

FrendyPlus: An Extensible Nuclear Data Processing Code Interfacing Frendy

Changyuan Liu^{a*}

^aNew Compute Laboratory¹, No. 89 Jianguo Road., Beijing, China 100025

*Corresponding author: changyuan_liu@163.com

1. Introduction

Open-source repository for technical demonstration:

<https://jihulab.com/newcomputelab/frendyplus>

The beginning section first reviews the nuclear data processing codes, then discusses about the Julia and Python programming languages with support for libraries written in other languages through the C interfaces. Next, it addresses the existing practices of Python interface to nuclear data processing codes. And finally, it gives the purposes of this work.

1.1 Review of the Nuclear Data Processing Codes

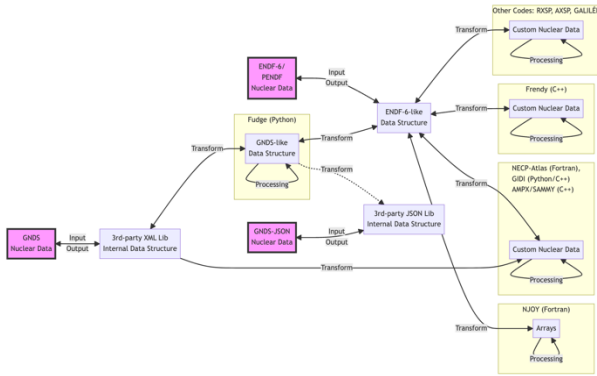


Fig. 1. The current states of support for nuclear data formats in processing codes

Nuclear data provide fundamental physics properties for nuclear engineering and radiological sciences. Because of its key roles playing in applications, the Nuclear Data Section (NDS) of International Atomic Energy Agency (IAEA) and OECD Nuclear Energy Agency (NEA) dedicate to maintain a central database of evaluated nuclear data.

The evaluated nuclear data are usually not directly ready for being used in applications, and the data need to be processed into computational friendly formats. This is where the nuclear data processing codes participate.

The evaluated nuclear data are stored in the ENDF-6 [1], GNDS [2] or GNDS-JSON [3] formats. Fig. 1 summaries the current state of support for nuclear data formats in processing codes.

Most nuclear data processing codes such as NJOY [4] are designed to support ENDF-6 format. The data are first transformed into data structures similar to ENDF-6 data layout, and then are transformed into other custom internal data structures. These internal

data structures work as scratchpads to store intermediate data produced during data processing.

GNDS is a newer format. Although not specifying what data format is used, the practice is the XML (later HDF5) format, which is more structural and flexible than ENDF-6. Up to now, there are already codes such as NECP-Atlas [5], GIDI [6], AMPX/SAMMY [7] and Frendy [8] support GNDS, and other codes such as RXSP [9], AXSP [10], and GALILÉE [11] are modern programs which are expected to be easily integrated with support for GNDS.

In the meanwhile, the Fudge [12] code is designed to support GNDS format as its core function. The data are first accessed by 3rd-party XML library, and are then transformed into custom internal data structures similar to the GNDS data layout.

GNDS-JSON follows mostly the GNDS specification, but using JSON to replace XML as the base data language. With a closer similarity to data structures in many programming languages than GNDS, GNDS-JSON reduces the nuclear data complexities. Until now, there have been no processing code supporting GNDS-JSON.

Section 2 will discuss more about the background about GNDS and GNDS-JSON.

1.2 Julia and Python Programming Languages and the C Interfaces

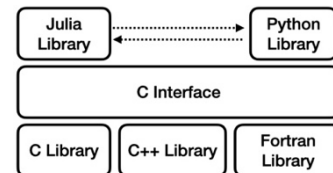


Fig. 2. The Julia and Python programming languages and the C interfaces to C, C++ and Fortran libraries

Many processing codes as illustrated in Fig. 1 are born at earlier times, where C, C++, and Fortran programming languages are adopted for best performance. Some recent codes such as Fudge and GIDI have begun to write in Python [13] to reduce the resources spent on software development.

In addition to Python, Julia [14] is another relatively new programming languages with comparable easiness of coding to Python, but Julia generally delivers higher performance than Python and is designed for scientific computation. A comparison between Julia and Python is shown in Table I.

¹ A research start-up group

Unlike C, C++ or Fortran, both Julia and Python do not require programmers to manually manage memories. Instead, automatic memory management through garbage collection is adopted as the key language feature. At the time of coding, both languages allow dynamic types to reduce the coding efforts.

However, Python keeps the mechanism of dynamic types to the runtime, but Julia uses type inference to determine all possibilities of the types at compilation time, and compile the code into binaries. As a result, Julia natively executes faster than Python since the types are static at runtime.

What makes it possible for Julia to have dynamic types at compilation and static types at runtime is its special compiler design. The dynamic types of all possibilities are compiled separately, where the same code of different types may have different binary codes. And at runtime, the corresponding binary codes for the inferred type are executed at runtime. While as, at runtime, Python uses interpreter to determine the type, and translate the corresponding source code into binary code to execute. There is extra interpretation work at runtime for Python, so native code Python is at greater chance slower than Julia.

