MontePy: a Python library for reading, editing, and writing MCNP input files.

Micah D. Gale [1¶], **Travis J. Labossiere-Hickman** [1], **Brenna A. Carbno**[1], and **Andrew J. Bascom** [1]

1 Idaho National Laboratory, USA ¶ Corresponding author

DOI: 10.21105/joss.07951

Software

- Review
- Repository
- Archive

Editor: Kelly Rowland

Reviewers:

- @munkm
- @jpmorgan98

Submitted: 17 June 2024
Published: 09 April 2025

Summary

The Monte Carlo N-Particle (MCNP) radiation transport code is a highly capable and accurate code with a long legacy. MCNP uses the Monte Carlo simulation process to simulate the path of particles (e.g., neutrons, photons, charged particles, etc.), and their interaction with materials. It is widely used in nuclear engineering, high-energy physics, and other fields. Its origins in the mid-twentieth century predate many modern software conventions. MCNP users provide an input file to MCNP, which it then uses to create an internal representation of the simulation problem. These input files originally had to be stored as punchcard decks, and the user manual still uses the terminology of cards and decks, despite moving beyond punchcards. MCNP predates nearly all modern human readable markup or data serialization languages, such as the extensible Markup Language (XML), the Standard Generalized Markup Language (SGML), YAML (YAML Ain't Markup Language), and Javascript Object Notation (JSON). Due to this, MCNP uses an entirely custom defined syntax language for its input, making off-the-shelf libraries for XML, YAML, and JSON impossible to use for scripting various operations on MCNP input files (Kulesza et al., 2022).

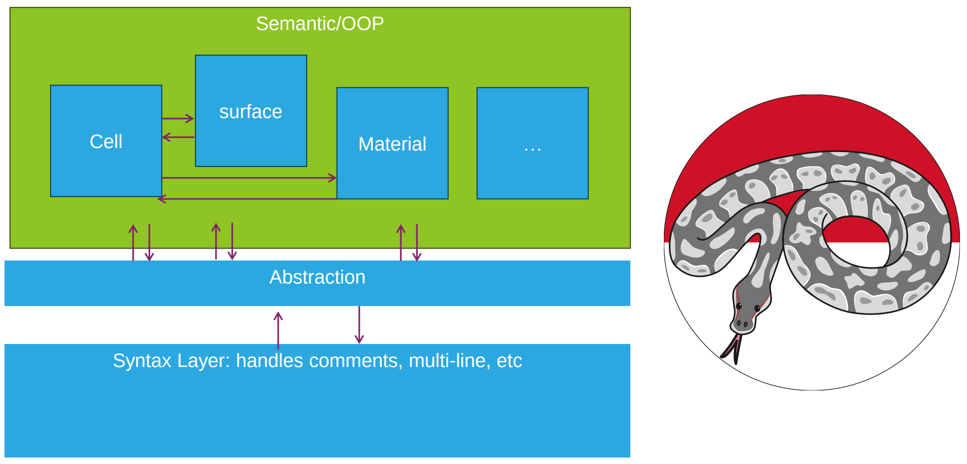

Figure 1: Diagram of how the different models of MontePy interact to form an Object Oriented Programming (OOP) interface.

MCNP simulation problems use three-dimensional constructive solid geometry (CSG). The simulation is composed of a series of cells, representing spatial regions in the modeled geometry. These cells have assigned densities, and are linked to a material definition. These materials define the relative amounts of different isotopes in the material, and can be shared between cells. The cell geometry in CSG is defined by a series of Boolean set operations of geometry primitives, which are usually quadratic surfaces. For instance, a cylindrical nuclear fuel pellet's geometry could be defined as the inside of an axially infinite cylinder with a specified radius, and above a bottom plane, and below a top plane.

MontePy is a Python library for reading, editing, and writing these MCNP input files. It provides an object-oriented programming (OOP) interface for interacting with the simulation problems. MontePy does not perform any of its own radiation transport, or neutronics calculations. MontePy uses a syntax parser built on top of the Python package for parsers: SLY (Beazly & contributors, 2016). This parser builds a concrete syntax tree of the input file. An example syntax tree for defining a cell is given in Figure 2. This allows MontePy to parse the input file without losing any information or formatting. The MontePy objects that represent MCNP objects such as: cells, surfaces, materials, etc., crawl through this syntax tree and update their internal values to reflect this tree. These attributes will then be exposed to the user as properties. Once a file has been fully read in, all objects will be linked together. The interaction between the components of MontePy are shown in Figure 1. For instance a cell object will be given a "pointer" to the material object that it is defined to be filled with. When the user wants to write their modified model to file this process will be reversed. The objects will crawl their syntax tree as necessary to ensure it has the correct value. If a value has changed, MontePy will try to match the numerical precision that the user used initially in the input file.

