Journal of Machine Learning Research 27 (2026) 1-10 Submitted 10/25; Revised 6/26; Published 8/26

OptunaHub: A Platform for Black-Box Optimization

Yoshihiko Ozaki[*,1] YOZAKI@PREFERRED.JP
Shuhei Watanabe[*,†,2] SHUHEI.WATANABE.UTOKYO@GMAIL.COM
Toshihiko Yanase[1] YANASE@PREFERRED.JP
[1]*Preferred Networks, Inc., Otemachi Bldg., 1-6-1 Otemachi, Chiyoda-ku, Tokyo, Japan*
[2]*Independent Researcher*

Editor: Joaquin Vanschoren

Abstract

Black-box optimization (BBO) underpins advances in domains such as AutoML and Materials Informatics, yet implementations of algorithms and benchmarks remain fragmented across research communities. We introduce OptunaHub (`https://hub.optuna.org/`), a community-oriented, decentralized platform for distributing BBO components under a unified Optuna-compatible interface. OptunaHub enables independent publication, discovery, and reuse of optimization algorithms and benchmark problems through a lightweight Python module, a contributor-driven registry, and a searchable web interface. The source code is publicly available in the `optunahub`, `optunahub-registry`, and `optunahub-web` repositories under the Optuna organization on GitHub (`https://github.com/optuna/`).

Keywords: black-box optimization, platform, open-source software, Optuna, Python

1. Introduction

Black-box optimization (BBO) has enabled significant advances in modern research domains, including AutoML (Hutter et al., 2019) and Materials Informatics (Terayama et al., 2021). These advances have driven the development of increasingly sample-efficient algorithms (Turner et al., 2021) and benchmarking toolkits (Hansen et al., 2021; Doerr et al., 2018; Eggensperger et al., 2021; Pfisterer et al., 2022). Generally, existing BBO software efforts fall into two categories: BBO frameworks such as Optuna (Akiba et al., 2019), Nevergrad (Bennet et al., 2021), and pymoo (Blank and Deb, 2020), which provide algorithm collections within centrally maintained codebases; and benchmarking and profiling toolkits such as COCO (Hansen et al., 2021) and IOHprofiler (Doerr et al., 2018), which focus primarily on standardized evaluation. While these efforts have substantially improved the dissemination of advanced optimization algorithms as well as reproducibility and evaluation practices, they are not intended to distribute modules implemented by independent contributors. This centralized design promotes coherence and standardization, but algorithm and benchmark implementations often remain fragmented across research communities and software ecosystems, ultimately hindering cross-domain reuse and broader adoption of new optimization methods.

∗. These authors equally contributed to this work.
†. This work was conducted while the author was affiliated with Preferred Networks, Inc.

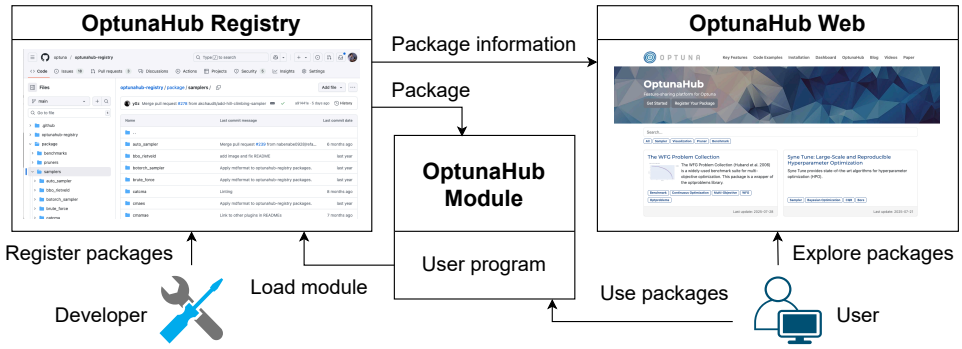

Figure 1: Conceptual visualization of the relationships between OptunaHub components. The OptunaHub Module enables users to load packages from the OptunaHub Registry, while package information is available via the OptunaHub Web interface. All packages integrate seamlessly with the Optuna interface, allowing different algorithms to be applied across problems with minimal modification. The web interface also provides full-text search for efficient package discovery.