Table I: Comparison of Features for Julia and Python

	Julia	Python
Automatic Memory Management	Yes, garbage collection	Yes, garbage collection
Dynamic Types	Yes	Yes
Dynamic Types at Runtime	No, use type inference	Yes
Compiled or Not	Yes, just-in-time (JIT)	No

As indicated in Fig. 2, both Julia and Python provide interface in C language to precompiled dynamic libraries written in other languages such as C, C++, and Fortran.

It is also possible for Julia and Python to inter-operate with each other in the source code. Section 4 discusses the use of the PyCall [15] library to allow applications written in Julia to directly use the OpenMC [16] utilities code in Python to generate the HDF5 [17] format nuclear data libraries.

1.3 Existing Python Interface to Nuclear Data Processing Codes

PyNJOY [18] is an interface to the NJOY nuclear data processing code, whose purpose is to provide an additional module to produce cross section libraries for deterministic neutron transport code.

OpenMC code also provides some utilities in Python to generate NJOY inputs and call NJOY executable to process nuclear data.

In these two approaches, the Python applications communicate with NJOY through the filesystem, and the Python applications and NJOY are running in separated memory space.

1.4 Purposes of the FrendyPlus and Contribution

In this work, the FrendyPlus extensible interface to Frendy processing code is proposed to fulfill the following goals:

- Provide a C interface for Frendy code to Julia and Python, where the traditional input files are replaced with frontend Julia/Python source code
- Demonstrate initial support of the GNDS-JSON formatted nuclear data for Frendy
- Propose a nuclear data processing pipeline with seamless integration with the OpenMC HDF5 formatted nuclear data utilities

1.5 Description of Contents

In section 2, an overview of Frendy and FrendyPlus in the future nuclear data ecosystem is presented. In section 3, details about the design of FrendyPlus are elaborated. And in section 4, a verification of nuclear data processing using Monte Carlo neutron transport benchmarks is addressed. Finally in section 5, the cross-platform dependencies are discussed.

2. FrendyPlus and Future Nuclear Data Ecosystem

This section provides an overview of the Frendy and FrendyPlus processing code in the nuclear data ecosystem.

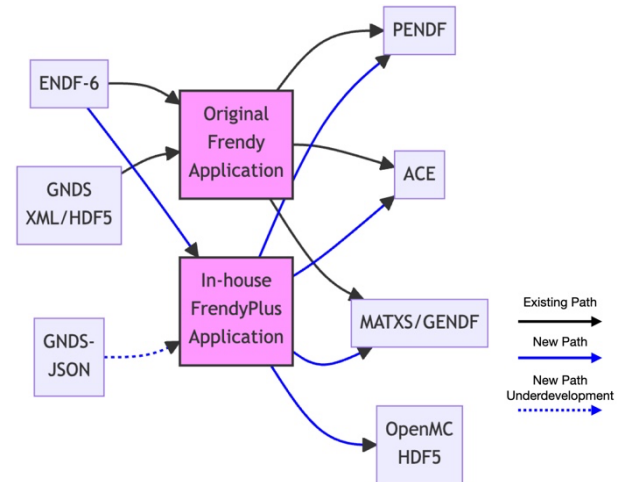


Fig. 3. Overview of Frendy and FrendyPlus in the nuclear data ecosystem. PENDF is the intermediate nuclear data file generated during data processing. ACE is the nuclear data format used for continuous energy particle transport. MATXS and GENDF are the nuclear data formats used for multigroup particle transport. OpenMC HDF5 stands for the nuclear data format used in OpenMC.

2.1 New Data Processing Paths from FrendyPlus

The original Frendy code takes evaluated nuclear data in the ENDF-6 and GNDS data format, and generate PENDF and ACE and MATXS/GENDF formatted data. Fig. 3 illustrates where Frendy and FrendyPlus play roles in the nuclear data ecosystem.

FrendyPlus allows additional nuclear data processing paths such as reading the GNDS-JSON formatted data and generating computing friendly data formats other than ACE: such as the HDF5 format for OpenMC.

One advantage of FrendyPlus is that it allows agile development of new paths in the Julia/Python programming languages.

2.2 GNDS and XML/HDF5/JSON Implementations

When original proposed, GNDS is a specification of the data structure of the evaluated nuclear data without regulations on which underlying data format should be adopted.

At the earlier time shortly after GNDS specification is proposed, the community has reached a consensus for XML to be used as the underlying data format. But XML greatly increases the nuclear data file size.

Much recent, to alleviate the problem of occupying too much storage space, parts of especially large data arrays are replaced with HDF5. So GNDS is now usually stored in a mixed XML and HDF5 format.

GNDS-JSON takes another route that uses JSON as the underlying data format, and when desired, binary formats such as MessagePack [19] serves as a replacement. Data in MessagePack can be accessed in the same way as JSON, and data in JSON and MessagePack can be mutually exchanged without loss of information.

Instead of considering it to be a new nuclear data format, GNDS-JSON should be thought as the GNDS specification implemented in JSON.

Table II provides a brief comparison of implementations of the GNDS specification in XML, HDF5 and JSON.

