





libEnsemble: A complete Python toolkit for dynamic ensembles of calculations

Stephen Hudson ^{1*}, Jeffrey Larson ^{1*}, John-Luke Navarro ^{1*}, and Stefan M. Wild ^{2,3*}

¹ Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL, USA ² AMCR Division, Lawrence Berkeley National Laboratory, Berkeley, CA, USA ³ Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL, USA  Corresponding author * These authors contributed equally.

DOI: [10.21105/joss.06031](https://doi.org/10.21105/joss.06031)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Mehmet Hakan Satman](#) 



Reviewers:

- [@jsoishi](#)
- [@SergeyYakubov](#)

Submitted: 01 November 2023

Published: 18 December 2023

License

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

Summary

Almost all science and engineering applications eventually stop scaling: their runtime no longer decreases as available computational resources increase. Therefore, many applications will struggle to efficiently use emerging extreme-scale high-performance, parallel, and distributed systems. libEnsemble is a complete Python toolkit and workflow system for intelligently driving *ensembles* of experiments or simulations at massive scales. It enables and encourages multidisciplinary design, decision, and inference studies portably running on laptops, clusters, and supercomputers.

Statement of Need

While a growing number of packages are aimed at workflows, relatively few focus on running dynamic ensembles of calculations on clusters and supercomputers. Dynamic ensembles are workflows of computations that are defined and steered based on intermediate results. Examples include determining simulation parameters using numerical optimization methods, machine learning techniques, or statistical calibration tools. In each of these examples, the ensemble members are typically simulations that use different parameters or data. Additional examples of applications that have used libEnsemble are surveyed in the [Representative libEnsemble Use Cases](#) section below.

The target audience for libEnsemble includes scientists, engineers, and other researchers who stand to benefit from such dynamic workflows.

Key considerations for packages running dynamic ensembles include the following:

- Ease of use – whether the software requires a complex setup
- Portability – running on diverse machines with different schedulers, hardware, and communication modes (e.g., MPI runners) with minimal modification to user scripts
- Scalability – working efficiently with large-scale and/or many concurrent simulations
- Interoperability – the modularity of the package and the ability to interoperate with other packages
- Adaptive resource management – the ability to adjust resources given to each simulation throughout the ensemble
- Efficient resource utilization – including the ability to cancel simulations on the fly

libEnsemble seeks to satisfy the above criteria using a generator–simulator–allocator model. libEnsemble’s generators, simulators, and allocators – commonly referred to as user functions – are simply Python functions that accept and return NumPy ([Harris et al., 2020](#)) structured arrays. Generators produce input for simulators, simulators evaluate those inputs, and allocators decide whether and when a simulator or generator should be called; any level of complexity is supported. Multiple concurrent instances (an “ensemble”) of user functions are coordinated by libEnsemble’s worker processes. Workers are typically assigned/reassigned compute resources; within user functions, workers can launch applications, evaluate intermediate results, and communicate via the manager.

Related Work

Other packages for managing workflows and ensembles include Colmena ([Ward et al., 2021](#)) and the RADICAL-Ensemble Toolkit ([Balasubramanian et al., 2016](#)) as well as packages such as Parsl ([Babuji et al., 2019](#)) and Balsam ([Salim et al., 2019](#)) that provide backend dispatch and execution.

libEnsemble’s unique generator–simulator–allocator paradigm eliminates the need for users to explicitly define task dependencies. Instead, it emphasizes data dependencies between these customizable Python user functions. This modular design also lends itself to exploiting the large library of example user functions provided with libEnsemble or available from the community (e.g., libEnsemble Community ([2023](#))), maximizing code reuse. For instance, users can readily choose an existing generator function and tailor a simulator function to their particular needs.

libEnsemble takes the philosophy of minimizing required dependencies while supporting various backend mechanisms when needed. In contrast to other packages that cover only a subset of such a workflow, libEnsemble is a complete toolkit that includes generator-in-the-loop and backend mechanisms. For example, Colmena uses frontend components to create and coordinate tasks while using Parsl to dispatch simulations.

