The Journal of Open Source Software

Minion: A C++ and Python Library for Single-Objective Optimization

Khoirul Faiq Muzakka [1], **Sören Möller** [1], **and Martin Finsterbusch** [1]

1 Institute of Energy Materials and Devices (IMD-2), Forschungszentrum Jülich GmbH, Germany

DOI: 10.21105/joss.10145

Software

- Review
- Repository
- Archive

Editor: Abhishek Tiwari

Reviewers:

- @monacofj
- @anugrahjo

Submitted: 05 October 2025
Published: 09 November 2026

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

Summary

Optimization problems arise throughout science, engineering, and machine learning, including tasks such as simulation calibration, experimental data fitting, hyperparameter tuning, and design optimization. A common setting is single-objective optimization, where the goal is to minimize or maximize one performance measure. In many practical cases, the objective function is a black box: it can be evaluated at chosen points, but no closed-form expression, gradient, or other derivative information is available. To support optimization problems of this kind, we introduce Minion, a derivative-free optimization library for unconstrained and bound-constrained single-objective black-box problems. It is implemented in C++, with a Python interface available through MinionPy.

The library offers a centralized implementation of advanced derivative-free optimization algorithms, including state-of-the-art Differential Evolution (DE) methods that have performed strongly in IEEE Congress on Evolutionary Computation (CEC) competitions, Covariance Matrix Adaptation Evolution Strategy (CMA-ES) variants, and several Particle Swarm Optimization (PSO) variants. Alongside these research-grade solvers, Minion ships widely used optimizers such as Nelder–Mead, Dual Annealing (generalized simulated annealing), L-BFGS, and L-BFGS-B, allowing practitioners to combine established baselines with advanced heuristics within a single API. Many existing optimization libraries include only elementary variants of these methods and lack standardized benchmark problems. Minion addresses this by integrating multiple CEC benchmark suites (2011, 2014, 2017, 2019, 2020, and 2022) to facilitate algorithm evaluation and comparison.

Compared with widely adopted toolkits such as SciPy, NLopt, and pagmo2/pygmo, Minion emphasizes a unified interface for batch-evaluated objective functions, provides native support for modern population-based methods and finite-difference quasi-Newton solvers, and bundles curated CEC benchmark suites for reproducible experimentation. These design choices serve researchers developing bespoke algorithms as well as practitioners seeking robust defaults for black-box optimization.

State of the field

SciPy (Virtanen et al., 2020) underpins a large fraction of scientific computing in Python and offers a stable interface to classical optimization routines, including gradient-based methods and a handful of derivative-free heuristics such as Nelder–Mead and Powell's method. Its design prioritizes broad accessibility and numerical reliability; consequently, coverage of recent population-based metaheuristics or fully vectorized objective evaluations is limited.

NLopt (Johnson, 2007) collects an extensive set of local and global optimizers behind a C API with bindings to multiple languages. The library provides deterministic algorithms (e.g. COBYLA, BOBYQA) and stochastic search methods (e.g. CRS, ISRES, ESCH), yet relies

on single-sample objective calls. Users who require DE or PSO heuristics typically integrate third-party implementations alongside NLopt's core offerings.

pagmo2/pygmo (Biscani & Izzo, 2020) is geared towards island-based, massively parallel search. It excels at composing heterogeneous portfolios of solvers and supports sophisticated multi-objective workflows. For practitioners focused on single-objective, derivative-free problems, realizing a streamlined setup—particularly when benchmarking CEC-style test suites or coupling to noisy quasi-Newton updates—can involve additional configuration effort.

DEAP (Fortin et al., 2012) provides a highly extensible Python framework for constructing evolutionary algorithms from modular operators. This flexibility is valuable for exploratory research, but achieving high-throughput optimization requires users to supply their own performance-oriented backends, batched evaluation loops, and curated algorithm configurations.

Minion aims to complement these ecosystems by concentrating on single-objective optimization with built-in support for batch evaluation, finite-difference quasi-Newton solvers, modern DE, CMA-ES, and PSO variants, and integrated CEC benchmark suites. The goal is not to replace these libraries, but to offer an option tailored to scenarios where such capabilities are central requirements.

