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**Published:** unpublished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 describes how electromagnetic radiation travels through and interacts with a medium, such as Earth's atmosphere, ocean waters, or planetary atmospheres. It is a fundamental process that governs how light and thermal radiation are absorbed, scattered, and emitted by gases, aerosols, clouds, and surfaces, and is foundational to climate modeling, remote sensing, and planetary science. Accurate and efficient radiative transfer solvers are therefore essential for interpreting satellite observations, simulating climate systems, and studying atmospheres across the Solar System and beyond.

The Discrete Ordinates Radiative Transfer (DISORT) algorithm is one of the most widely used numerical approaches for solving the radiative transfer equation in plane-parallel media. DISORT discretizes the angular domain into a finite set of propagation directions and solves the resulting coupled system of equations. Among the core radiative processes — absorption, emission, and scattering — scattering is often the most computationally demanding, as it requires integrating contributions from all incoming angles to all outgoing angles. For realistic atmospheres with many vertical layers, strong scattering, and multiple wavelengths, radiative transfer calculations can become a dominant computational cost.

`Pydisort` is a modern, high-performance Python package for plane-parallel radiative transfer based on DISORT. It provides a user-friendly, `pip`-installable interface while retaining the numerical robustness of established DISORT implementations. By combining a compiled C backend with a `PyTorch`-based tensor interface, `Pydisort` enables efficient batch processing, parallel execution over wavelengths and atmospheric columns, and seamless integration with contemporary scientific and machine-learning workflows.

## Statement of Need

Radiative transfer models based on DISORT are essential tools in atmospheric science, climate modeling, remote sensing, and planetary physics. The original DISORT implementation, written in Fortran (Stamnes et al., 1988), has been widely adopted and validated across decades of Earth and planetary studies (Clough et al., 2005; Komacek et al., 2022; Lee, 2024; Li et al., 2018; Tan & Showman, 2021; Zhang et al., 2015). However, this implementation relies on static memory allocation, requiring users to specify the number of atmospheric layers and radiation streams at compile time, which limits flexibility and complicates modern workflows.

To address usability concerns, several Python wrappers around the original Fortran DISORT have been developed using tools such as `f2py` (e.g., pyDISORT variants by chanGimeno, SeregaOsipov, danielkoll, and mjwolff). While these efforts improve accessibility, they inherit the underlying static-memory constraints and impose nontrivial build and compilation requirements. An alternative approach is a pure-Python reimplementation of DISORT (Ho, 2024), which

removes compilation barriers but sacrifices computational performance, making it unsuitable for large-scale or high-throughput applications.

`Pydisort` addresses these limitations by providing a precompiled, `pip`-installable package that wraps a high-performance DISORT implementation written in a compiled language. Specifically, it

1. eliminates the need for users to configure complex build environments or toolchains;
2. avoids local compilation by distributing prebuilt shared libraries via PyPI;
3. achieves performance exceeding existing C/Fortran implementations on modern devices, with a clear path toward GPU acceleration;
4. exposes a modern, idiomatic Python interface suitable for interactive use and scripting;
5. integrates naturally with `PyTorch`-based scientific and machine-learning workflows.

The target audience includes atmospheric scientists, planetary scientists, climate modelers, and remote-sensing researchers who require accurate radiative transfer calculations but prefer Python-based ecosystems and scalable computational tools.

## State of the Field

Several established tools implement or build upon DISORT for radiative transfer calculations. The original Fortran DISORT (Stamnes et al., 1988) remains a reference implementation but is constrained by static memory allocation and legacy language choices. The `cdisort` library (Buras et al., 2011), a C reimplementation of DISORT, represents a significant advance: it introduces dynamic memory allocation, uses consistent double-precision arithmetic to avoid numerical instabilities present in mixed-precision Fortran code, and achieves substantial performance gains through improved intensity correction methods and more efficient memory initialization. `cdisort` is widely used as a core component of the `libRadtran` radiative transfer package (Emde et al., 2016).