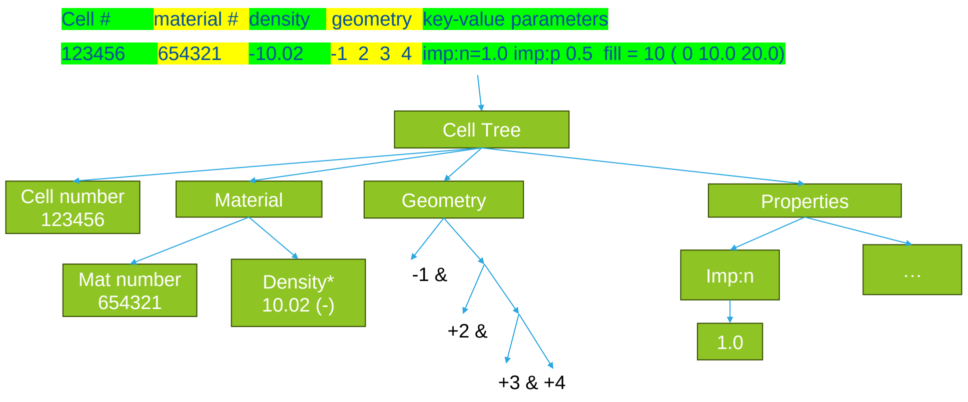

Figure 2: The syntax for defining a cell for MCNP, and the syntax tree that MontePy will extract from it.

MontePy is focused on using good software practices to simplify adoption as much as possible. MontePy uses industry standard continuous integration and continuous deployment (CI/CD) tools to test all changes, and ensure changes meet the standards of MontePy. Currently the test suite has over 380 tests, which have over 98% code coverage of the source code. This practice significantly reduces the risk of regressions and the introduction of new bugs. All software changes must be reviewed by at least one person prior to being accepted for deployment to users. MontePy is also written only in Python and only has two dependencies. This was intentionally done to make it as easy as possible for a new user to install it, which they can do with a single command. The decision to only use Python was made to facilitate the creation of a user community. All end-users should know how to write some Python, so any user could, with some guidance, become a developer.

Statement of Need

MCNP is a popular Monte Carlo radiation transport code. It has nearly unmatched capabilities in its physics modeling, and its support for 37 different particle types (Kulesza et al., 2022). However, due to the fact that MCNP uses a custom syntax, it requires referencing other objects (e.g., a cell referencing a material) by their number, and tendency of radiation transport simulations to be complex, working with MCNP input files can be tedious and error-prone. This issue is well suited for automation.

Current packages do exist for automating the generation of new MCNP input files. One such package is the Advanced Reactor Modeling Interface (ARMI). ARMI is a modular open-source framework for coupling multiple simulation codes. A closed-source MCNP plugin for ARMI does exist. ARMI is more focused on making its internal model of a nuclear reactor fit in an MCNP input file, rather than being able to open and parse an arbitrary MCNP model (Touran et al., 2017). This is indicative of the various tools that have been created for working with MCNP in the past. They tend to be purpose-built for a specific problem, or class of problems, and are not easily generalized. MCNP input models could also be created with a templating engine, like the Workflow Template and Toolkit System (WATTS) (Romano et al., 2022). This though requires the user to create a template from the set of problems they plan to model. This would be well suited for a sensitivity study where many very similar simulations are run, but not for making large edits to an input file or making a single problem from scratch.

PyNE: the Nuclear Engineering Toolkit offers some similar capabilities to WATTS for input generation. PyNE can create MCNP input files for specific features and extract some data from MCNP output files, but it cannot read MCNP input files. However, its full capabilities extend far beyond interfacing with MCNP. PyNE can simplify material creations, analyses of cross section data, transmutations of complex systems, and interfacing with other common nuclear engineering software and data formats (Scopatz et al., 2012). PyNE is an excellent companion tool to MontePy.

All of these previous solutions were incomplete in one way or another. None of these previous solutions are able to read in a previous MCNP input file and edit it in a general manner. In addition, WATTS does not have any fundamental understanding of what fields are for a user provided MCNP template. The same is true for a myriad of application-specific industry tools that are tailor-made for specific problems. There is a clear need for an object-oriented interface to these files that can both "understand" the input, and read and edit the files. This sort of model interface has been present for years in the Python API for the Monte Carlo code, OpenMC (Romano et al., 2015). Since its incorporation in the code, this interface has become by far the most dominant user interface for that code, as opposed to manual editing of the XML input files.