The decision to build Minion rather than contribute a single extension to an existing package follows from this scope. Adding one or two solvers to SciPy, NLopt, DEAP, or pagmo2 would not by itself provide a coherent benchmark platform, a shared batched-objective interface across both global and local solvers, and C++/Python access to a curated collection of recent CEC-oriented algorithms. Minion's scholarly contribution is therefore the integration of these components into a reproducible research tool: users can compare modern DE variants, CMA-ES variants, swarm-based methods, and classical local search algorithms, and optionally evaluate them on standardized CEC benchmark suites without changing objective-function conventions or stitching together multiple libraries.

Statement of need

Minion was created to address several limitations in existing optimization libraries:

1. Lack of a centralized library for advanced derivative-free optimization algorithms. Many optimization libraries lack a unified framework that combines modern DE variants, CMA-ES variants, PSO variants, and established local-search methods through a simple interface in both C++ and Python. While basic implementations of these algorithm families are common, recent research-grade variants such as L-SHADE, jSO, NL-SHADE-RSP, L-SRTDE, BIPOP-ACMAES, and DMS-PSO are often absent or distributed across separate packages. Minion fills this gap by providing these algorithms alongside a platform for researchers to create and test new optimizers.
2. Lack of centralized CEC benchmark support for reproducible comparison. Benchmarking derivative-free algorithms often requires separate implementations of CEC test suites, custom data files, and ad hoc scripts, making results harder to reproduce and compare. Minion integrates CEC benchmark suites from 2011, 2014, 2017, 2019, 2020, and 2022 under the same objective-function convention as user-defined problems, streamlining benchmarking and comparison with existing methods.
3. Limited support for straightforward batch evaluation. Several widely used libraries offer population-based optimizers, but their APIs typically accept one sample at a time, making it cumbersome to exploit highly parallel objective evaluations. Minion accepts batches natively so that vectorized or distributed functions can be used without additional wrappers.
4. Lack of robust L-BFGS-B and L-BFGS implementations that perform well under noise and support batch objective evaluations. Traditional finite-difference quasi-Newton

implementations can struggle with noisy function calls, which are common in real-world applications such as experimental data fitting. Minion mitigates this by using batched derivative evaluations together with a curvature-aware adaptive finite-difference step size and an optional Lanczos-style derivative estimate.

Software design

Minion is organized around a small C++ core interface rather than around algorithm-specific entry points. The user-facing `Minimizer` class acts as a factory and dispatcher: it normalizes an algorithm identifier, constructs the corresponding solver, and exposes execution through both `optimize()` and the call operator. All concrete optimizers inherit from `MinimizerBase`, which stores common state including the objective function, bounds, initial guesses, stopping tolerance, evaluation budget, random seed, callback, option map, current result, and optimization history. This structure keeps each algorithm implementation focused on its search logic while preserving uniform validation, configuration, and result reporting.

The central design choice is that objective functions are batched by default. In C++, a `MinionFunction` maps `std::vector<std::vector<double>>` candidate points to a vector of objective values, and the same convention is exposed in Python. This is less familiar than scalar objective calls, but it reflects the workload that motivated Minion: population-based algorithms, finite-difference gradients, noisy objectives, and benchmark campaigns often evaluate many candidates per iteration. A batched interface lets users vectorize evaluations in Python, run compiled objective kernels efficiently, or distribute objective calls without wrapping every solver separately.

The library balances this specialized interface with conventions that are familiar to scientific Python users. `MinionResult` follows the structure of `scipy.optimize.OptimizeResult`, with fields for the best point, objective value, iteration count, number of function evaluations, success flag, and diagnostic message. Solver options use a common map of typed values, and the `x0` argument accepts either one starting point or multiple candidate starts. This is useful when prior knowledge suggests more than one promising candidate solution. For population-based algorithms, the supplied points are inserted into the initial population and the remaining members are sampled from the bounds. For local methods, Minion evaluates the supplied candidates and starts from the best one. This design supports domain-specific initial guesses without making each algorithm expose a different API.

A second design choice is that Minion implements the performance-critical algorithms in C++ while providing a Python interface for accessibility. This avoids the overhead of pure-Python inner loops in large benchmark runs, but keeps installation and use close to standard Python workflows through MinionPy. The project also exposes C++ headers for native applications, including a single umbrella header and separate headers for individual solvers and CEC test functions. This split supports both exploratory notebooks and compiled scientific applications.

