The Journal of Open Source Software

Pydisort: A Modern Python Package for Parallelized Discrete Ordinates Radiative Transfer (DISORT)

Zoey Hu [1] **and Cheng Li** [1]

1 University of Michigan, Ann Arbor, MI, USA

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 02 May 2025
Published: unpublished

Summary

Radiative transfer is the study of how electromagnetic radiation travels through and interacts with a medium—such as Earth's atmosphere, ocean waters, or a planetary atmosphere. It is a fundamental process that governs how light and thermal radiation are absorbed, scattered, and emitted by gases, aerosols, clouds, and surfaces. Understanding radiative transfer is crucial for interpreting remote sensing data from satellites and spacecrafts, modeling climate systems, and studying planetary atmospheres.

DISORT (Discrete Ordinates Radiative Transfer) is a widely-used algorithm that abstracts the medium as a plane-parallel structure, with the horizontal properties uniform but possibly varying vertically. It solves the radiative transfer equation using the discrete ordinates method, which discretizes the angular domain into a finite number of directions and solves the resulting system of equations. Among the three main components of radiative transfer — absorption, scattering, and emission — scattering is often the most complex and computationally intensive to model, as it involves integral from all incoming angles to all outgoing angles. In a realistic atmosphere with variable scattering, absorption, multiple layers, and possibly multiple wavelengths, in cases where the radiance is sensitive to small-scale details, or when it's necessary to obtain high accuracy, the scalability and efficiency of the radiative transfer model become critical.

The original DISORT algorithm was implemented in Fortran (Stamnes et al., 1988). It has been widely used in Earth and planetary atmospheres (Clough et al., 2005; Komacek et al., 2022; Lee, 2024; Li et al., 2018; Tan & Showman, 2021; Zhang et al., 2015). However, the original Fortran implementation is limited by static memory allocation, which requires the user to specify the number of atmospheric layers and radiation streams at compile time. This limitation persists even with efforts to wrap the original Fortran code via `f2py` for more user-friendly access (e.g. chanGimeno's, SeregaOsipov's, danielkoll's, mjwolff's). Another approach is to implement the DISORT algorithm in pure Python, which solves the problem of static memory allocation but is less efficient than the Fortran implementation (Ho, 2024).

An endeavor to overcome the limitations of the original Fortran implementation with even better efficiency is the `cdisort` library, which is a C reimplementation of DISORT(Buras et al., 2011), and it has been an integral component of the `libRadtran` radiative transfer software package (Emde et al., 2016). It preserves its stable, well-tested numerical core while adding portability and ease of embedding in modern workflows. A key advancement over the Fortran version is the adoption of dynamic memory allocation, eliminating the need for compile-time parameters. It's a 100% double-precision package that steers away from the occasional numerical instabilities caused by the mixed-precision calculations in the original Fortran code. Additionally, for large problem sizes, `cdisort` achieves over a 20× speedup through an improved intensity correction method and by leveraging the operating system's background memory-zeroing instead of the explicit nested loop structure used in Fortran DISORT for initializing multidimensional arrays.

The cdisort library, while powerful, remains a single-threaded, low-level implementation that lacks a modern interface to exploit today's parallel-computing advances — GPU acceleration, heterogeneous architectures, or seamless integration with machine-learning frameworks. More critically, cdisort is not a GitHub-hosted easy-to-contribute library and is not easily accessible to the broader scientific community, particularly those more comfortable with Python than C and eager to utilize modern tools like PyTorch.

To provide the radiative transfer community with a truly modern, high-performance, and user-friendly solution, we developed Pydisort, a pip-installable, compile-free Python package that wraps the cdisort library with support for parallel processing and compatibility with machine learning frameworks via PyTorch. Pydisort is designed for ease of use, efficiency, and flexibility, enabling users to perform radiative transfer calculations in a plane-parallel atmosphere with minimal effort. It supports batch processing, allowing efficient computation of radiative properties across multiple atmospheric layers and wavelengths in a single invocation.

