The Journal of Open Source Software

DeepHyper: A Python Package for Massively Parallel Hyperparameter Optimization in Machine Learning

Romain Egele[1*¶], Prasanna Balaprakash[1*], Gavin M. Wiggins[1], and Brett Eiffert[1]

1 Oak Ridge National Laboratory, Oak Ridge, TN, United States ¶ Corresponding author * These authors contributed equally.

DOI: 10.21105/joss.07975

Software

- Review
- Repository
- Archive

Editor: Mehmet Hakan Satman

Reviewers:

- @jungtaekkim
- @y0z

Submitted: 10 March 2025
Published: 12 May 2025

Summary

Machine learning models are increasingly applied across scientific disciplines, yet their effectiveness often hinges on heuristic decisions—such as data transformations, training strategies, and model architectures—that are not learned by the models themselves. Automating the selection of these heuristics and analyzing their sensitivity is crucial for building robust and efficient learning workflows. `DeepHyper` addresses this challenge by democratizing hyperparameter optimization, providing accessible tools to streamline and enhance machine learning workflows from a laptop to the largest supercomputer in the world. Building on top of hyperparameter optimization, it unlocks new capabilities around ensembles of models for improved accuracy and uncertainty quantification. All of these organized around efficient parallel computing.

Statement of need

`DeepHyper` is a Python package for parallel hyperparameter optimization or neural architecture search. The project started in 2018 (Balaprakash et al., 2018) with a focus on making Bayesian optimization more efficient on high-performance computing clusters. It provides access to a variety of asynchronous parallel black-box optimization algorithms via `deephyper.hpo`. The software offers a variety of parallel programming backends such as Asyncio, threading, processes, Ray, and MPI via `deehyper.evaluator`. The hyperparameter optimization can be single or multi-objective, composed of mixed variables, using explicit or hidden constraints, and benefit from early-discarding strategies via `deephyper.stopper`. Leveraging the results of hyperparameter optimization or neural architecture search it provides parallel ensemble algorithms via `deephyper.ensemble` that can help improve accuracy or quantify disentangled predictive uncertainty. A diagram of our software architecture is shown in Figure 1.

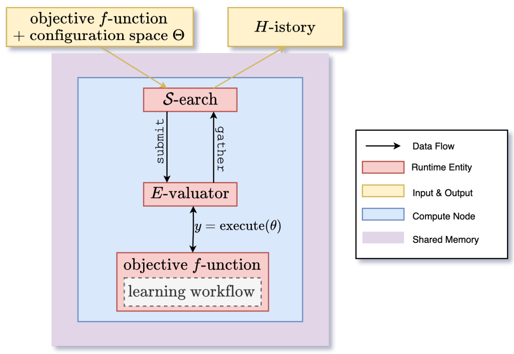

Figure 1: DeepHyper Software Architecture

`DeepHyper` was designed to help research in the field of automated machine learning and also to be used out-of-the box in scientific projects where learning workflows are being developed.

Related Work

Numerous software packages now exist for hyperparameter optimization (HPO), including:

- BoTorch (and Ax platform) (Balandat et al., 2020), (doc.)
- HyperMapper (Nardi et al., 2019), (doc.)
- Hyperopt (J. Bergstra et al., 2013; James Bergstra et al., 2015), (doc.)
- OpenBox (Jiang et al., 2024), (doc.)
- Optuna (Akiba et al., 2019), (doc.)
- SMAC3 (Lindauer et al., 2022), (doc.)

These tools differ in scope and design: some are research-oriented (e.g., SMAC), while others prioritize production-readiness and usability (e.g., Optuna). In this section, we focus our comparison on SMAC and Optuna, which are representative of these two directions.

DeepHyper is designed to maximize HPO efficiency across a wide range of parallelization scales, from sequential (single-core) runs to massively parallel evaluations on high-performance computing (HPC) systems with thousands of cores.

Feature Comparison

While the feature matrix below provides a high-level overview, it necessarily simplifies some nuanced implementation differences. We use the ✓ symbol for available features, the ≈ symbol for incomplete features, and no symbol for missing features.