Ideally this object-oriented interface should be in Python as it is such a prolific language, especially among novice and intermediate programmers. A few such libraries do exist: MCNPy, mckit, and others discussed later. MCNPy is a Python wrapper for a java engine that can read, edit, and write MCNP input files. It can "understand" MCNP inputs, or as the authors put it, it has a "metamodel" for MCNP (Kowal et al., 2023). Having a library written in another language than what the user is used to, introduces another barrier to converting a user into a developer. This could present a serious barrier to developing a thriving user and developer community for this open source software. It does not appear that MCNPy has any automated testing suite at this time, and so there is no guarantee that it will actually perform the functions it claims to. In addition it imposes additional formatting requirements on an input file that is read, beyond what MCNP requires (Kulesza et al., 2022). Mckit on the other hand is written primarily in python, and does use automated testing. Unfortunately the existing documentation is difficult to acces, incomplete, and primarily in russian, so it is currently difficult to use for those who cannot read russian. It was difficult to assess the state of this project due to this. It appeared that mckit is more of a functional programming style library, rather than an object-oriented programming style (Rodionov & Portnov, 2024). MontePy provides all of these listed capabilities, while also being written purely in Python, and avoiding this barrier to forming a thriving open source community.

The authors attempted to find as many open-source Python libraries which overlapped MontePy's capabilities as possible. This was not an exhaustive seach, but should cover many such libraries. Given the number of libraries found the following lists will simply be an attempt to categorize these libraries.

The first group of libraries are those which attempt to have a read, edit, write capability for MCNP input files. These all do not fully parse the inputs as they do not use context-free parsers, and are generally feature limited, and may lack sufficient documentation. These libaries are:

- numjuggler (Travleev et al., 2022)
- MCNP Input Reader (Mariano, 2022)
- mctools (Laghi, 2023)
- mc-tools (Batkov et al., 2024)
- PyMCNP (Persaud et al., 2024)

There are even more tools that specialize in input templating and generation. These are clearly not complete alternatives as they lack the ability to read MCNP input files. These libraries are:

- CardSharpForMCNP (Pacific Northwest National Laboratory, 2025)
- wig (Hagen, 2021)
- Plugin-MCNP [for Funz] (Richet, 2023)
- GDNP (niess, 2018)
- map-stp (Portnov, 2024)
- MCNP Input Generator (ikarino, 2021)
- Neutronics Material Maker (Shimwell et al., 2024)

There are also libraries that specialize in parsing an MCNP input file in order to convert the model to be an input for another program:

- MCNP Conversion tools for OpenMC (Romano et al., 2024)
- t4_geom_convert (Mancusi, 2024)

There are also libraries that have to parse MCNP inputs to some extent as they provide MCNP syntax highlighting support for various text editors:

- MCNP-syntax-highlighting (Turkoglu, 2018)
- NPP_MCNP_Plugin (Marcinkevicius, 2025)
- vscode_mcnp (Repositony, 2024)

Finally there are the libraries that have been purpose built for working with and automating a specific type of MCNP models:

- BEMP_Thesis (Galdon, 2024)
- MCNP6-HPGe_Detector_simulation (Hung, 2023)
- rodcal-mcnp (Park, 2021)

MontePy is currently targeting two primary communities. First, Nuclear Engineers with moderate Python experience as a user base. The goal is to get these users to use the interface to remove the tedium from their work when they need to make some modification to their model. In addition, these users can use MontePy to quickly interogate, and retrieve information from their models in order to validate them, or to just answer some questions they had about them. The other target user is the Nuclear Engineer developer, making automation tools. Many nuclear engineering departments have a large MCNP model that they need to frequently update. For instance, the authors of MontePy use a model of the Advanced Test Reactor (Campbell et al., 2021) for their work on a daily basis. This large and complex model needs to be updated every reactor cycle with the new fuel compositions, the specific control element configurations, etc. Their department does have an automation tool that relies heavily on template-like use of regular expressions. This tool will fail to run if the model is modified in a way that is allowed by MCNP, but which the tool cannot handle. This tool is a prime example of a real-life case where MontePy could be applied to improve the workflow and increase robustness.