For example, the vast majority of current use cases do not require a database or special runtime environment. For use cases that have such requirements, Balsam can be used on the backend by substituting the regular MPI executor for the Balsam executor. This approach simplifies the user experience and reduces the initial setup and adoption costs when using libEnsemble.

libEnsemble Functionality

libEnsemble communicates between a manager and multiple workers using either Python’s built-in multiprocessing, MPI (via mpi4py ([Dalcín et al., 2008](#))), or TCP.

To achieve portability, libEnsemble detects runtime setup information not commonly detected by other packages: It detects crucial system information such as scheduler details, MPI runners, core counts, and GPU counts (for different types of GPUs) and uses these to produce run-lines and GPU settings for these systems, without the user having to alter scripts. For example, on a system that uses Slurm’s `srun`, libEnsemble will use `srun` options to assign GPUs, while on other systems it will assign GPUs via the appropriate environment variables such as `ROCR_VISIBLE_DEVICES` or `CUDA_VISIBLE_DEVICES`, allowing the user to simply state the number of GPUs needed for each simulation. For cases where autodetection is insufficient, the user can supply platform information or the name of a known system via scripts or an environment variable. This makes it simple to transfer user scripts between platforms.

By default, libEnsemble equally divides available compute resources among workers. When simulation parameters are created, however, the number of processes and GPUs can also be specified for each simulation. The close coupling between the libEnsemble generators and simulators enables a generator to perform tasks such as asynchronously receiving results, updating models, and canceling previously initiated simulations. Simulations that are already running can be terminated and resources recovered. This approach is more flexible compared

with other packages, where the generation of simulations is external to the dispatch of a batch of simulations.

libEnsemble also supports “persistent user functions” that maintain their state while running on workers. This prevents the need to store and reload data as is done by other ensemble packages that support only a fire-and-forget approach to ensemble components.

Representative libEnsemble Use Cases

Examples of libEnsemble applications in science and engineering include the following:

- Optimization of variational algorithms on quantum computers (Liu et al., 2022)
- Parallelization of the ParMOO solver for multiobjective simulation optimization problems (Chang & Wild, 2023)
- Design of particle accelerators (A. Ferran Pousa et al., 2022; A. Ferran Pousa et al., 2023; Neveu et al., 2023)
- Sequential Bayesian experimental design (Sürer et al., 2023) and Bayesian calibration (Chan et al., 2023)

A selection of community-provided libEnsemble functions and workflows that users can build off is maintained in libEnsemble Community (2023).

Additional details on the parallel features and scalability of libEnsemble can be found in Hudson et al. (2022) and Hudson et al. (2023).

Acknowledgments

We acknowledge contributions from David Bindel. This article was supported in part by the PETSc/TAO activity within the U.S. Department of Energy’s (DOE’s) Exascale Computing Project (17-SC-20-SC) and by the CAMPA, ComPASS, and NUCLEI SciDAC projects within DOE’s Office of Science, Advanced Scientific Computing Research under contract numbers DE-AC02-06CH11357 and DE-AC02-05CH11231.

References

- Babuji, Y., Woodard, A., Li, Z., Katz, D. S., Clifford, B., Kumar, R., Lacinski, L., Chard, R., Wozniak, J., Foster, I., Wilde, M., & Chard, K. (2019). Parsl: Pervasive parallel programming in Python. *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*. <https://doi.org/10.1145/3307681.3325400>
- Balasubramanian, V., Treikalis, A., Weidner, O., & Jha, S. (2016). Ensemble toolkit: Scalable and flexible execution of ensembles of tasks. *International Conference on Parallel Processing*, 458–463. <https://doi.org/10.1109/ICPP.2016.59>
- Chan, M. Y.-H., Plumlee, M., & Wild, S. M. (2023). Constructing a simulation surrogate with partially observed output. *Technometrics*. <https://doi.org/10.1080/00401706.2023.2210170>
- Chang, T. H., & Wild, S. M. (2023). *Designing a framework for solving multiobjective simulation optimization problems* (No. 2304.06881). arXiv. <https://doi.org/10.48550/arXiv.2304.06881>
- Dalcín, L., Paz, R., Storti, M., & D’Elía, J. (2008). MPI for Python: Performance improvements and MPI-2 extensions. *Journal of Parallel and Distributed Computing*, 68(5), 655–662. <https://doi.org/10.1016/j.jpdc.2007.09.005>