Hyperparameter Optimization Capabilities

Table 1: Overview of optimization features available in different packages.

Feature	DeepHyper	Optuna	SMAC3
Single-objective	✓	✓	✓
Multi-objective	✓	✓	✓
Early stopping	✓	✓	✓
Fault tolerance	✓	✓	
Transfer learning	✓	✓	≈
Ensemble construction	✓		
Visualization	≈	✓	

Single-objective Optimization
DeepHyper employs a surrogate model based on random forests to estimate $P(\text{Objective} \mid \text{Hyperparameters})$, similar to SMAC. However, DeepHyper's implementation is typically faster per query, especially when the number of evaluations exceeds 200. In contrast, Optuna uses the Tree-structured Parzen Estimator (TPE), which models $P(\text{Hyperparameters} \mid \text{Objective})$. TPE offers faster query times but can struggle with complex optimization landscapes and tends to be less effective in refining continuous hyperparameters.

Multi-objective Optimization
DeepHyper uses scalarization-based approaches inspired by ParEGO (also used in SMAC), with randomized weights and a variety of scalarization functions. Optuna defaults to NSGA-II, a genetic algorithm that evolves solutions along estimated Pareto fronts. NSGA-II typically converges more slowly but catches up when the evaluation budget is sufficient.

Early Discarding
DeepHyper supports several early stopping techniques, including constant thresholds, median stopping rules, successive halving, and learning curve extrapolation. Optuna also offers several early discarding methods among which some are different from DeepHyper's.

Fault Tolerance
DeepHyper handles failed evaluations by assigning them the worst observed objective value, thereby preserving optimizer stability and avoiding cascading failures.

Transfer Learning
DeepHyper supports warm-starting optimizations using data from prior runs. This is effective when either the objective function changes while the search space remains fixed (e.g., different datasets), or when the search space expands (e.g., broader neural architecture configurations).

Ensemble Construction
DeepHyper enables model ensembling from the pool of evaluated configurations, helping to reduce variance and capture epistemic uncertainty in predictions.

Visualization
Basic visualization tools are provided via `deephyper.analytics`. For more interactive exploration, we recommend SandDance, a Visual Studio Code plugin. Figure 2 illustrates a 3D visualization of a Random Forest optimization, with `min_samples_split`, `min_weight_fraction_leaf`, and test accuracy as the x, y, and z axes (and color), respectively. The two plots compare configurations using `splitter="best"` (left) and `splitter="random"` (right). Such visualizations help identify the sensitivity of the objective to different hyperparameters.

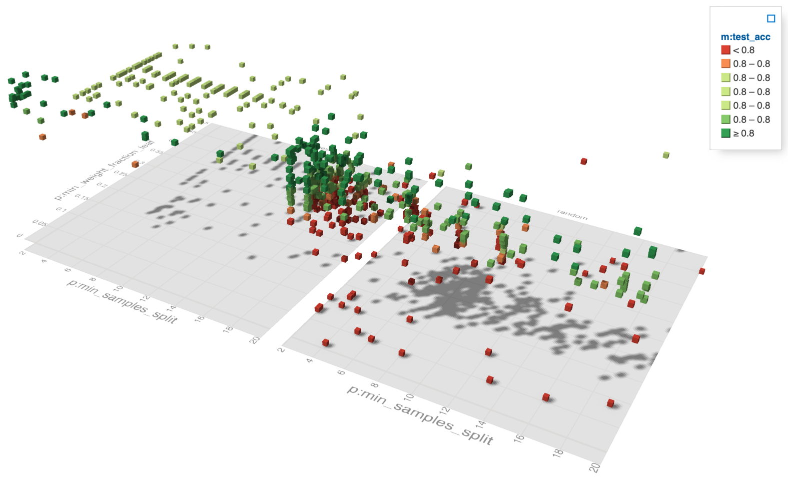

Figure 2: Visualization of hyperparameter optimization results with SandDance

Parallelization Capabilities

Table 2: Overview of parallelization features available in different packages.