Table II: Comparison of XML/HDF5/JSON Implementations of GNDS

	XML	HDF5	JSON
Text or Binary	Text	Binary	Text
Binary Solution	Partially in HDF5	N/A	e.g. Message Pack
Concepts	Label, property, children label	Dataset, property, attribute	Object, key-value pairs
Hierarchy	Label-children	Dataset-attribute	Object as value
Native Integer, Float, Bool	All are strings	Yes	Yes, treat integers

and String			and floats as numbers
Array of Native Types	No	Yes	Yes
Array of Complex Objects	No	No	Yes
Schema for Format Validation	Yes	No	Yes
Require Dedicated Data Access Code	Yes, e.g. GIDI+	Yes, e.g. GIDI+	No for many programming languages

XML and HDF5 share the similar pattern that a parent structure has properties and children. In XML these children are labels, and in HDF5 these children are primitive types or datasets. In JSON, the hierarchy structure is simplified where both properties and children are combined into the key mapped values.

What to distinguish JSON from XML and HDF5 is its native support for the arrays of complex data structures. This simplifies the data structural complexity as discussed by Liu² [3].

Because of this simplification in format, GNDS-JSON does not require the preparation of dedicated computer codes for data access interface. For example, many codes such as GIDI and Frendy adopt the interface GIDI+ [20] for manipulating GNDS in XML/HDF5. The GIDI+ interface is bulk in size and requires additional maintenance efforts. Moreover, this interface is prepared in C++ and Python programming languages, which will restrict the migration to other programming languages. See discussion by Liu [3] for further information.

2.3 Why GNDS-JSON?

In this section, the benefits of GNDS-JSON are further emphasized as following:

- Reduced nuclear data complexity: because JSON is simpler than XML/HDF5, the nuclear data complexity is reduced, so as that the efforts to manipulate the nuclear data is reduced.
- Consistent support for binarization: unlike the XML format uses HDF5 for binarization, JSON has more concise solutions for binarization using formats such as MessagePack. When two formats XML and HDF5 are used together, programming has increased complexity.

² <https://jihulab.com/newcomputelab/publications/>
(Temporary copy of publication: ICONE30.pdf)

- Format validation: GNDS-JSON has completed solution for format validation with JSON Schema. However, the format validation for GNDS in XML/HDF5 is not as completed as GNDS-JSON.
- Not requiring dedicated data access code: because of the data structural similarity of JSON to objects in many programming languages, no dedicated data access code is required. The default or common 3rd party JSON libraries in many programming languages can be used out of box.

2.4 Sustainability of GNDS-JSON Support

As illustrated in Fig. 1 and emphasized by Fig. 4, the GNDS-JSON nuclear data is generated by tools based on the Fudge code.

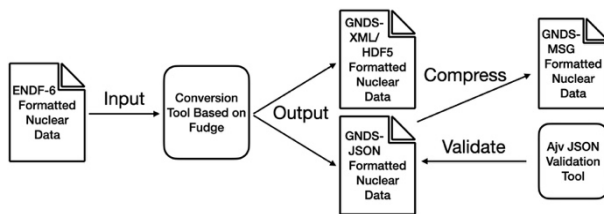


Fig. 4. Generation of GNDS and GNDS-JSON formatted nuclear data. MSG stands for the MessagePack binary format for JSON.

Because GNDS (XML/HDF5) and GNDS-JSON are rooted from the same GNDS specifications, the GNDS-JSON formatted data can be converted from GNDS (XML/HDF5) with little efforts.

Published in the on-line repository [21], nuclear data from BROND 3.1, CENDL 3.2, ENDF/B-VIII.0, JEFF 3.3, JENDL 4.0 and JENDL 5.0 in GNDS-JSON are open to public access. In the future, when a new nuclear data library is published, only a one-time effort is required on to prepare a copy in the GNDS-JSON format.

Because of the availability of nuclear data in GNDS-JSON now and in the foreseeable future, the adoption of GNDS-JSON is a sustainable option.

3. Technical Details of FrendyPlus

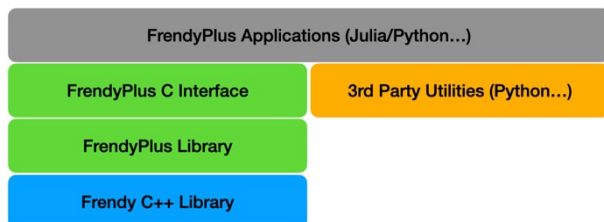


Fig. 5. An overview of the FrendyPlus extensible interface to the Frendy nuclear data processing code

This section addresses details about the design of FrendyPlus as an extensible interface to the Frendy nuclear data processing code.

3.1 The Frendy Nuclear Data Processing Code

The Frendy nuclear data processing code has a recent release of the version 2 (2022/11/04, Ver. 2.01.000) [18]. The Frendy code takes nuclear data in the ENDF-6 format, which is mapped to the ENDF C++ objects in memory. However, the ENDF objects are not directly used for data processing procedures, and these ENDF objects need being converted into the Frendy C++ objects. All processing procedures such as cross section reconstruction are built on these Frendy C++ objects.

Traditionally, a nuclear data processing code follows the way NJOY code adopts, where an input file specifying the steps of nuclear data processing is read and followed by the program. Frendy is able to directly handle NJOY input files, but also has its own version of input files.