Specifically, Pydisort provides the following features:

1. We use the cdisort library as the backend, which allows for dynamic memory allocation during runtime.
2. We use PyTorch (Paszke, 2019)'s tensor data structure for memory management, paving the way for heterogeneous computing and automatic parallelization.
3. We create an intermediate C++ layer to the backend C-library. The intermediate layer handles pre- and post-processing of the raw cdisort data structures, eliminating the need to feed long parameter lists to the backend library.
4. We leverage Pybind11 (Jakob, 2024) to interface between Python API calls and our intermediate C++ interface. Pybind11 is a header-only modern alternative to f2py with graceful type-handling, casting and memory management.
5. We design software architecture to support building the C/C++ backend libraries as shared libraries, linking them to libtorch and Python, and distributing them as a pip-installable PyPI package pydisort for various platforms, including Linux and MacOS.
6. We automate the building and distribution process using cibuildwheel on Mac images and Linux images with glibc 2.28+, which is the minimum version of glibc required by PyTorch v2.7+.
7. We dynamically determine the CXX11_ABI version from the upstream libtorch library. Currently with libtorch v2.7, the CXX11_ABI version is 1 for Linux distributions and 0 for MacOS distributions.
8. We provide two frontends: (1) a Python interface and (2) a C++ interface. The former is useful for users who want to use the package in Python and take advantage of the machine learning capabilities enabled by PyTorch, while the latter is useful for users who want to integrate the package into their own C/C++ packages.
9. We automate the Continuous Integration (CI) and Continuous Distribution (CD) processes using GitHub Actions to ensure that minimal human effort is required for developers contributing to and maintaining the package.
10. We adhere to the PEP 8 (Van Rossum et al., 2001) style guide for Python code, making the program a Python-first experience. The function calls make frequent use of Python features such as keyword arguments and named arguments, which are idiomatic to Python users.

Statement of need

Tools for calculating atmospheric radiative transfer are essential for a wide range of applications in atmospheric modeling and remote sensing. As demonstrated in the Summary section, the original Fortran implementation of DISORT has been widely used in research. To accommodate the growing popularity of Python in the scientific community, many groups have developed their own Python wrappers for the original Fortran implementation of DISORT. However, few have reached the same level of maturity, efficiency, and usability as our package.

Our package manages to distribute `pip`-installable pre-built library that wraps the DISORT algorithm implemented in a lower-level compiled language. Our package

1. saves users from the hassle of setting up the build environment and the tool chains;
2. skips the compilation process because we ship the compiled shared libraries with the package on PyPI;
3. achieves at least the same level of performance as the original `C/Fortran` implementation on the same hardware, with the extended capability of GPU acceleration in the future;
4. provides a modern `Python` interface that is easy to use and understand;
5. offers comprehensive documentation and user support to facilitate adoption and integration into existing workflows.

The previous effort of educational purpose has been made by Ho (2024), which is a pure `Python` implementation of the DISORT algorithm and Ricchiazzi et al. (1998), which is a `Fortran77` implementation. Since our backend subroutines are implemented in `C`, our code runs at least as fast as the original `Fortran` implementation in single-threaded mode, while the `Python` interface lowers the barrier to entry for users who are not familiar with `Fortran` or `C`.

Additionally, `Pydisort` improves upon the `C/Fortran` implementations by enabling parallelization over wavelengths and columns. For plane-parallel atmospheres, the radiative transfer equation is separable by columns and by wavelengths/wavenumbers. However, looping over columns and wavelengths in `Python` will significantly slow down the code. Therefore, we leverage the `PyTorch` tensor data structure and its `TensorIterator` functionality to enable concurrent execution over columns and wavelengths. This design achieves strong scaling performance on modern multi-core CPUs. Future improvements of the package will seamlessly allow GPU acceleration of the radiative transfer calculation when the original `C` backend is adapted to execute on GPUs.

Performance Evaluation

To quantify the efficiency of `Pydisort` relative to existing implementations, we benchmarked runtimes using a test driver based on Test Problem 9 ("General Emitting/Absorbing/Scattering") from the DISORT suite (`diotest`), with increased computational workload. Specifically, the benchmark configuration includes 32 streams and 100 atmospheric layers, typical for atmospheric radiation calculations; all other parameters are kept consistent with the original test case. We increase the workload by solving the radiative transfer equations repetitively for multiple wavelengths or columns, representing a realistic application case in remote sensing and atmospheric modeling such that multiple spectral bands or atmospheric columns are processed simultaneously.