Finally, the benchmark code is treated as part of the software design, not as an external afterthought. CEC suites are bundled behind the same batched evaluation convention as user objectives, and example scripts store reproducible benchmark outputs for multiple algorithms and budgets. This matters because Minion is intended not only as an optimizer library, but also as an environment for evaluating new single-objective derivative-free methods against established baselines.

Algorithms

Minion currently implements the following optimization algorithms:

- Basic DE (Storn & Price, 1997)
- JADE (Zhang & Sanderson, 2009)

- L-SHADE (Tanabe & Fukunaga, 2014)
- L-SHADE-cnEpSin (Nashat H. Awad et al., 2017)
- jSO (Brest et al., 2017)
- j2020 (Brest et al., 2020)
- NL-SHADE-RSP (Stanovov et al., 2021)
- L-SRTDE (Stanovov & Semenkin, 2024)
- Adaptive Restart-Refine DE (ARRDE) (Muzakka, Shali, et al., 2026)
- Artificial Bee Colony (ABC) (Karaboga, 2005)
- Canonical PSO (Kennedy & Eberhart, 1995)
- SPSO-2011 (Zambrano-Bigiarini et al., 2013)
- Dynamic Multi-Swarm PSO (DMS-PSO) (Liang & Suganthan, 2005)
- CMA-ES (Hansen & Ostermeier, 1996)
- BI-population CMA-ES (BIPOP-ACMAES) (Hansen, 2009)
- Generalized Simulated Annealing (Dual Annealing) (Xiang et al., 1997)
- Nelder–Mead (Nelder & Mead, 1965)
- L-BFGS-B (Byrd et al., 1995)
- L-BFGS (Liu & Nocedal, 1989)

Additional algorithms are planned for future releases. Minion also ships benchmark suites from CEC competitions spanning 2011 (Das & Suganthan, 2011), 2014 (Liang et al., 2013), 2017 (N. H. Awad et al., 2016), 2019 (Price et al., 2018), 2020 (Yue et al., 2019), and 2022 (Kumar et al., 2021). The library further bundles classic analytic test functions—such as sphere, Rosenbrock, and Rastrigin—for quick experimentation and unit testing.

Minion's L-BFGS and L-BFGS-B implementations build on LBFGSpp (Qiu, 2013) but introduce derivative calculations tailored to noisy, vectorized workloads. Gradient estimates are generated from batched finite differences. The finite-difference step for each coordinate is adapted from an estimate of local curvature, obtained from L-BFGS curvature information accumulated in previous iterations, and from a multiplicative model of function noise. This follows the relation $h = 2\sqrt{\epsilon_f/|f^{(2)}|}$, where ϵ_f is the estimated function-value noise and $f^{(2)}$ is the local second derivative. Minion also supports a Lanczos-style derivative formula derived from least-squares fitting, while preserving a single batched objective call for the derivative evaluation. The accompanying benchmark notebook demonstrates the robustness of these quasi-Newton solvers on noisy CEC test suites while preserving a fully vectorized evaluation pipeline.

Research impact statement

Minion supports reproducible research in single-objective black-box optimization by combining advanced optimizers, batched objective evaluation, and standardized CEC benchmark suites in one C++/Python package. It also provides a practical platform for developing and evaluating new methods against strong baselines under consistent objective-function conventions, stopping criteria, and benchmark setups.

Minion is particularly relevant for optimization studies where objective evaluations are expensive, noisy, or naturally parallel, including simulation calibration, experimental data fitting, machine-learning hyperparameter tuning, and engineering design studies. It has already been used beyond its own benchmark examples, including in applied ion beam analysis software (Muzakka, Möller, Afzal, et al., 2026) and in the development and evaluation of new optimization methods (Muzakka, Shali, et al., 2026; Muzakka, Möller, & Finsterbusch, 2026).

The project also provides community-readiness signals and reusable research materials. MinionPy is packaged on PyPI, the source repository provides continuous-integration and test workflows, documentation is hosted on Read the Docs, and the software is archived with a Zenodo DOI (Muzakka et al., 2025). The repository includes C++ examples, Python examples, notebooks, benchmark comparison scripts, and stored CEC result files covering multiple algorithms, dimensions, and evaluation budgets. These materials make it possible

for users to reproduce benchmark-style studies, inspect optimizer behavior, and extend the software with new methods.