	DeepHyper	Optuna	SMAC3
Asynchronous optimization	✓	✓	?
Centralized optimization	✓	✓	
Decentralized optimization	✓	✓	
Parallelization backends	✓		≈
Memory backends	✓	✓	≈

The main difference between DeepHyper, Optuna and SMAC related to parallelization is that DeepHyper provides out-of-the-box parallelization software while Optuna leaves it to the user and SMAC limits itself to centralized parallelism with Dask.

Asynchronous optimization: DeepHyper's allows to submit and gather hyperparameter configuration by batch and asynchronously (in a centralized or decentralized setting).

Centralized optimization: DeepHyper's allows to run centralized optimization, including 1 master running the optimization and N workers evaluating hyperparameter configurations.

Decentralized optimization: DeepHyper's allows to run decentralized optimization, including N workers, each running centralized optimization.

Parallelization backends: DeepHyper's provides compatibility with several parallelization backends: AsyncIO functions, thread-based, process-based, Ray, and MPI. This allows to easily adapt to the context of the execution.

Memory backends: DeepHyper's provides compatibility with several shared-memory backends: local memory, server managed memory, MPI-based remote memory access, Ray.

Black-box Optimization Benchmarks

The benchmark is run on 9 different continuous black-box benchmark functions: Ackley (5 dim.), Branin (2 dim.), Griewank (5 dim.), Hartmann (6 dim.), Levy (5 dim.), Michalewicz (5 dim.), Rosen (5 dim.), Schwefel (5 dim.) and Shekel (5 dim.). Figures 3-7 present the benchmark results with the average regret and the standard error over 10 random repetitions for each method. The average regret is given by $y^* - \hat{y}^*$ where y^* is the true optimal objective value and $\hat{y}^*$ is the current estimated best optimal objective value. All methods are run sequentially (no parallelization features enabled for consistent evaluation) with default arguments. For DeepHyper, the `CBO` search is used. For Optuna, the `TPE` sampler with default arguments is used. For SMAC, the default parameters using the OptunaHub SMAC sampler is used.

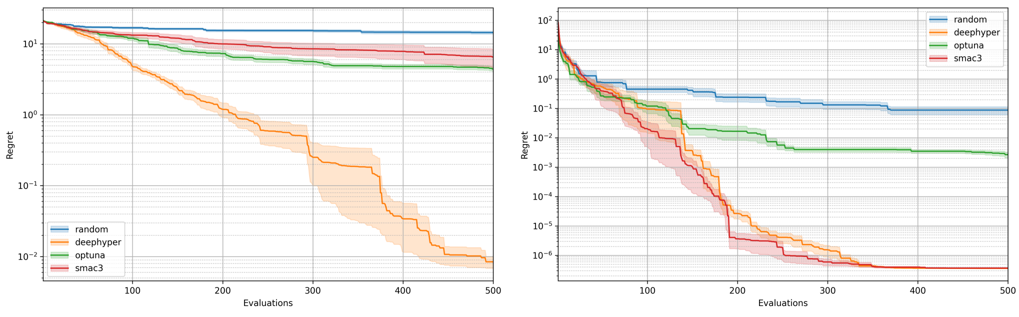

Figure 3: Ackley 5D (left) and Branin 2D (right)

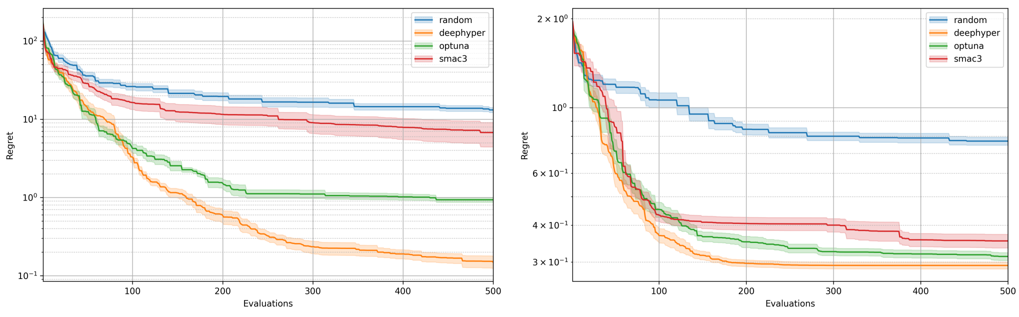

Figure 4: Griewank 5D (left) and Hartmann 6D (right)