Status of MontePy

As of MontePy 0.5.4, many of the most commonly used MCNP inputs (cards) are supported. These include:

- Cells, which are the base of an MCNP geometry and contain a material and a CSG geometry definition.
  - Cell modifier inputs:
    * `IMP` inputs which specify a cell's importance for variance reduction, and other uses.
    * `FILL`, `LAT`, and `U` inputs which are used for defining universes, and filling cells with those universes.
    * `VOL` input which specifies the volume for a cell
- Surface inputs, which are used to define the primitive surfaces used. All surfaces are supported at a basic level. The following surface types are supported in a semantic way where the constants are tied to their geometric meaning:
  - `PX`, `PY`, and `PZ` surfaces, which are planes perpendicular to a specific axis.
  - `CX`, `CY`, and `CZ` surfaces, which are cylinders parallel to a specific axis and centered at the origin.
  - `C\X`, `C\Y`, and `C\Z` surfaces, which are cylinders parallel to a specific axis, and not centered at the origin.
- `M` inputs, which define the composition of a specific material.
- `MT` inputs, which define a thermal scattering law to use for a specific material.
- `mode` inputs, which define which particle types to run in the simulation.
- `TR` inputs, which define a geometry transformation.

MontePy does not support reading output files, and there are no current plans to add such support. First, MCNP is export controlled software, with a publicly released manual. MontePy was based solely on this manual. It does not document the formatting of the MCNP output files, so this feature is not included. Secondly, there is already an Open-Source tool available to read some MCNP output files, MCNPtools. This is a Python wrapper for a C++ tool to read meshtal, and mctal files output by MCNP (Bates et al., 2022). So for the time being, to avoid scope creep, the core MontePy developers will not be adding support for output files to allow development to focus on supporting more input features.

Future Work

MCNP supports over 140 different inputs (cards). For almost all of the remaining input types that MontePy doesn't support the information from the input is still available to the user. The next planned release at the time of publication is version 1.0.0. This new release is significant redesign of the material definition interface, making the material interface much more user-friendly. The exceptions are those inputs with syntax that conflicts with the rest of MCNP, which need to be handled specifically on their own. Adding more object-oriented support for all of these inputs is an ongoing project. Development is primarily prioritized by most commonly used inputs. Finally, MontePy and OpenMC's Python interface have many similar features. Harmonizing MontePy and OpenMC to be intercompatible would unlock a whole new set of possibilities. It would then be possible to translate OpenMC models to MCNP, and vice versa, which would be ideal for code-to-code comparisons.

Acknowledgments

Work supported through the Advanced Fuels Campaign (AFC) under DOE Idaho Operations Office Contract DE-AC07-05ID14517. The authors wish to thank the U.S. Department of Energy Office of Isotope R&D and Production for their vital and continued support and funding of the Co-60 program at INL under Contract No. DE-AC07-05ID14517. Co-60 is sold by the National Isotope Development Center (NIDC). Quotes on Co-60 can be obtained from NIDC at www.isotopes.gov/products/cobalt. This research made use of Idaho National Laboratory's High Performance Computing systems located at the Collaborative Computing Center and supported by the Office of Nuclear Energy of the U.S. Department of Energy and the Nuclear Science User Facilities under Contract No. DE-AC07-05ID14517.

References

Bates, C. R., Bolding, S. R., Josey, C. J., Kulesza, J. A., Solomon, C. J., Jr., & Zukaitis, A. J. (2022). *The MCNPTools package: Installation and use* (Report LA-UR-22-28935). Los Alamos National Laboratory. https://doi.org/10.2172/1884737

Batkov, K., Borghi, N., Furutaka, K., Ansell, S., & Vezhlev, E. (2024). *Mc-tools* (Version 1.0.post1). https://github.com/kbat/mc-tools

Beazly, D., & contributors, S. (2016). *SLY (sly lex yacc)*. https://sly.readthedocs.io/en/latest/

Campbell, J., Marshall, F., & Longhurst, G. (2021). *Advanced test reactor user guide* (Report INL/EXT-21-64328). Idaho National Laboratory. https://doi.org/10.2172/1826354

Galdon, M. (2024). *BEMP_thesis* (Version 1.0.0). https://github.com/mgaldon17/BEMP_Thesis

Hagen, A. (2021). *Wig*. https://github.com/alexhagen/wig

Hung, B. T. (2023). *MCNP6-HPGE_detector_simulation*. https://github.com/hungbt1908/MCNP6-HPGE_Detector_Simulation

ikarino. (2021). *MCNP input generator*. https://github.com/ikarino/mcnp_input_generator