Availability and documentation

MinionPy can be installed from PyPI using `pip`, while the C++ library can be built from source. The source code is available at https://github.com/khoirulmuzakka/Minion. Documentation for both interfaces, including installation instructions, API references, and examples, is available at https://minion-py.readthedocs.io/.

AI usage disclosure

Generative AI tools, including OpenAI ChatGPT Codex (versions 5.0-5.4), were used to assist with drafting and language editing of this paper, generating some parts of the documentation, and coding assistance. The authors reviewed and edited the resulting text and code and remain responsible for the scientific claims, software description, citations, implementation, and final manuscript. Generative AI was not used as an autonomous contributor to the software implementation.

References

Awad, Nashat H., Ali, M. Z., & Suganthan, P. N. (2017). Ensemble sinusoidal differential covariance matrix adaptation with Euclidean neighborhood for solving CEC2017 benchmark problems. *2017 IEEE Congress on Evolutionary Computation (CEC)*, 372–379. https://doi.org/10.1109/CEC.2017.7969336

Awad, N. H., Ali, M. Z., Suganthan, P. N., Liang, J. J., & Qu, B. Y. (2016). *Problem Definitions and Evaluation Criteria for the CEC 2017 Special Session and Competition on Single Objective Real-Parameter Numerical Optimization*. Nanyang Technological University; Jordan University of Science and Technology; Zhengzhou University.

Biscani, F., & Izzo, D. (2020). A parallel global multiobjective framework for optimization: pagmo. *Journal of Open Source Software*, *5*(53), 2338. https://doi.org/10.21105/joss.02338

Brest, J., Maučec, M. S., & Bošković, B. (2020). Differential evolution algorithm for single objective bound-constrained optimization: Algorithm j2020. *2020 IEEE Congress on Evolutionary Computation (CEC)*, 1–8. https://doi.org/10.1109/CEC48606.2020.9185551

Brest, J., Maučec, M. S., & Bošković, B. (2017). Single objective real-parameter optimization: Algorithm jSO. *2017 IEEE Congress on Evolutionary Computation (CEC)*, 1311–1318. https://doi.org/10.1109/CEC.2017.7969456

Byrd, R. H., Lu, P., Nocedal, J., & Zhu, C. (1995). A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, *16*(5), 1190–1208. https://doi.org/10.1137/0916069

Das, S., & Suganthan, P. N. (2011). Problem Definitions and Evaluation Criteria for CEC 2011 Competition on Testing Evolutionary Algorithms on Real World Optimization Problems. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*. https://api.semanticscholar.org/CorpusID:14174334

Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., & Gagné, C. (2012). DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, *13*, 2171–2175.

Hansen, N. (2009). Benchmarking a BI-population CMA-ES on the BBOB-2009 function testbed. *Proceedings of the 11th Annual Conference Companion on Genetic*

and Evolutionary Computation Conference: Late Breaking Papers, 2389–2396. https://doi.org/10.1145/1570256.1570333

Hansen, N., & Ostermeier, A. (1996). Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. *Proceedings of IEEE International Conference on Evolutionary Computation*, 312–317. https://doi.org/10.1109/ICEC.1996.542381

Johnson, S. G. (2007). *The NLopt nonlinear-optimization package*. https://github.com/stevengj/nlopt.

Karaboga, D. (2005). *An idea based on honey bee swarm for numerical optimization* (Technical Report No. TR06). Erciyes University, Engineering Faculty, Computer Engineering Department. https://abc.erciyes.edu.tr/pub/tr06_2005.pdf

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. *Proceedings of ICNN'95 - International Conference on Neural Networks*, *4*, 1942–1948. https://doi.org/10.1109/ICNN.1995.488968

Kumar, A., Price, K. V., Mohamed, A. W., Hadi, A. A., & Suganthan, P. N. (2021). *Problem Definitions and Evaluation Criteria for the CEC 2022 Special Session and Competition on Single Objective Bound Constrained Numerical Optimization*. Indian Institute of Technology (BHU), Varanasi; Cairo University; The American University in Cairo; Arab Open University; Nanyang Technological University.