3.2 FrendyPlus Design Overview

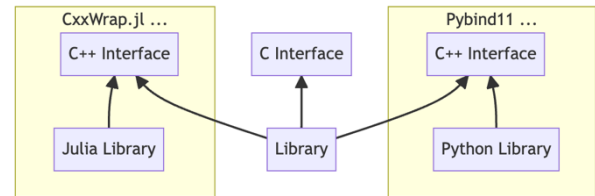


Fig. 6. Comparison of the C and C++ interfaces to a library

One of the design goals of FrendyPlus is to enable an easily extensible interface to Frendy. So, the Julia and Python programming languages are chosen as the frontend languages, with which the users interact by writing source codes.

As illustrated in Fig. 5, on top of the FrendyPlus library, a C interface to FrendyPlus is prepared for Julia/Python to communicate with the FrendyPlus library. Unlike previous approaches used in PyNJOY and OpenMC NJOY utilities, the communication with underlying processing code is done in memory.

Besides the C interface, as illustrated in Fig. 6, Julia can use the 3rd-party library such as CxxWrap.jl [22], and Python can use the 3rd-party library such as PyBind11 [23] to better interact with the library underneath written in C++. These interfaces are called the C++ interface. In these approaches, the programming patterns in Julia or Python are kept intact, but it requires external coding efforts for the development of interface to the underneath library. What makes it worse, these approaches require compilation of the underlying C++ libraries and linkage to the Julia or Python development libraries (written in C/C++), which will cause extra efforts to config the software environment and build up compilation toolchain.

Therefore, to make the FrendyPlus library more portable, plain C interface instead of C++ interfaces is shipped.

The FrendyPlus libraries are tested in computing environment with a combination of operating systems (OS) and CPU architectures as summarized in Table III. The precompiled FrendyPlus libraries on Windows/Linux /MacOS and x86-64 (AMD64)/aarch64 (ARM64) CPUs are provided in the online repository [24], except for Windows on aarch64 CPUs (not for technical reasons). In all platforms listed, there are Julia and Python interface visioned. The Julia interface is ready, but the Python interface is under development. In the meanwhile, the calling to the FrendyPlus libraries through C interface are tested on all supported platforms.

Table III: Current System Architecture Support for FrendyPlus

OS	Architecture	FrendyPlus Library	Julia Interface	Python Interface
Windows	x86-64	Yes	Yes	Not yet
Windows	aarch64	Not yet	Yes	Not yet
Linux	x86-64	Yes	Yes	Not yet
Linux	aarch64	Yes	Yes	Not yet
MacOS	x86-64	Yes	Yes	Not yet
MacOS	aarch64	Yes	Yes	Not yet

The preparation of the FrendyPlus libraries keeps the minimization of the external dependencies in mind. Details about the dependencies are summarized in Table IV. On Linux, the Lapack version 3 libraries are needed, which is accessible on nearly all Linux distributions. On MacOS and Windows, no external libraries are required.

Table IV: External Dependencies of the FrendyPlus Libraries

OS	Dependencies
Windows	All dependencies internal
Linux	Need Lapack version 3 runtime libraries
MacOS	OS provided frameworks

The FrendyPlus online repository has the following structures:

- /doc: documentation providing extra information beyond the ReadMe file
- /examples: example nuclear data and input files
- /julia: applications in Julia programming language
- /lib: platform dependent compiled FrendyPlus libraries with necessary dependencies
- /python: applications in Python programming language

3.3 Support for GNDS-JSON Nuclear Data Format

One of the design goals for FrendyPlus is to provide first class support for the GNDS-JSON nuclear data format as to ENDF-6.

In order to support the GNDS-JSON formatted nuclear data, utilities to convert the GNDS-JSON data into the Frendy objects are needed according to the recent JAEA's efforts to support GNDS in Frendy [25].

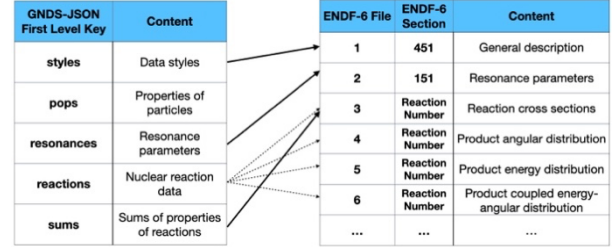


Fig. 7. The conversion rules from GNDS-JSON to ENDF-6

The conversion requires the mapping from GNDS-JSON data structures to that of ENDF-6. As indicated by Fig. 7, some of the important first level keys in the GNDS-JSON data structure are identified.

The general description is extracted from the data styles and mapped into ENDF-6 file 1 section 451. The resonance parameters are mapped into the ENDF-6 file 2 section 151.

Unlike ENDF-6, GNDS-JSON collects the data of the same reaction together. In ENDF-6, files such as number 3, 4, 5, 6 only store portions of the nuclear reaction data. So, the data corresponding to different ENDF-6 files need to be separately extracted from GNDS-JSON reaction data.

In GNDS-JSON, there are also sums of reaction properties. The most common sum is the sum of cross sections, which is mapped to ENDF-6 file 3.

Currently, in the technical demonstration shipped online, only enough implementation to process the neutron-neutron reaction (Material ID MAT=25) from ENDF/B-VIII.0 is provided.

As a side notice, JAEA is developing the next version of Frendy with support for GNDS [25]. The FrendyPlus may by then be upgraded to support GNDS as well.