By comparison, the broader machine learning community has undergone a paradigm shift with the emergence of the Hugging Face Hub, which enables researchers to independently publish, discover, and reuse models and data sets through a unified interface. This model has influenced research workflows by improving visibility, interoperability, and reuse (Osborne et al., 2024). Despite the rapid growth of BBO research, a comparable community-driven distribution infrastructure for optimization algorithms, benchmark problems, and related utilities has yet to emerge.

To this end, we propose *OptunaHub* (`https://hub.optuna.org/`), a community-oriented platform for BBO. OptunaHub consists of (1) a Python library (*OptunaHub Module*) providing unified APIs built on top of Optuna (Akiba et al., 2019), (2) a package registry (*OptunaHub Registry*) that enables contributors to independently develop and distribute algorithms, benchmark problems, and utilities with isolated dependencies, and (3) a web interface (*OptunaHub Web*) that facilitates searchable discovery of registered packages (Figure 1). Our goal is to reduce fragmentation in BBO research by fostering cross-domain exchange and sustained interaction between contributors and practitioners. The OptunaHub infrastructure increases not only the visibility and discoverability of optimization algorithms but also their reusability thanks to the unified interface backed by the Optuna assets. Appendix A further details the position of OptunaHub within the BBO software landscape.

2. OptunaHub Ecosystem

This section describes the three major components of OptunaHub illustrated in Figure 1: OptunaHub Module (a Python library providing unified APIs and utilities for interacting with registry packages), OptunaHub Registry (a repository of contributed packages), and OptunaHub Web (a web interface that aggregates and presents package information).

Code 1: An example codeblock to run modules via `optunahub`.

```
1  from optuna import create_study
2  from optuna.visualization import plot_optimization_history,
       plot_param_importances
3  from optunahub import load_module
4
5  # All modules have their own package pages at https://hub.optuna.org/.
6  auto_sampler = load_module("samplers/auto_sampler")
7  nasbench201 = load_module("benchmarks/nasbench201")
8
9  sampler = auto_sampler.AutoSampler()
10 objective = nasbench201.Problem(dataset_id=0)  # Use the CIFAR-10 data set.
11 study = create_study(sampler=sampler, directions=objective.directions)
12 study.optimize(objective, n_trials=100)
13 plot_optimization_history(study).show()
14 plot_param_importances(study).show()
```

2.1 OptunaHub Module: Unified APIs Compatible with Optuna

OptunaHub Module, available via `pip install optunahub`, provides two primary features: (1) the `load_module` function and (2) a set of base classes (e.g., `SimpleBaseSampler` and `BaseProblem`) for implementing Optuna-compatible interfaces.[1]

The `load_module` function dynamically imports a module from the OptunaHub Registry package specified as `load_module("category/package_name")`. For example, Lines 6–7 in Code 1 load `"samplers/auto_sampler"`, a meta-sampler that automatically selects an appropriate optimization strategy, and `"benchmarks/nasbench201"`, a wrapper of NAS-Bench-201 (Dong and Yang, 2020). Downloaded packages via `load_module` are cached locally, enabling subsequent offline use. Each package has a corresponding web page (Section 2.3) that documents its available APIs.

Through the provided base classes, samplers (i.e., BBO algorithms) and benchmarks conform to the Optuna interface. This compatibility enables users to swap samplers and benchmark problems seamlessly and to analyze optimization results using standard Optuna utilities (Lines 13–14). Optuna itself is a widely adopted optimization framework, and registered packages benefit directly from its mature backend and features.

2.2 OptunaHub Registry: Gateway for Community Contributions

OptunaHub Registry enables researchers and practitioners to share modular packages of BBO functionality that are accessible through the Optuna interface. Each package—typically an optimization algorithm or benchmark suite—behaves as an independent Python package. These packages can be fetched individually via `load_module`, and may specify their own dependencies. Importantly, the fact that OptunaHub accepts new packages from the community distinguishes it from traditional BBO libraries such as Optuna (Akiba et al., 2019), Nevergrad (Bennet et al., 2021), and pymoo (Blank and Deb, 2020), which provide a predefined set of functionality. Appendix B details the package registration process, registry structure, and package components.