- Ferran Pousa, A., Jalas, S., Kirchen, M., Martinez de la Ossa, A., Thévenet, M., Hudson, S., Larson, J., Huebl, A., Vay, J.-L., & Lehe, R. (2022). Multitask optimization of laser-plasma accelerators using simulation codes with different fidelities. *Proceedings of the 13th International Particle Accelerator Conference*, 13, 1761–1764. <https://doi.org/10.18429/JACoW-IPAC2022-WEPOST030>
- Ferran Pousa, A., Jalas, S., Kirchen, M., Martinez de la Ossa, A., Thévenet, M., Hudson, S., Larson, J., Huebl, A., Vay, J.-L., & Lehe, R. (2023). Bayesian optimization of laser-plasma accelerators assisted by reduced physical models. *Physical Review Accelerators and Beams*, 26, 084601. <https://doi.org/10.1103/PhysRevAccelBeams.26.084601>
- Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., ... Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825), 357–362. <https://doi.org/10.1038/s41586-020-2649-2>
- Hudson, S., Larson, J., Navarro, J.-L., & Wild, S. M. (2022). libEnsemble: A library to coordinate the concurrent evaluation of dynamic ensembles of calculations. *IEEE Transactions on Parallel and Distributed Systems*, 33(4), 977–988. <https://doi.org/10.1109/TPDS.2021.3082815>
- Hudson, S., Larson, J., Wild, S. M., Bindel, D., & Navarro, J.-L. (2023). *libEnsemble user manual, version 1.0.0* [Tech report]. Argonne National Laboratory. <https://libensemble.readthedocs.io>
- libEnsemble Community. (2023). *A selection of libEnsemble functions and complete workflows from the community*. <https://github.com/Libensemble/libe-community-examples>
- Liu, X., Angone, A., Shaydulin, R., Safro, I., Alexeev, Y., & Cincio, L. (2022). Layer VQE: A variational approach for combinatorial optimization on noisy quantum computers. *IEEE Transactions on Quantum Engineering*, 3, 1–20. <https://doi.org/10.1109/TQE.2021.3140190>
- Neveu, N., Chang, T. H., Franz, P., Hudson, S., & Larson, J. (2023). Comparison of multiobjective optimization methods for the LCLS-II photoinjector. *Computer Physics Communications*, 283, 108566. <https://doi.org/10.1016/j.cpc.2022.108566>
- Salim, M., Uram, T., Childers, J. T., Vishwanath, V., & Papka, M. (2019). Balsam: Near real-time experimental data analysis on supercomputers. *1st Annual Workshop on Large-Scale Experiment-in-the-Loop Computing*. <https://doi.org/10.1109/xloop49562.2019.00010>
- Sürer, Ö., Plumlee, M., & Wild, S. M. (2023). Sequential Bayesian experimental design for calibration of expensive simulation models. *Technometrics*. <https://doi.org/10.1080/00401706.2023.2246157>
- Ward, L., Sivaraman, G., Pauloski, J. G., Babuji, Y., Chard, R., Dandu, N., Redfern, P. C., Assary, R. S., Chard, K., Curtiss, L. A., Thakur, R., & Foster, I. (2021). Colmena: Scalable machine-learning-based steering of ensemble simulations for high performance computing. *Workshop on Machine Learning in High Performance Computing Environments*, 9–20. <https://doi.org/10.1109/MLHPC54614.2021.00007>