We evaluated `Pydisort` performance against two alternative implementations:

- **`cdisort`**: the original C reimplementation of DISORT, which serves as our baseline performance reference,
- **`PythonicDISORT`**: a pure-Python translation of DISORT, used to highlight the performance gap between interpreted and optimized implementations.

To assess parallel performance, we ran `Pydisort` in both single-threaded and multi-threaded modes using `PyTorch`'s internal parallelism configuration (`torch.set_num_threads`). Multi-threaded tests were conducted with 10 threads to illustrate scalability on modern CPU architectures. All tests were conducted on a MacBook Pro equipped with an Apple M1 Max chip (10-core CPU, peak clock speed of 3.22 GHz) and 64 GB of RAM.

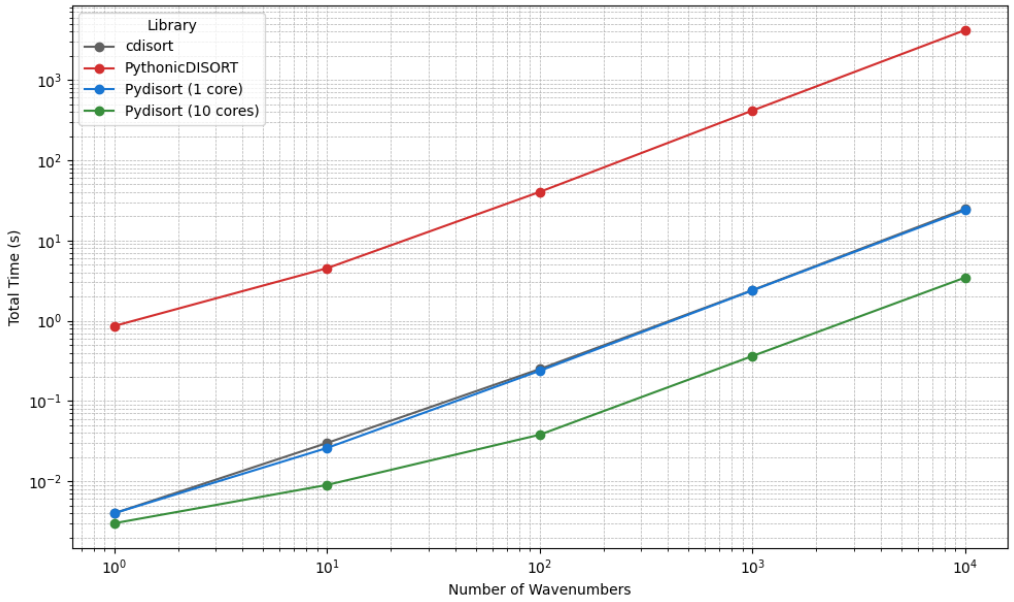

Figure 1: Runtime comparison of radiative transfer implementations as a function of the number of wavenumbers, evaluated on a fixed test problem (DISORT Test 09 with 32 streams and 100 layers). `cdisort` (gray) serves as the baseline C implementation. `PythonicDisort` (red) is a pure-Python reimplementation. `Pydisort` is shown in both single-threaded (blue, 1 core) and multi-threaded (green, 10 cores) modes, using `PyTorch`-based parallelism. Both axes use logarithmic scaling. The results demonstrate that `Pydisort` outperforms `cdisort` in runtime efficiency and exhibits strong scaling performance with increasing spectral resolution.

We did not benchmark the original Fortran DISORT implementation, as prior work by Buras et al. (2011) has demonstrated that the C version (`cdisort`) provides better runtime performance and is commonly used in research settings as part of the `libRadtran` package (Emde et al., 2016).

Figure 1 summarizes the results, showing runtime as a function of the number of wavenumbers. Both axes use logarithmic scaling to highlight relative performance trends across a wide range of spectral resolutions. As shown, `Pydisort` is at least as efficient as `cdisort` when using a single core, and outperforms `cdisort` in multi-threaded mode by an order of magnitude when the workload increases, demonstrating the effectiveness of `PyTorch`'s parallelism for this problem. The pure-Python implementation (`PythonicDISORT`) is significantly slower, highlighting the performance benefits of using a compiled backend. Additionally, `Pydisort` shows strong scaling performance, demonstrating its suitability for high-throughput radiative transfer workloads.