1. `https://optuna.github.io/optunahub/reference.html`

As of this writing, more than 100 packages are available. The sampler collection ranges from classical methods, such as Nelder–Mead (Nelder and Mead, 1965), to recent approaches including HEBO (Cowen-Rivers et al., 2022), Vizier (Song et al., 2022), and LLM-enhanced Bayesian optimization (Liu et al., 2024). Several methods have been contributed by their original authors, including CatCMA (Hamano et al., 2024), SMAC (Lindauer et al., 2022), SyneTune (Salinas et al., 2022), and PFNs4BO (Müller et al., 2023). The registry also accepts packages providing benchmarks, pruning methods, visualization tools, and callback functions. Registered benchmarks span synthetic function suites such as BBOB (Hansen et al., 2009) and WFG (Huband et al., 2006), AutoML-related tasks including HPO/NAS-Bench (Klein and Hutter, 2019; Eggensperger et al., 2021; Dong and Yang, 2020; Dong et al., 2021), and real-world problem benchmarks such as aircraft design (Namura, 2025).

2.3 OptunaHub Web: Package Catalog with Full-Text & Tag Search

OptunaHub Web serves as the discovery and presentation layer of the ecosystem, enhancing package accessibility. It provides individual package pages that are automatically generated from each package's `README.md` (cf. Appendix B), ensuring that documentation remains synchronized with the source repository. In addition, the web interface offers full-text and tag-based search capabilities, enabling users to efficiently locate relevant algorithms, benchmarks, and utilities across domains. The top page presents package thumbnails, tags, and summaries to facilitate browsing. Each package page includes author and license information, a summary, tags, dependency details, API documentation, code examples, and links for discussion or bug reporting. Figure 2 illustrates the `auto_sampler` package page as an example.[2] By standardizing package documentation and presentation, OptunaHub Web improves discoverability and reuse while providing contributors with a consistent way to disseminate and maintain their work.

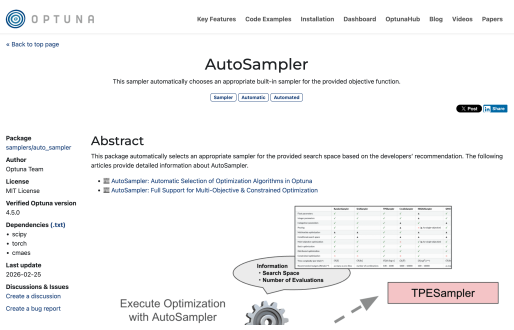

Figure 2: Package page example.

3. Final Remarks

Open-source software and collaborative development play a central role in modern scientific research. In this paper, we introduced OptunaHub, a decentralized distribution platform for BBO components built on top of Optuna. By combining modular package development with a unified execution interface, OptunaHub enables independent contribution while preserving interoperability and reproducibility. As the ecosystem continues to evolve, we expect OptunaHub to complement existing BBO libraries and benchmarking toolkits by providing infrastructure for community-driven dissemination of algorithms and benchmark problems. We hope this platform will contribute to broader cross-domain reuse and sustained progress in BBO research.

2. `https://hub.optuna.org/samplers/auto_sampler/`

Acknowledgments

We thank the editor and the anonymous reviewers for their constructive comments and suggestions. We also thank Kei Akita, Kaito Baba, Yugo Fusawa, Hideaki Imamura, Naoto Mizuno, Masashi Shibata, Hiroki Takizawa, and Yunzhuo Wang for their support. We acknowledge the BBO community, particularly the contributors to OptunaHub. We thank Yasuhiro Fujita for his advice on this submission.

Appendix A. OptunaHub's Position within the BBO Software Landscape

This appendix provides a structured comparison of OptunaHub with representative software efforts in BBO. Our goal is not to rank existing tools, but to clarify differences in design philosophy, governance structure, and intended use cases.

A.1 Library-Centric BBO Frameworks

Optuna (Akiba et al., 2019), Nevergrad (Bennet et al., 2021), pymoo (Blank and Deb, 2020), and several recently proposed BBO frameworks (Tian et al., 2017; Thieu and Mirjalili, 2023; Jiang et al., 2024) provide extensive collections of optimization algorithms (and some of them also provide benchmark problems) within centrally maintained codebases. These libraries offer rich functionality and coherent APIs, enabling users to apply a variety of algorithms through a unified interface. New features and algorithms are typically integrated directly into a shared codebase and maintained under unified project governance. Such centralized integration facilitates consistency and quality control. However, the primary objective of these frameworks is to provide comprehensive optimization toolkits rather than to function as open distribution platforms for third-party components.