Liang, J. J., Qu, B. Y., & Suganthan, P. N. (2013). *Problem Definitions and Evaluation Criteria for the CEC 2014 Special Session and Competition on Single Objective Real-Parameter Numerical Optimization*. Zhengzhou University; Nanyang Technological University.

Liang, J. J., & Suganthan, P. N. (2005). Dynamic multi-swarm particle swarm optimizer. *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.*, 124–129. https://doi.org/10.1109/SIS.2005.1501611

Liu, D. C., & Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, *45*(1), 503–528. https://doi.org/10.1007/BF01589116

Muzakka, K. F., Möller, S., Afzal, S. A., Bram, M., & Finsterbusch, M. (2026). AutoNRA: A high-throughput parallel tool for self-consistent ion beam analysis. *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms*, *574*, 166076. https://doi.org/10.1016/j.nimb.2026.166076

Muzakka, K. F., Möller, S., & Finsterbusch, M. (2025). *Minion : A high-performance derivative-free optimization library designed for solving complex optimization problems.* (Version 0.1.9). Zenodo. https://doi.org/10.5281/zenodo.14893994

Muzakka, K. F., Möller, S., & Finsterbusch, M. (2026). *RCMAES: A robust CMA-ES variant for CEC2026 competition*. https://arxiv.org/abs/2604.27138

Muzakka, K. F., Shali, A. H., Suhendar, H., Möller, S., & Finsterbusch, M. (2026). *Robust differential evolution via nonlinear population size reduction and adaptive restart: The ARRDE algorithm*. https://arxiv.org/abs/2511.18429

Nelder, J. A., & Mead, R. (1965). A simplex method for function minimization. *The Computer Journal*, *7*(4), 308–313. https://doi.org/10.1093/comjnl/7.4.308

Price, K. V., Awad, N. H., Ali, M. Z., & Suganthan, P. N. (2018). *Problem Definitions and Evaluation Criteria for the 100-Digit Challenge Special Session and Competition on Single Objective Numerical Optimization*. Nanyang Technological University.

Qiu, Y. (2013). *LBFGSpp: C++ library for L-BFGS and L-BFGS-B*. https://github.com/yixuan/LBFGSpp.

Stanovov, V., Akhmedova, S., & Semenkin, E. (2021). NL-SHADE-RSP algorithm with adaptive archive and selective pressure for CEC 2021 numerical optimization. *2021 IEEE Congress on Evolutionary Computation (CEC)*, 809–816. https://doi.org/10.1109/CEC45853.2021.9504959

Stanovov, V., & Semenkin, E. (2024). Success rate-based adaptive differential evolution L-SRTDE for CEC 2024 competition. *2024 IEEE Congress on Evolutionary Computation (CEC)*, 1–8. https://doi.org/10.1109/CEC60901.2024.10611907

Storn, R., & Price, K. (1997). Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, *11*(4), 341–359. https://doi.org/10.1023/A:1008202821328

Tanabe, R., & Fukunaga, A. S. (2014). Improving the search performance of SHADE using linear population size reduction. *2014 IEEE Congress on Evolutionary Computation (CEC)*, 1658–1665. https://doi.org/10.1109/CEC.2014.6900380

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2

Xiang, Y., Sun, D. Y., Fan, W., & Gong, X. G. (1997). Generalized simulated annealing algorithm and its application to the Thomson model. *Physics Letters A*, *233*(3), 216–220. https://doi.org/10.1016/S0375-9601(97)00474-X

Yue, C. T., Price, K. V., Suganthan, P. N., Liang, J. J., Ali, M. Z., Qu, B. Y., Awad, N. H., & Biswas, P. P. (2019). *Problem Definitions and Evaluation Criteria for the CEC 2020 Special Session and Competition on Single Objective Bound Constrained Numerical Optimization*. Zhengzhou University; Nanyang Technological University.

Zambrano-Bigiarini, M., Clerc, M., & Rojas, R. (2013). Standard particle swarm optimisation 2011 at CEC-2013: A baseline for future PSO improvements. *2013 IEEE Congress on Evolutionary Computation*, 2337–2344. https://doi.org/10.1109/CEC.2013.6557848

Zhang, J., & Sanderson, A. C. (2009). JADE: Adaptive differential evolution with optional external archive. *IEEE Transactions on Evolutionary Computation*, *13*(5), 945–958. https://doi.org/10.1109/TEVC.2009.2014613