3.4 FrendyPlus Applications

As discussed in section 3.2, there are some applications provided in Julia and Python (Python's interface not yet ready) to facilitate the processing of nuclear data. All applications have the same functionality between Julia and Python. On one hand, these applications provide examples for the user to integrate the FrendyPlus extensive interface into their own data processing pipeline, and on the other hand, these applications provide important common components for nuclear data processing.

Theses applications are listed as followed:

- *run_frendy*: it is the application (run through Julia/Python) providing the same interface to the original Frendy executable. It is for the user to quickly start up with the FrendyPlus without scarifying the familiarity with Frendy

- *build_ace_from_endf*: it is the application converting an ENDF-6 file to an ACE file. The steps including resonance reconstruction, Doppler broadening, gas production cross section generation, probability table generation and ACE file generation are decoupled. The communication with the Frendy library underneath is done in memory.
- *build_ace_from_gnds_json*: same as the *build_ace_from_endf* application, but the input file is in the GNDS-JSON format. The underlying support for GNDS-JSON is under development. The user can only test it with the provided neutron-neutron reaction data (Material ID MAT=25) from ENDF/B-VIII.0.
- *build_ace_thermal_from_endf*: it is the application converting an ENDF-6 thermal scattering file to an ACE file. The steps are similar to the neutron ACE file generation case, except that a thermal scattering cross section generation stage is inserted between gas production cross section generation and probability table generation.
- *build_openmc_hdf5_from_endf*: it is the application converting an ENDF-6 file into an HDF5 file for OpenMC Monte Carlo neutron transport code. Unlike ACE files, the nuclear data at different temperatures in HDF5 files are stored together, and the common parts among these ACE files are stored only once. With the new in-memory operation interface, no intermediate ACE files are generated. So, the storage space is saved. The verification of Monte Carlo neutron transport using OpenMC is further discussed in section 4.

4. Verification of Continuous Energy Monte Carlo Simulation

This section verifies the nuclear data generated from FrendyPlus with the Doppler defect pin-cell benchmarks [26] simulating with OpenMC code. It is also a demonstration of the seamless integration the OpenMC Python nuclear data utilities.

4.1 Integrated Support for OpenMC HDF5 Format

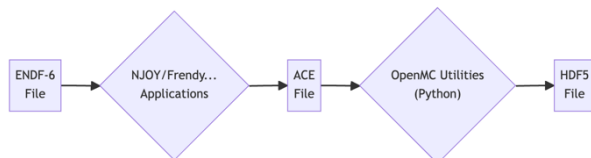


Fig. 8. Traditional workflow for converting ENDF-6 files to OpenMC HDF5 files

OpenMC uses custom HDF5 nuclear data files. Traditionally, as indicated in Fig. 8, ENDF-6 nuclear data files are first processed by code such as NJOY or Frendy to generate ACE files. Then, the ACE files are

converted by OpenMC Python utilities to generate HDF5 files.

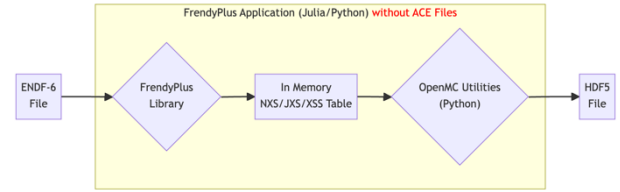


Fig. 9. An integrated FrendyPlus application in Julia or Python for converting ENDF-6 files to OpenMC HDF5 files without the generation of intermediate ACE files

However, since each temperature needs a separate ACE file, when the number of temperatures is large, there can be significant requirement of storage for these intermediate ACE files.

For this reason, the FrendyPlus provide a C interface to directly read the ACE NXS/JXS/XSS tables and directly send these tables to the OpenMC Python utilities in memory. As illustrated in Fig. 9, no intermediate ACE files are generated. So, the storage is saved. Moreover, floating point number precision is also increased, since the ACE files only keep 12 digits, and by accessing data directly in memory, full double precision is preserved, which is about 16 digits.

Thanks to the PyCall library, the Python data structures can be directly accessed and manipulated in Julia without data copy. The syntax to use Python libraries in Julia is similar to the native Python. For example, the following code in Python:

```
data = openmc.data.IncidentNeutron.from_ace(table)
```

is replaced by the Julia code:

```
local data = openmc.data.IncidentNeutron.from_ace(table)
```

where the syntax is almost the same.

The simulation of the Doppler defect benchmarks requires the generation of data at the temperatures of 0K, 600K, and 900K. Although not used, the 0K cross sections are intended for the resonance scattering calculation.

4.2 Simulation of the Doppler Defect UO₂ Pin-cell Benchmarks without Thermal Scattering

The Doppler defect benchmark problems study the Doppler broadening effects of fuel in the pin-cells. Simulation here includes 14 cases of Pin-cells filled with the UO₂ fuel. The data is from ENDF/B-VIII.0, and both the HDF5 formatted nuclear data from FrendyPlus and NJOY2016 are adopted by OpenMC. The simulation parameters are summarized in Table V.