Moreover, some libraries are developed with a specific methodological focus or problem domain in mind. For example, pymoo and PlatEMO (Tian et al., 2017) are primarily oriented toward evolutionary multi-objective optimization within the broader BBO landscape.

A.2 Benchmarking and Profiling Toolkits

Benchmarking and profiling toolkits such as COCO (Hansen et al., 2021) and IOHprofiler (Doerr et al., 2018) focus on standardized evaluation and performance analysis of optimization algorithms. They provide curated benchmark suites, experimental protocols, and profiling tools to support reproducible comparison. These toolkits advance methodological rigor in BBO research. However, their design centers on carefully curated evaluation infrastructures, where benchmark suites are defined and maintained within a centralized framework rather than distributed as independently developed packages.

In terms of problem coverage, the benchmarking toolkits mentioned above emphasize synthetic test functions, such as BBOB (Hansen et al., 2009), designed to analyze algorithmic properties under controlled conditions. Some benchmark collections instead focus on specific application domains, e.g., HPO/NAS-Bench (Klein and Hutter, 2019; Eggensperger et al., 2021; Dong and Yang, 2020; Dong et al., 2021) target AutoML tasks. While these efforts provide valuable resources for their respective purposes, they are typically distributed

as curated data sets or evaluation toolkits rather than as open registries enabling third-party contributors to publish and maintain diverse benchmark implementations.

OptunaHub complements these approaches by enabling contributors to implement and distribute benchmark problems as modular packages under a unified interface. This decentralized distribution model allows the ecosystem to expand beyond predefined synthetic suites or domain-specific collections, supporting a broader range of application-driven and community-contributed benchmarks. Importantly, OptunaHub Web enhances searchability across domains and communities.

A.3 OptunaHub's Design Philosophy

OptunaHub differs from the above projects in its governance and distribution model. Rather than integrating all functionality into a shared codebase, it adopts a decentralized package registry in which each component—such as an optimization algorithm or a benchmark problem—can be developed, versioned, and maintained independently by its contributors. All packages conform to Optuna-compatible interfaces, ensuring runtime interoperability, while implementation details and dependencies remain encapsulated within each package. This modular separation allows contributors to develop components independently without modifying or relying on other packages. The review process follows lightweight criteria that emphasize basic functionality, safety, and ethical compliance, thereby lowering the contribution barrier while preserving ecosystem integrity (cf. Appendix B). Additionally, OptunaHub Web facilitates the discovery of registered packages and access to their contributor-maintained documentation, enhancing visibility and usability across the ecosystem.

Overall, OptunaHub introduces a complementary distribution layer that enables decentralized publication and discovery of BBO components under a unified execution interface, rather than replacing existing libraries or benchmarking toolkits. Furthermore, OptunaHub is designed as a method-agnostic, domain-flexible infrastructure: it does not prioritize specific optimization paradigms, such as evolutionary algorithms or Bayesian optimization, or predefined problem classes, but instead provides a unified interface through which diverse algorithms and application domains can coexist within a shared ecosystem.

Finally, OptunaHub's method-agnostic stance is particularly relevant for optimization paradigms whose methodological rigor and empirical effectiveness have been critically questioned in the BBO community, such as metaphor-based heuristics (Aranha et al., 2022). Recent benchmarking and critical-analysis studies have further examined these concerns, raising questions about the empirical performance and novelty of some proposed methods (Vermetten et al., 2024; Halsema et al., 2024). Rather than adjudicating such debates, OptunaHub is designed to support transparent dissemination and open community discussion by providing accessible implementations that enable independent evaluation and reproducibility. This emphasis on openness is particularly important in areas where methodological debates remain active, as it enables the broader community to rigorously evaluate and discuss proposed methods. OptunaHub's transparent discussion channels on GitHub are intended to support precisely this kind of community-driven scrutiny.

Appendix B. OptunaHub Package Registration & Development

This appendix describes the package registration process and outlines the structure of the OptunaHub Registry and its packages.