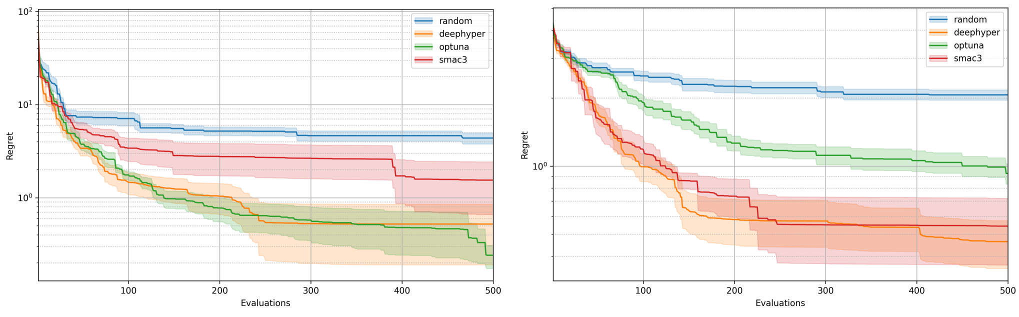

Figure 5: Levy 5D (left) and Michalewicz 5D (right)

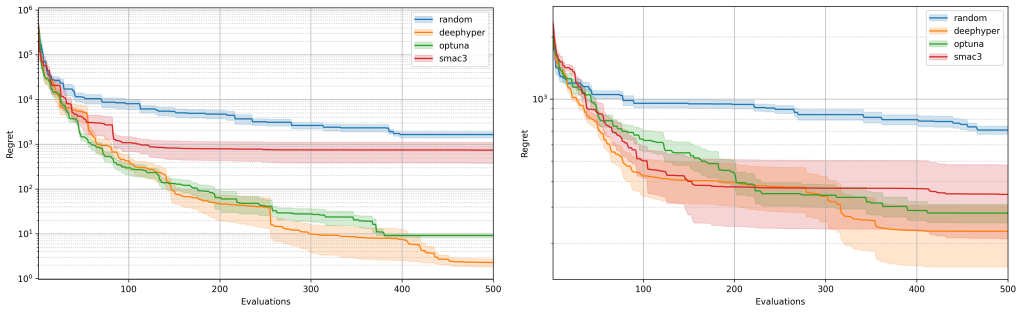

Figure 6: Rosen 5D (left) and Schwefel 5D (right)

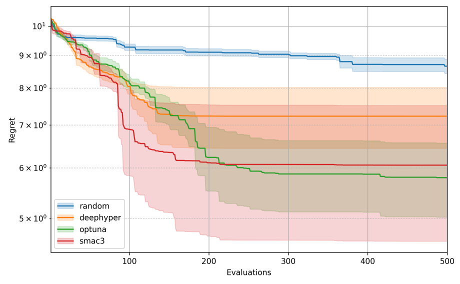

Figure 7: Shekel 5D

Black-box Optimization Methods

The algorithms used for hyperparameter optimization are black-box optimization algorithms for mixed search spaces. A mixed search space, can be composed of real, discrete, or categorical (nominal or ordinal) values. The search space can also include constraints (e.g., explicit such as $x_0 < x_1$, or implicit such as "unexpected out-of-memory error"). The objective function $Y = f(x)$ can be stochastic where Y is a random variable, x is a vector of input hyperparameters, and f is the objective function. The `DeepHyper`, main hyperparameter optimization algorithm, is based on Bayesian optimization.

The Bayesian optimization of `DeepHyper` relies on Extremely Randomized Forest as surrogate model to estimate $E_Y[Y|X = x]$ by default. Extremely Randomized Forests (Geurts et al., 2006) are a type of Randomized Forest algorithms in which the split decision involves a random process for each newly created node of a tree. It provides smoother epistemic uncertainty estimates with an increasing number of trees (left side in Figure 8) compared to usual Random Forests that use a deterministic "best" split decision (right side in Figure 8).