Acknowledgements

We acknowledge Dr. Timothy E. Dowling for his work on migrating the original FORTRAN version of DISORT to C, which is the basis for our implementation. We acknowledge Dr. Xi Zhang and Dr. Tianhao Le for initiating the project and testing the code. We also thank Andrew Ryan for early testing and feedback on the package.

References

Buras, R., Dowling, T., & Emde, C. (2011). New secondary-scattering correction in DISORT with increased efficiency for forward scattering. *Journal of Quantitative Spectroscopy and Radiative Transfer*, *112*(12), 2028–2034. https://doi.org/10.1016/j.jqsrt.2011.03.019

Clough, S. A., Shephard, M. W., Mlawer, E. J., Delamere, J., Iacono, M. J., Cady-Pereira, K., Boukabara, S., & Brown, P. D. (2005). Atmospheric radiative transfer modeling: A summary of the AER codes. *Journal of Quantitative Spectroscopy and Radiative Transfer*, *91*(2), 233–244. https://doi.org/10.1016/j.jqsrt.2004.05.058

Emde, C., Buras-Schnell, R., Kylling, A., Mayer, B., Gasteiger, J., Hamann, U., Kylling, J., Richter, B., Pause, C., Dowling, T., & others. (2016). The libRadtran software package for radiative transfer calculations (version 2.0. 1). *Geoscientific Model Development*, *9*(5), 1647–1672. https://doi.org/10.5194/gmd-9-1647-2016

Ho, D. J. (2024). PythonicDISORT: A python reimplementation of the discrete ordinate radiative transfer package DISORT. *Journal of Open Source Software*, *9*(103), 6442. https://doi.org/10.21105/joss.06442

Jakob, W. (2024). *pybind11 documentation*. https://pybind11.readthedocs.io/en/stable/

Komacek, T. D., Tan, X., Gao, P., & Lee, E. K. (2022). Patchy nightside clouds on ultra-hot jupiters: General circulation model simulations with radiatively active cloud tracers. *The Astrophysical Journal*, *934*(1), 79. https://doi.org/10.3847/1538-4357/ac7723

Lee, E. K. (2024). Testing approximate infrared scattering radiative-transfer methods for hot jupiter atmospheres. *The Astrophysical Journal*, *965*(2), 115. https://doi.org/10.3847/1538-4357/ad2e8e

Li, C., Le, T., Zhang, X., & Yung, Y. L. (2018). A high-performance atmospheric radiation package: With applications to the radiative energy budgets of giant planets. *Journal of Quantitative Spectroscopy and Radiative Transfer*, *217*, 353–362. https://doi.org/10.1016/j.jqsrt.2018.06.002

Paszke, A. (2019). Pytorch: An imperative style, high-performance deep learning library. *arXiv Preprint arXiv:1912.01703*. https://doi.org/10.48550/arXiv.1912.01703

Ricchiazzi, P., Yang, S., Gautier, C., & Sowle, D. (1998). SBDART: A research and teaching software tool for plane-parallel radiative transfer in the earth's atmosphere. *Bulletin of the American Meteorological Society*, *79*(10), 2101–2114. https://doi.org/10.1175/1520-0477(1998)079%3C2101:SARATS%3E2.0.CO;2

Stamnes, K., Tsay, S.-C., Wiscombe, W., & Jayaweera, K. (1988). Numerically stable algorithm for discrete-ordinate-method radiative transfer in multiple scattering and emitting layered media. *Applied Optics*, *27*(12), 2502–2509. https://doi.org/10.1364/AO.27.002502

Tan, X., & Showman, A. P. (2021). Atmospheric circulation of brown dwarfs and directly imaged exoplanets driven by cloud radiative feedback: Global and equatorial dynamics. *Monthly Notices of the Royal Astronomical Society*, *502*(2), 2198–2219. https://doi.org/10.1093/mnras/stab060

Van Rossum, G., Warsaw, B., & Coghlan, N. (2001). PEP 8–style guide for python code. *Python.org*, *1565*, 28. https://www.python.org/dev/peps/pep-0008/

Zhang, X., West, R. A., Irwin, P. G., Nixon, C. A., & Yung, Y. L. (2015). Aerosol influence on energy balance of the middle atmosphere of jupiter. *Nature Communications*, *6*(1), 10231. https://doi.org/10.1038/ncomms10231