J., Tyrchan, C., Engkvist, O., and Chehreghani, M. H. Autonomous drug design with multi-armed bandits. In *2022 IEEE International Conference on Big Data (Big Data)*, pp. 5584–5592. IEEE, 2022.

Tallorin, L., Wang, J., Kim, W. E., Sahu, S., Kosa, N. M., Yang, P., Thompson, M., Gilson, M. K., Frazier, P. I., Burkart, M. D., et al. Discovering de novo peptide substrates for enzymes using machine learning. *Nature communications*, 9(1):1–10, 2018.

Tørring, J. O., Hvarfner, C., Nardi, L., and Själander, M. CATBench: A Compiler Autotuning Benchmarking Suite for Black-box Optimization. *arXiv preprint arXiv:2406.17811*, 2024.

Turner, R., Eriksson, D., McCourt, M., Kiili, J., Laaksonen, E., Xu, Z., and Guyon, I. Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020. In *NeurIPS 2020 Competition and Demonstration Track*, pp. 3–26. PMLR, 2021.

Wang, L., Fonseca, R., and Tian, Y. Learning Search Space Partition for Black-box Optimization using Monte Carlo Tree Search. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:19511–19522, 2020.

Wang, Z., Gehring, C., Kohli, P., and Jegelka, S. Batched large-scale Bayesian optimization in high-dimensional spaces. In *International Conference on Artificial Intelligence and Statistics*, pp. 745–754, 2018.

Xu, Z., Wang, H., Phillips, J. M., and Zhe, S. Standard gaussian process is all you need for high-dimensional bayesian optimization. In *The Thirteenth International Conference on Learning Representations*, 2024.

A. Benchmark Overview

Source	Benchmarks
Šehić et al. (2022a)	`lasso-simple` (d = 60, cont.), `lasso-medium` (d = 100, cont.), `lasso-high` (d = 300, cont.), `lasso-hard` (d = 1000, cont.), `lasso-breastcancer` (d = 10, cont.), `lasso-diabetes` (d = 8, cont.), `lasso-leukemia` (d = 7129, cont.), `lasso-dna` (d = 180, cont.), `lasso-rcv1` (d = 19959, cont.)
Eriksson et al. (2019)	`mopta08` (d = 124, cont., see also Jones (2008)), `rover` (d = 60, cont.), `robotpushing` (d = 14, cont., see also Wang et al. (2018)), `lunarlander` (d = 12, cont.)
Deshwal et al. (2023)	`maxsat60` (d = 60, binary)
Papenmeier et al. (2023)	`maxsat125` (d = 125, binary)
Eriksson & Poloczek (2021)	`svm` (d = 388, cont., this is an adapted version introduced in Papenmeier et al. (2022))
Wang et al. (2020)	`mujoco-ant` (d = 888, cont.), `mujoco-hopper` (d = 33, cont.), `mujoco-walker` (d = 102, cont.), `mujoco-halfcheetah` (d = 102, cont.), `mujoco-swimmer` (d = 16, cont.), `mujoco-humanoid` (d = 6392, cont.)
Oh et al. (2019)	`pestcontrol` (d = 25, cat.)
Hansen et al. (2019); Doerr et al. (2018)	`bbob-sphere` (cont.), `bbob-ellipsoid` (cont.), `bbob-rastrigin` (cont.), `bbob-buecherastrigin` (cont.), `bbob-linearslope` (cont.), `bbob-attractivesector` (cont.), `bbob-stepellipsoid` (cont.), `bbob-rosenbrock` (cont.), `bbob-rosenbrockrotated` (cont.), `bbob-ellipsoidrotated` (cont.), `bbob-discus` (cont.), `bbob-bentcigar` (cont.), `bbob-sharpridge` (cont.), `bbob-differentpowers` (cont.), `bbob-rastriginrotated` (cont.), `bbob-weierstrass` (cont.), `bbob-schaffers10` (cont.), `bbob-schaffers1000` (cont.), `bbob-griewankrosenbrock` (cont.), `bbob-schwefel` (cont.), `bbob-gallagher101` (cont.), `bbob-gallagher21` (cont.), `bbob-katsuura` (cont.), `bbob-lunacekbirastrigin` (cont.)
Doerr et al. (2018)	`pbo-onemax` (binary), `pbo-leadingones` (binary), `pbo-linear` (binary), `pbo-onemaxdummy1` (binary), `pbo-onemaxdummy2` (binary), `pbo-onemaxneutrality` (binary), `pbo-onemaxepistasis` (binary), `pbo-onemaxruggedness1` (binary), `pbo-onemaxruggedness2` (binary), `pbo-onemaxruggedness3` (binary), `pbo-leadingonesdummy1` (binary), `pbo-leadingonesdummy2` (binary), `pbo-leadingonesneutrality` (binary), `pbo-leadingonesepistasis` (binary), `pbo-leadingonesruggedness1` (binary), `pbo-leadingonesruggedness2` (binary), `pbo-leadingonesruggedness3` (binary), `pbo-labs` (binary), `pbo-isingring` (binary), `pbo-isingtorus` (binary), `pbo-isingtriangular` (binary), `pbo-mis` (binary), `pbo-nqueens` (binary), `pbo-concatenatedtrap` (binary), `pbo-nklandscapes` (binary)