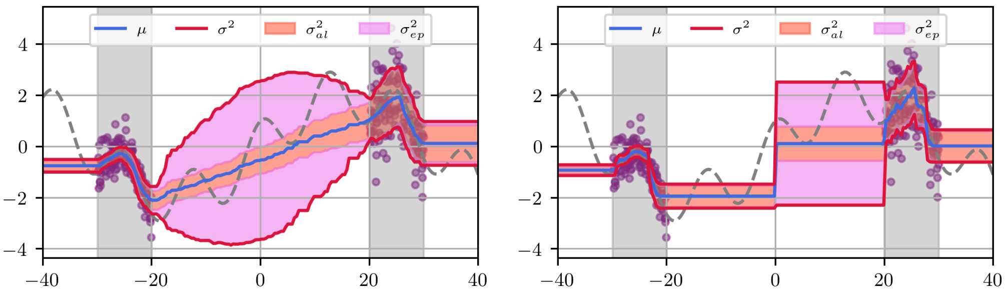

Figure 8: Uncertainty of Randomized Forests, on the left-side with random split, and on the right-side with best split.

Then, a custom acquisition function UCBd$(x) = \mu(x) + \kappa \cdot \sigma_{\text{ep}}(x)$, combines the mean predicition $\mu(x)$ with the epistemic uncertainty $\sigma_{\text{ep}}(x)$ of this surrogate (purple area in Figure 8) for improved efficiency. We also apply a periodic exponential decay (Figure 9), impacting the exploration-exploitation parameter κ to escape local solutions (Egelé et al., 2023).

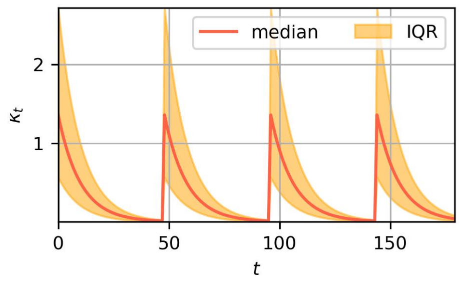

Figure 9: Periodic Exponential Decay for Bayesian Optimization

Batch parallel genetic algorithms are provided to resolve efficiently the sub-problem of optimizing the acquisition function and it is also more accurate than Monte-Carlo approaches to find the optimum of the acquisition function. A cheap and efficient multi-point acquisition strategy `qUCBd` is provided for better parallel scalability (Egelé et al., 2023).

The multi-objective optimization is enabled by scalarization functions and objective rescaling (Égelé et al., 2023).

The early-discarding strategies include asynchronous successive halving and a robust learning curve extrapolation (Egele et al., 2024).

The ensemble strategies is modular to allow: exploring models tested during hyperparameter optimization, classification and regression problems to be treated, disentangled uncertainty quantification (Égelé et al., 2022). It can leverage the same parallelization features as the optimization.

Finally, a transfer-learning strategy for hyperparameter optimization (Dorier et al., 2022) is available. This strategy used to be based on variational auto-encoders for tabular data. It is now based on the Gaussian mixture-model for tabular data.

Software Development

The `DeepHyper` package adheres to best practices in the Python community by following the PEP 8 style guide for Python naming and formatting conventions. The layout and configuration of the package follows suggestions made by the Python Packaging User Guide which is maintained by the Python Packaging Authority. A `pyproject.toml` file provides all configuration settings and meta data for developing and publishing the `DeepHyper` package. The ruff tool is used to enforce style and format conventions during code development and during continuous integration checks via GitHub Actions. Unit tests are also conducted in the CI workflow with the pytest framework. Conda environments and standard Python virtual environments are supported by the project to ensure consistent development environments. Documentation for the package is generated with the Sphinx application. See the Developer's Guide section in the DeepHyper documentation for contributing guidelines and more software development information.