Despite its numerical strengths, `cdisort` remains a low-level, single-threaded library that is not easily accessible to the broader scientific community. It lacks a modern Python interface, is not distributed as a standalone, community-oriented GitHub project, and does not integrate naturally with parallel computing frameworks or machine-learning ecosystems. As a result, many users face a "build vs. usability" trade-off: relying on performant but inaccessible compiled code, or using slower but more user-friendly Python implementations.

`Pydisort` occupies a distinct position in this landscape. Rather than reimplementing DISORT yet again, it builds directly on the well-tested `cdisort` numerical core while addressing its accessibility and scalability limitations. By wrapping `cdisort` in a C++/Python interface and adopting `PyTorch` tensors as the primary data structure, `Pydisort` enables parallel execution over wavelengths and atmospheric columns — dimensions that are naturally separable in plane-parallel radiative transfer problems. This design allows `Pydisort` to outperform existing single-threaded C implementations on modern multi-core CPUs, while remaining significantly faster than pure-Python alternatives.

In short, `Pydisort` provides a clear scholarly contribution by combining numerical fidelity, modern software design, and scalable performance in a form that existing DISORT-based tools do not offer. It fills a gap between legacy compiled solvers and emerging Python-centric scientific workflows, making high-accuracy radiative transfer calculations more accessible and extensible for contemporary research.

## Software Design

`Pydisort` is designed as a modular, layered software system that bridges a high-performance radiative transfer backend with modern, Python-centric scientific workflows. The core numerical

solver is provided by the `cdisort` library, which serves as the computational backend and supports dynamic memory allocation at runtime. This choice enables flexible problem sizes while retaining the numerical robustness of the established DISORT implementation.

To facilitate efficient memory management and future-proof the codebase for heterogeneous computing, `Pydisort` adopts `PyTorch` (Paszke, 2019) tensors as its primary data structure at the user interface level. Using tensors allows seamless integration with `PyTorch`'s automatic parallelization, GPU acceleration, and machine-learning ecosystem, while maintaining compatibility with CPU-only environments.

An intermediate C++ layer is introduced between the Python interface and the C backend. This layer encapsulates the raw `cdisort` data structures and is responsible for pre-processing inputs and post-processing outputs. By centralizing this logic, the design avoids exposing users to long, error-prone parameter lists and isolates backend-specific details from the public API. This intermediate layer also enables reuse of the backend functionality in non-Python contexts.

The Python bindings are implemented using `pybind11` (Jakob, 2024), a modern, header-only C++ library that provides robust type conversion, memory ownership semantics, and exception handling. Compared to traditional approaches such as `f2py`, `pybind11` offers greater flexibility and maintainability for mixed C++/Python codebases.

The build system is designed to produce shared C and C++ libraries that link against both `libtorch` and Python, allowing the package to be distributed as a standard `pip`-installable PyPI package (`pydisort`). Prebuilt binary wheels are provided for Linux and macOS platforms. The build and distribution process is fully automated using `cibuildwheel`, targeting Linux systems with `glibc` version 2.28 or newer, which is the minimum required by `PyTorch` v2.7 and later.

To ensure ABI compatibility with upstream `PyTorch` binaries, the build system dynamically determines the appropriate `CXX11_ABI` setting from the linked `libtorch` distribution. For `libtorch` v2.7, this corresponds to `CXX11_ABI=1` on Linux and `CXX11_ABI=0` on macOS.

`Pydisort` provides two user-facing interfaces: a Python API and a C++ API. The Python interface is intended for interactive use, scripting, and integration with machine-learning workflows, while the C++ interface supports embedding `Pydisort` directly into larger C or C++ simulation frameworks.

Continuous integration (CI) and continuous distribution (CD) are handled through GitHub Actions, enabling automated testing, building, and release of binary wheels with minimal manual intervention. The Python codebase follows the PEP 8 style guide (Van Rossum et al., 2001) and adopts a Python-first design philosophy, making extensive use of keyword arguments and named parameters to provide a clear, idiomatic user experience.