Table V: Simulation Parameters for Doppler Defect UO₂ Pin-cell Benchmarks without thermal scattering

Parameter	Value
Code	OpenMC version 0.13.1
Total Batches	4,000

Inactive Batches	100
Particles per Batch	10,000
H in H ₂ O S(a,b)/Thermal Scattering	Not considered
Unresolved Resonance	Considered
Resonance Scattering	Not considered

The results of K-effective eigenvalues along with standard deviations are listed in Table VI.

Table VI: K-Effective Values for Doppler Defect UO₂ Pin-cell Benchmarks without Thermal Scattering

	FrendyPlus	NJOY2016
0.711% 600K	0.65779(10)	0.65760(10)
0.711% 900K	0.65229(10)	0.65232(10)
1.6% 600K	0.95773(13)	0.95808(13)
1.6% 900K	0.95011(13)	0.95030(13)
2.4% 600K	1.09994(14)	1.09998(14)
2.4% 900K	1.09139(14)	1.09154(14)
3.1% 600K	1.18040(14)	1.18054(14)
3.1% 900K	1.17129(14)	1.17169(14)
3.9% 600K	1.24503(14)	1.24525(14)
3.9% 900K	1.23563(15)	1.23578(15)
4.5% 600K	1.28148(15)	1.28177(14)
4.5% 900K	1.27213(15)	1.27209(14)
5.0% 600K	1.30693(14)	1.30635(14)
5.0% 900K	1.29730(14)	1.29695(15)

The difference in K-effectives in terms of the number of standard deviations between FrendyPlus and NJOY for each of the 14 cases are plotted in Fig. 10. All differences are within 4 times of standard deviations. So, the agreement of K-eigenvalues is good. Although the plot hints that FrendyPlus produces slightly larger K-Eigenvalues, there are too few comparison cases to draw an affirmative conclusion.

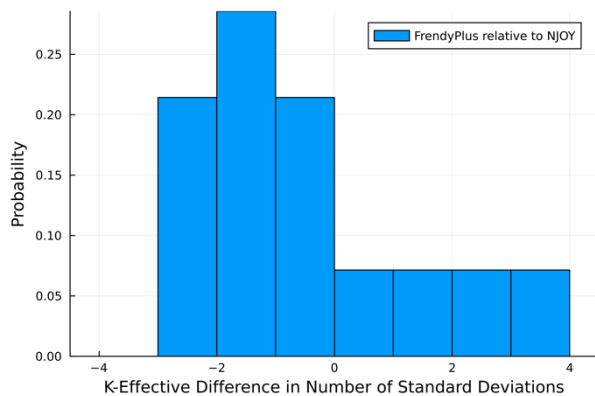


Fig. 10. Distribution of the relative difference K-Effective between FrendyPlus and NJOY in terms of number of standard deviations among the 14 cases of UO₂ benchmarks without thermal scattering

The Doppler coefficients calculated as in Eq. (1) are compared in Fig. 11. Both FrendyPlus and NJOY have the results agreed within 1-2 standard deviations.

$$\frac{\Delta \rho}{\Delta T} = \frac{k_{900K} - k_{600K}}{k_{900K} \times k_{600K}} \times 10^5 \text{ pcm/300K} \quad (1)$$

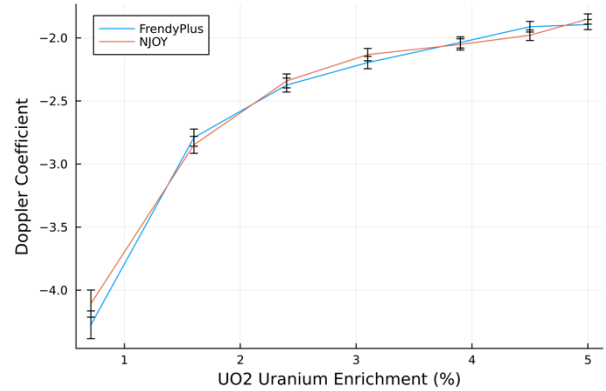


Fig. 11. Comparison of Doppler coefficients for UO₂ benchmarks without thermal scattering

4.3 Simulation of the Doppler Defect UO₂ Pin-cell Benchmarks with Thermal Scattering

Similar to section 4.2, in this section, the UO₂ pin-cell benchmark problems are simulated with Hydrogen in H₂O thermal scattering. The thermal scattering nuclear data file comes from ENDF/VIII.0 library, which is processed by FrendyPlus into the OpenMC HDF5 format. Again, no intermediate ACE files are generated. The simulation configurations follow Table V, except that the thermal scattering is considered.

The results of K-effective eigenvalues along with standard deviations are listed in Table VII.

Table VII: K-Effective Values for Doppler Defect UO₂ Pin-cell Benchmarks with Thermal Scattering

	FrendyPlus	NJOY2016
0.711% 600K	0.65365(10)	0.65349(10)
0.711% 900K	0.64801(10)	0.64809(10)
1.6% 600K	0.95308(13)	0.95342(13)
1.6% 900K	0.94589(13)	0.94559(13)
2.4% 600K	1.09541(14)	1.09542(14)
2.4% 900K	1.08669(14)	1.08707(14)
3.1% 600K	1.17623(14)	1.17615(14)
3.1% 900K	1.16720(14)	1.16723(14)
3.9% 600K	1.24102(14)	1.24105(15)
3.9% 900K	1.23204(14)	1.23184(14)
4.5% 600K	1.27794(14)	1.27813(14)
4.5% 900K	1.26867(14)	1.26833(15)
5.0% 600K	1.30320(14)	1.30311(14)
5.0% 900K	1.29345(14)	1.29337(14)

The difference in K-effectives in terms of the number of standard deviations between FrendyPlus and NJOY for each of the 14 cases are plotted in Fig. 12. All

differences are within 3 times of standard deviations. So, the agreement of K-eigenvalues is good.