Acknowledgements

We acknowledge contributions from Misha Salim, Romit Maulik, Venkat Vishwanath, Stefan Wild, Joceran Gouneau, Dipendra Jha, Kyle Felker, Matthieu Dorier, Felix Perez, Bethany Lush, Gavin M. Wiggins, Tyler H. Chang, Yixuan Sun, Shengli Jiang, rmjcs2020, Albert Lam, Taylor Childers, Z223I, Zachariah Carmichael, Hongyuan Liu, Sam Foreman, Akalanka Galappaththi, Brett Eiffert, Sun Haozhe, Sandeep Madireddy, Adrian Perez Dieguez, and Nesar Ramachandra. We also would like to thank Prof. Isabelle Guyon for her guidance on the machine learning methodologies developed in this software.

References

Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2623–2631. https://doi.org/10.1145/3292500.3330701

Balandat, M., Karrer, B., Jiang, D. R., Daulton, S., Letham, B., Wilson, A. G., & Bakshy, E. (2020). BOTORCH: A framework for efficient monte-carlo Bayesian optimization. *Proceedings of the 34th International Conference on Neural Information Processing Systems*. ISBN: 9781713829546

Balaprakash, P., Salim, M., Uram, T. D., Vishwanath, V., & Wild, S. M. (2018). DeepHyper: Asynchronous hyperparameter search for deep neural networks. *IEEE 25th International Conference on High Performance Computing (HiPC)*, 42–51. https://doi.org/10.1109/HiPC.2018.00014

Bergstra, James, Komer, B., Eliasmith, C., Yamins, D., & Cox, D. D. (2015). Hyperopt: A Python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, *8*(1), 014008. https://doi.org/10.1088/1749-4699/8/1/014008

Bergstra, J., Yamins, D., & Cox, D. D. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, I-115-I-123. https://dl.acm.org/doi/10.5555/3042817.3042832

Dorier, M., Égelé, R., Balaprakash, P., Koo, J., Madireddy, S., Ramesh, S., Malony, A. D., & Ross, R. (2022). HPC storage service autotuning using variational- autoencoder -guided asynchronous Bayesian optimization. *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, 381–393. https://doi.org/10.1109/CLUSTER51413.2022.00049

Egele, R., Mohr, F., Viering, T., & Balaprakash, P. (2024). The unreasonable effectiveness of early discarding after one epoch in neural network hyperparameter optimization. *Neurocomputing*, 127964. https://doi.org/10.1016/j.neucom.2024.127964

Egelé, R., Guyon, I., Vishwanath, V., & Balaprakash, P. (2023). Asynchronous decentralized

Bayesian optimization for large scale hyperparameter optimization. *2023 IEEE 19th International Conference on e-Science (e-Science)*, 1–10. https://doi.org/10.1109/e-Science58273.2023.10254839

Égelé, R., Chang, T., Sun, Y., Vishwanath, V., & Balaprakash, P. (2023). Parallel multi-objective hyperparameter optimization with uniform normalization and bounded objectives. *arXiv Preprint arXiv:2309.14936*. https://doi.org/10.48550/arXiv.2309.14936

Égelé, R., Maulik, R., Raghavan, K., Lusch, B., Guyon, I., & Balaprakash, P. (2022). AutoDEUQ: Automated deep ensemble with uncertainty quantification. *26th International Conference on Pattern Recognition (ICPR)*, 1908–1914. https://doi.org/10.1109/ICPR56361.2022.9956231

Geurts, P., Ernst, D., & Zehenkel, L. (2006). Extremely randomized trees. *Machine Learning*. https://doi.org/10.1007/s10994-006-6226-1

Jiang, H., Shen, Y., Li, Y., Xu, B., Du, S., Zhang, W., Zhang, C., & Cui, B. (2024). OpenBox: A Python toolkit for generalized black-box optimization. *Journal of Machine Learning Research*, *25*(120), 1–11. https://dl.acm.org/doi/10.5555/3722577.3722697

Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Deng, D., Benjamins, C., Ruhkopf, T., Sass, R., & Hutter, F. (2022). SMAC3: A versatile Bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, *23*(54), 1–9. https://dl.acm.org/doi/10.5555/3586589.3586643

Nardi, L., Souza, A., Koeplinger, D., & Olukotun, K. (2019). HyperMapper: A practical design space exploration framework. *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 425–426. https://doi.org/10.1109/MASCOTS.2019.00053