Kowal, P. J., Blake, C. E., Dominesey, K. A., Lefebvre, R. A., Brown, F. B., & Ji, W. (2023). Enhancing monte carlo workflows for nuclear reactor analysis with metamodel-driven modeling. *Nuclear Science and Engineering*, *197*(8). https://doi.org/10.1080/00295639.2022.2153617

Kulesza, J., Adams, T., Armstrong, J. C., Bolding, S. R., Brown, F. B., Bull, J. S., Burke, T. P., Clark, A. R., Forster, R. A. I., Giron, J. F., Grieve, T. S., Josey, C. J., Martz, R. L., McKinney, G. W., Pearson, E. J., Rising, M. E., Solomon, C. J. Jr., Swaminarayan, S., Trahan, T. J., … Zukaitis, A. J. (2022). *MCNP code version 6.3.0 theory & user manual* (Report LA-UR-22-30006, Rev. 1). Los Alamos National Laboratory. https://doi.org/10.2172/1889957

Laghi, D. (2023). *Mctools*. https://github.com/dodu94/mctools

Mancusi, D. (2024). *t4_geom_convert* (Version 1.1.2). https://github.com/arekfu/t4_geom_convert

Marcinkevicius, B. (2025). *NPP_MCNP_plugin*. https://github.com/kordusas/npp_mcnp_plugin

Mariano, G. (2022). *MCNP input reader* (Version 0.2.1). https://github.com/

ENEA-Fusion-Neutronics/MCNP-Input-Reader

niess. (2018). *GDNP*. https://github.com/niess/gdnp

Pacific Northwest National Laboratory. (2025). *CardSharpForMCNP* (Version 1.4.2). https://github.com/pnnl/CardSharpForMCNP

Park, P. (2021). *Rodcal-mcnp*. https://github.com/patrickpark910/rodcal-mcnp

Persaud, A., Unzueta, M. A., Surry, E. K., & Parsons, A. M. (2024). Python-based software tools for MCNP. *2024 IEEE Nuclear Science Symposium (NSS), Medical Imaging Conference (MIC) and Room Temperature Semiconductor Detector Conference (RTSD)*, 1–1. https://doi.org/10.1109/NSS/MIC/RTSD57108.2024.10655039

Portnov, D. (2024). *Map-stp*. https://github.com/MC-kit/map-stp

Repositony. (2024). *Vscode_mcnp*. https://github.com/repositony/vscode_mcnp

Richet, Y. (2023). *Funz plugin-MCNP* (Version 1.16-0). https://github.com/Funz/plugin-MCNP

Rodionov, R., & Portnov, D. (2024). *Mckit* (Version 0.8.3). https://github.com/MC-kit/mckit

Romano, P. K., Horelik, N. E., Herman, B. R., Nelson, A. G., Forget, B., & Smith, K. (2015). OpenMC: A state-of-the-art monte carlo code for research and development. *Annals of Nuclear Energy*, *82*. https://doi.org/10.1016/j.anucene.2014.07.048

Romano, P. K., Li, K., Shriwise, P., & Valderrama, J. (2024). *MCNP conversion tools for OpenMC*. https://github.com/openmc-dev/openmc_mcnp_adapter

Romano, P. K., Stauff, N. E., Ooi, Z. J., Miao, Y., Lund, A., & Zou, L. (2022). WATTS: Workflow and template toolkit for simulation. *Journal of Open Source Software*, *7*(79). https://doi.org/10.21105/joss.04735

Scopatz, A., Romano, P. K., Wilson, P. P. H., & Huff, K. D. (2012). *PyNE: Python for nuclear engineering* (Vol. 107, pp. 985–987). American Nuclear Society.

Shimwell, J., Billingsley, J., Buendia, C., & Neutronics Material Material Contributors. (2024). *Neutronics material maker* (Version 1.2.1). https://github.com/fusion-energy/neutronics_material_maker

Touran, N., Gilleland, J., Malmgren, G., Whitmer, C., & Gates, W. H. (2017). Computational tools for the integrated design of advanced nuclear reactors. *Engineering*, *3*(4). https://doi.org/10.1016/J.ENG.2017.04.016

Travleev, A., Previti, A., & Portnov, D. (2022). *Numjuggler* (Version 2.42.36). https://github.com/travleev/numjuggler

Turkoglu, D. (2018). *MCNP-syntax-highlighting*. https://github.com/danyalturkoglu/MCNP-syntax-highlighting