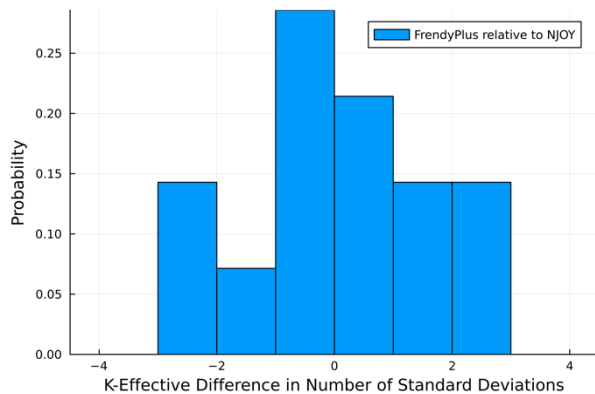


Fig. 12. Distribution of the relative difference K-Effective between FrendyPlus and NJOY in terms of number of standard deviations among the 14 cases of UO₂ benchmarks with thermal scattering

The Doppler coefficients calculated as in Eq. (1) are compared in Fig. 13. Both FrendyPlus and NJOY have the results agreed within 3 standard deviations.

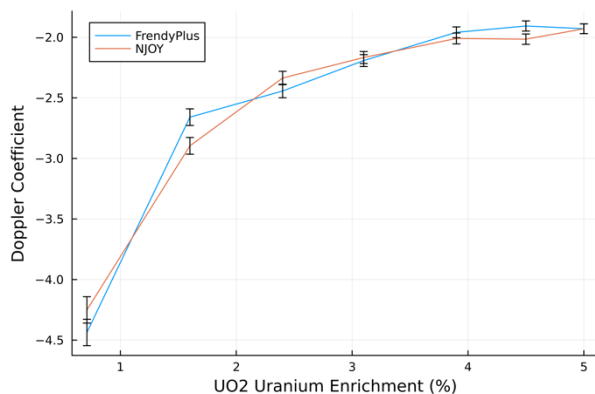


Fig. 13. Comparison of Doppler coefficients for UO₂ benchmarks with thermal scattering

5. Discussion of Cross-Platform Dependencies

This section discusses dependencies of the FrendyPlus libraries, including the modifications applied on the underlying Frendy libraries.

5.1 Free from Boost Library Linkage

The Frendy libraries rely on the Boost C++ libraries [27] to provides timing utilities, which requires dynamic linkage to the 'boost_chrono' and 'boost_timer' libraries. Relying on any specific version of Boost will impair the capabilities of cross-platform migration. So, the linkage to Boost library is removed.

5.2 Unix Directory Support on Windows

The Frendy libraries rely on the POSIX file system directory utility C header '<dirent.h>', which is replaced by a Windows substitute. [28]

6. Conclusions

FrendyPlus as an extensible interface in Julia/Python programming languages to the Frendy nuclear data processing code has been proposed. FrendyPlus enables nuclear data processing procedures built directly in Julia/Python source codes without the use of input files. Cross-platform FrendyPlus binary libraries have been provided to enable a consistent user experience across different OS and CPU architectures. The example scripts of processing neutron and thermal scattering reactions have been provided in the online repository. And in the meanwhile, the seamless integration with OpenMC Python utilities to generate the HDF5 data files has been demonstrated. In the demonstration, consistent results of K-Eigenvalues from FrendyPlus and NJOY for the Doppler defect UO₂ pin-cell benchmark problems are achieved. Moreover, FrendyPlus is the first nuclear data processing code demonstrating initial support for GNDS-JSON formatted nuclear data.

REFERENCES

- [1] Cross Section Evaluation Working Group, ENDF-6 Formats Manual - Data Formats and Procedures for the Evaluated Nuclear Data Files ENDF/B-VI, ENDF/B-VII and ENDF/B-VIII, Edited by A. Trkov, M. Herman and D. A. Brown, CSEWG Document ENDF-102, Report BNL-203218-2018-INRE, 2018.
- [2] NEA WPEC Subgroup 38, Specifications for the Generalised Nuclear Database Structure (GNDS), www.oecd-nea.org/jcms/pl_39689/specifications-for-the-generalised-nuclear-database-structure-gnds (accessed on August 15, 2023).
- [3] C. Liu, Proposal for GNDS-JSON Format with Reduced Nuclear Data Structural Complexity. Proceedings of the 30th International Conference on Nuclear Engineering (ICONE30). May 21-26, 2023. Kyoto, Japan.
- [4] H. Wim, M. F. Staley, and N. A. Gibson. The NJOY update for GNDS 2.0 and lessons learned for GNDS 3.0, WPEC December 2021 Subgroup Meetings, 2021-12-07/2021-12-10 (Paris, France), LA-UR-21-32005, 2021.
- [5] Y. Huang, T. Zu, L. Cao, and H. Wu, Capability of processing the GNDS format evaluated nuclear data in NECP-Atlas for neutronics calculations, Progress in Nuclear Energy, vol. 150, pp. 104293, August, 2022.
- [6] B. Beck, and USDOE National Nuclear Security Administration, General Interaction Data Interface. Computer software, <https://www.osti.gov/servlets/purl/1542552>. Vers. 3.17. USDOE National Nuclear Security Administration (NNSA). Web. doi:10.11578/dc.20190715.7. 11 Mar. 2019.
- [7] D. Wiarda, A. Holcomb, G. Arbanas, and J. Brown, Status of GNDS support in AMPX/SAMMY and future outlook, Consultancy Meeting on model code output & application