B.1 Overview of the Package Registration Process

OptunaHub accepts new package registrations and updates through pull requests to the `optunahub-registry` repository on GitHub. Pull requests are reviewed by at least one OptunaHub committer to ensure basic functionality (by verifying a minimal working example), safety (e.g., absence of malicious code), compliance with licensing requirements, and adherence to general research and engineering ethics. Each package is primarily maintained by its contributors, who are responsible for the scientific claims, implementation details, and long-term maintenance of their work. OptunaHub committers oversee alignment with contribution guidelines during code review, but do not independently verify the empirical claims or scientific validity of contributed methods. This separation of responsibilities reflects OptunaHub's design as an open distribution platform: inclusion of a package indicates that the contribution meets basic engineering and ethical standards, not an endorsement of the empirical performance or scientific validity. As with open distribution platforms in other research communities, the platform's role is to support transparent dissemination and community-based evaluation, enabling the broader research community to assess the contributed methods directly (cf. Section A.3).

OptunaHub's review practices differ from the stricter quality assurance standards adopted by Optuna. Rather than requiring extensive design reviews or comprehensive unit testing, OptunaHub follows lightweight review criteria that emphasize basic functionality and safety. This approach balances baseline quality with broad community participation.

B.2 Registry & Package Details

OptunaHub Registry follows a category–package directory hierarchy, where each package resides in its corresponding category. An OptunaHub package is a self-contained directory, similar to a standard Python package, that implements specific functionality with its own dependencies. While all packages conform to the Optuna-compatible interface, contributors can be agnostic to other packages and focus on developing their packages independently. As described in Section 2.1, packages can be loaded directly in user code via `load_module("category/package_name")` without `pip install` or explicit `import`. OptunaHub package versions correspond to Git commits. By specifying `ref="commit_hash"` in `load_module`, users can load the version corresponding to a particular commit of `optunahub-registry`. If no reference is provided, the latest version is loaded by default.

Figure 3 shows the structure of the `auto_sampler` package as an example of a package. The package is located in the "samplers" directory as it belongs to the "samplers" category. In the package, `__init__.py` and `_sampler.py` are the implementation of the functionality. `LICENSE` is the software license file for the package, which must be included. `README.md` contains metadata (e.g., abstract, authors, and tags) and descriptions of the package (typically, API documentation, benchmark results, and citations), which is published as a package details page (cf. Section 2.3). Image files used in `README.md` can be placed in the "images" directory (the first appearing image is used as the thumbnail of the package). `requirements.txt`, which is optional, lists the package-specific dependencies. Finally, contributors are encouraged to include test code in their packages; however, to lower the barrier to contribution, unit tests are not strictly required at the time of initial submission. For review purposes, a functional example demonstrating basic usage must be provided either in the pull request description, in `README.md`, or in a separate file (e.g., `example.py`). Further instructions are available in the official tutorials for contributors.[3]

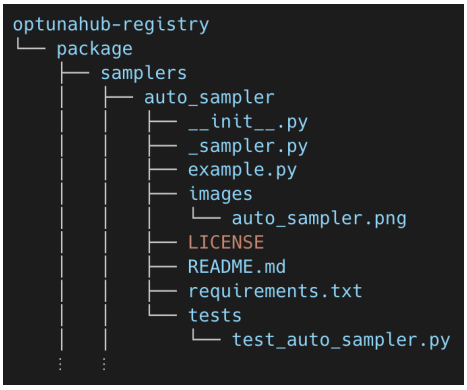

Figure 3: A package example.

References

T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019.

C. Aranha, L. Christian V. Camacho, F. Campelo, M. Dorigo, R. Ruiz, M. Sevaux, K. Sörensen, and T. Stützle. Metaphor-based metaheuristics, a call for action: The elephant in the room. *Swarm Intelligence*, 16(1):1–6, 2022.

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

J. Blank and K. Deb. Pymoo: Multi-objective optimization in python. *IEEE Access*, 8: 89497–89509, 2020.

AI. Cowen-Rivers, W. Lyu, R. Tutunov, Z. Wang, A. Grosnit, RR. Griffiths, AM. Maraval, H. Jianye, J. Wang, J. Peters, and H. Bou-Ammar. HEBO: Pushing the limits of sample-efficient hyper-parameter optimisation. *Journal of Artificial Intelligence Research*, 74, 2022.