## 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.

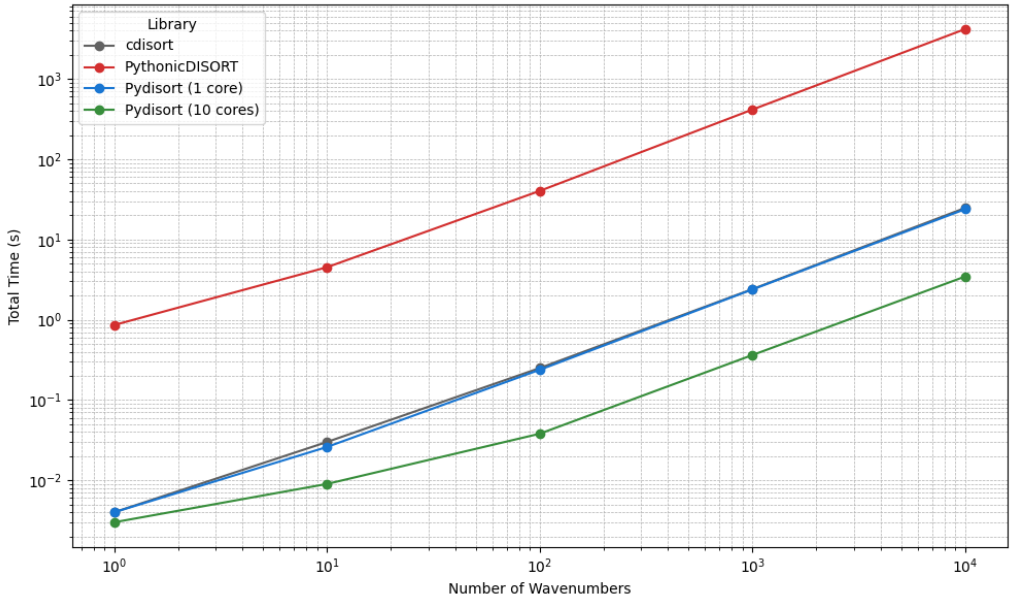


**Figure 1:** Runtime comparison of radiative transfer implementations as a function of the number of wavenumbers, evaluated on a fixed test problem (DISORT Test 09 with 32 streams and 100 layers). `cdisort` (gray) serves as the baseline C implementation. `PythonicDisort` (red) is a pure-Python reimplementation. `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.

## Research Impact Statement

`Pydisort` solves the radiative transfer equation in an efficient and user-friendly manner, addressing a long-standing need in atmospheric science, climate modeling, and planetary

science for a modern, high-performance DISORT-based tool. It has been adopted as a core computation component within a growing ecosystem of research projects, supporting studies that span terrestrial atmospheres, giant planets, exoplanets, finite volum models for compressible fluids, coupled chemistry-thermodynamics frameworks, and physically based rendering in vision and imaging applications. It's currently used in dozens of GitHub repositories across multiple institutions, for both research and educational purposes.

By integrating with `PyTorch`'s tensor and parallel computing infrastructure, `Pydisort` can be deployed in modern high-performance computing environments, enabling large-scale simulations and high-throughput data analysis that were difficult to achieve with traditional DISORT implementations. Its native compatibility with machine-learning frameworks like `PyTorch` enables researchers to integrate radiative transfer calculations into data-driven models. Adoption metrics further demonstrates `Pydisort`'s impact. The PyPI public dataset shows a marked increase in downloads following the transition to `PyTorch`-based infrastructure in March 2025.

As of January 2026, `Pydisort` has been downloaded over 192,000 times since its initial release in 2023, placing it within the top 10% of all Python packages by download counts. The user base spans over 140 countries, reflecting its global reach and impact.

Together, these trends reflect the growing recognition of `Pydisort` as a reliable, scalable, and versatile tool for radiative transfer calculations that lowers technical barriers while enabling new classes of scientific inquiry.

## 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 Dr. Andrew Ryan for early testing and feedback on the package.

## AI Usage Disclosure

GitHub Copilot contributed to several pull requests by assisting with the generation or refactoring of test cases, updating Python type stubs, and improving documentation clarity. All AI-assisted contributions were reviewed, edited, and validated by human authors prior to merging. For all other parts of the software and this paper, no generative AI tools were used, and all authors take responsibility for the final content.

## 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

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