nuclear data form structure. Web. doi.org/10.2172/1878201. Mar. 15, 2021.

[8] N. Fujimoto, K. Tada, H. Ho, S. Hamamoto, S. Nagasumi, and E. Ishitsuka. *Nuclear data processing code FRENDY: A verification with HTTR criticality benchmark experiments*, Annals of Nuclear Energy, vol. 158, pp. 108270, 2021

[9] J. Yu, S. Li, K. Wang, G. Wang, and G. Yu, The Development and Validation of Nuclear Cross Section Processing Code for Reactor-RXSP, 21st International Conference on Nuclear Engineering, chengdu. doi:10.1115/ICONE21-15442. July 29–August 2, 2013.

[10] K. Hu, X. Ma, Y. Huang, and etc., Development and Verification of a New Nuclear Data Processing Code AXSP, Frontiers in Energy Research 10, September, 2022.

[11] M. Coste-Delcaux, C. Jouanne, F. Moreau, and C. Mounier, GALILÉE-1: a validation and processing system for ENDF-6 and GND evaluations, The European Physical Journal Conferences 111:06005. doi:10.1051/epjconf/201611106005. January, 2016.

[12] C. M. Mattoon, B. R. Beck, N. R. Patel, N. C. Summers, G. W. Hedstrom, and D. A. Brown, Generalized Nuclear Data: A New Structure (with Supporting Infrastructure) for Handling Nuclear Data, Nuclear Data Sheets, Volume 113, Issue 12, December 2012, Pages 3145-3171, 2012.

[13] Python Software Foundation, www.python.org. (accessed on August 15, 2023).

[14] Julia Programming Language, julia.org. (accessed on August 15, 2023).

[15] PyCall.jl Contributors, Calling Python functions from the Julia language. Open-source repository, github.com/JuliaPy/PyCall.jl. (accessed on August 15, 2023).

[16] P. K. Romano, N. E. Horelik, B. R. Herman, A. G. Nelson, B. Forget, and K. Smith, OpenMC: A State-of-the-Art Monte Carlo Code for Research and Development, Ann. Nucl. Energy, Vol 82, p.90-97, 2015.

[17] The HDF5 Group, HDF5: High-performance data management and storage suite. www.hdfgroup.org/solutions/hdf5. (accessed on August 15, 2023)

[18] A. Hébert. PyNjoy-2012: A system for producing cross-section libraries for the DRAGON lattice code. International Conference on Nuclear Data for Science and Technology: ND2016.

[19] S. Furuhashi. MessagePack, msgpack.org (accessed on August 15, 2023)

[20] GIDI+ Contributors, C++ libraries for accessing nuclear data from the Generalized Nuclear Database Structure (GNDS). Open-source repository, github.com/LLNL/gidiplus. (accessed on August 15, 2023).

[21] GNDS-JSON Contributors, GNDS: a proposal for the GNDS data in the JSON format. Open-source repository, [jihulab.com/newcomputelab/gnds-json](https://github.com/newcomputelab/gnds-json). (accessed on August 15, 2023).

[22] CxxWrap.jl Contributors, Package to make C++ libraries available in Julia. Open-source repository, github.com/JuliaInterop/CxxWrap.jl. (accessed on August 15, 2023).

[23] PyBind11 Contributors, Seamless operability between C++11 and Python. Open-source repository, github.com/pybind/pybind11. (accessed on August 15, 2023).

[24] FrendyPlus Contributors, An extensible nuclear data processing code interfacing Frendy. Open-source repository, [jihulab.com/newcomputelab/frendyplus](https://github.com/newcomputelab/frendyplus). (accessed on August 16, 2023).

[25] K. Tada, Handling of GNDS in FRENDY and our recent activity, IAEA Consultancy Meeting on

GNDS/FUDGE/TAGNDS, May 22-25, 2023. Vienna, Austria. conferences.iaea.org/event/352/contributions/27278/

(accessed on August 15, 2023).

[26] R. Mosteller, The Doppler-Defect Benchmark: Overview and Summary of Results, Joint International Topical Meeting on Mathematics & Computation and Supercomputing in Nuclear Applications (M&C + SNA 2007), Monterey, California, April 15-19, 2007.

[27] Boost Contributors, Super-project for modularized Boost. Open-source repository, github.com/boostorg/boost. (accessed on August 15, 2023).

[28] Msdirect Contributors, C/C++ library for retrieving information on files and directories. Open-source repository, github.com/tronkko/direct. (accessed on August 15, 2023).