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

X. Dong and Y. Yang. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020.

3. `https://optuna.github.io/optunahub/tutorials_for_contributors.html`

X. Dong, L. Liu, K. Musial, and B. Gabrys. NATS-Bench: Benchmarking NAS algorithms for architecture topology and size. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

K. Eggensperger, P. Müller, N. Mallik, M. Feurer, R. Sass, A. Klein, N. Awad, M. Lindauer, and F. Hutter. HPOBench: A collection of reproducible multi-fidelity benchmark problems for HPO. In *NeurIPS (Datasets and Benchmarks Track)*, 2021.

M. Halsema, D. Vermetten, T. Bäck, and N. Van Stein. A critical analysis of raven roost optimization. In *Genetic and Evolutionary Computation Conference Companion*, pages 1993–2001, 2024.

R. Hamano, S. Saito, M. Nomura, K. Uchida, and S. Shirakawa. CatCMA: Stochastic optimization for mixed-category problems. In *Genetic and Evolutionary Computation Conference*, 2024.

N. Hansen, S. Finck, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. Technical report, INRIA, 2009.

N. Hansen, A. Auger, R. Ros, O. Mersmann, T. Tušar, and D. Brockhoff. COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36, 2021.

S. Huband, P. Hingston, L. Barone, and L. While. A review of multiobjective test problems and a scalable test problem toolkit. *IEEE Transactions on Evolutionary Computation*, 10, 2006.

F. Hutter, L. Kotthoff, and J. Vanschoren. *Automated Machine Learning: Methods, Systems, Challenges*. Springer Nature, 2019.

H. Jiang, Y. Shen, Y. Li, B. Xu, S. Du, W. Zhang, C. Zhang, and B. Cui. Openbox: A Python toolkit for generalized black-box optimization. *Journal of Machine Learning Research*, 25, 2024.

A. Klein and F. Hutter. Tabular benchmarks for joint architecture and hyperparameter optimization. *arXiv:1905.04970*, 2019.

M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter. SMAC3: A versatile Bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23, 2022.

T. Liu, N. Astorga, N. Seedat, and M. van der Schaar. Large language models to enhance Bayesian optimization. In *International Conference on Learning Representations*, 2024.

S. Müller, M. Feurer, N. Hollmann, and F. Hutter. PFNs4BO: In-context learning for Bayesian optimization. In *International Conference on Machine Learning*, 2023.

N. Namura. Single and multi-objective optimization benchmark problems focusing on human-powered aircraft design. In *International Conference on Evolutionary Multi-Criterion Optimization*, 2025.

JA. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7, 1965.

C. Osborne, J. Ding, and HR. Kirk. The AI community building the future? A quantitative analysis of development activity on Hugging Face Hub. *Journal of Computational Social Science*, 7, 2024.

F. Pfisterer, L. Schneider, J. Moosbauer, M. Binder, and B. Bischl. YAHPO gym - an efficient multi-objective multi-fidelity benchmark for hyperparameter optimization. In *International Conference on Automated Machine Learning*. PMLR, 2022.

D. Salinas, M. Seeger, A. Klein, V. Perrone, M. Wistuba, and C. Archambeau. Syne Tune: A library for large scale hyperparameter tuning and reproducible research. In *International Conference on Automated Machine Learning*, 2022.

X. Song, S. Perel, C. Lee, G. Kochanski, and D. Golovin. Open source Vizier: Distributed infrastructure and API for reliable and flexible blackbox optimization. In *International Conference on Automated Machine Learning*, 2022.

K. Terayama, M. Sumita, R. Tamura, and K. Tsuda. Black-box optimization for automated discovery. *Accounts of Chemical Research*, 54, 2021.

N. Van Thieu and S. Mirjalili. MEALPY: An open-source library for latest meta-heuristic algorithms in Python. *Journal of Systems Architecture*, 139, 2023.

Y. Tian, R. Cheng, X. Zhang, and Y. Jin. PlatEMO: A MATLAB platform for evolutionary multi-objective optimization [educational forum]. *IEEE Computational Intelligence Magazine*, 12, 2017.

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

D. Vermetten, C. Doerr, H. Wang, AV. Kononova, and T. Bäck. Large-scale benchmarking of metaphor-based optimization heuristics. In *Genetic and Evolutionary Computation Conference*, pages 41–49, 2024.