2026
IJS-DP: 12881

VERITAS

Visual Explorer for Real vs. Idealized neutron Transport and Analytic Solutions

DEVELOPER MANUAL

Jan Kokot, Luka Snoj

Table of contents

1. Introduction

The VERITAS (Visual Explorer for Real vs. Idealized neutron Transport and Analytic Solutions) simulator is a high-fidelity computational tool designed to model the process of neutron moderation across a variety of media and environmental conditions. By leveraging the OpenMC Monte Carlo particle transport code [1], VERITAS provides users with an intuitive yet rigorous platform for visualizing the stochastic nature of neutron interactions.

The software is primarily intended as an educational resource for students learning nuclear physics fundamentals, as well as a versatile tool for entry-level research and preliminary design studies.

The simulator is documented in three manuals:

- User Manual – Describing the operation of the simulator from the user's perspective.
- Physics Manual – Describing the underlying physical models and nuclear data processing.
- Developers Manual – Description of the software architecture and integration of the Dash/OpenMC framework.

This document is the Developer Manual for the VERITAS simulator.

2. Installation guide

The VERITAS installation process is designed to configure both the graphical interface and the underlying containerized OpenMC environment.

2.1. Prerequisites

Before executing the installer, the host system must meet the following software requirements. Without these components, the application backend will fail to initialize:

- **WSL2 (Windows Subsystem for Linux) [2]:** Required for Linux binary execution on Windows.
- **Docker Desktop [3]:** Necessary for managing the containerized OpenMC engine [1]. Docker will start automatically when the VERITAS app is launched.

2.2. Installation Procedure

The installation is handled via a dedicated `.exe` installer created with **Inno Setup** [4], which is available on the official VERITAS website. The process follows several configuration stages:

1. **Directory Selection:** The user defines the primary installation path for the application binaries (approx. 790 MB).

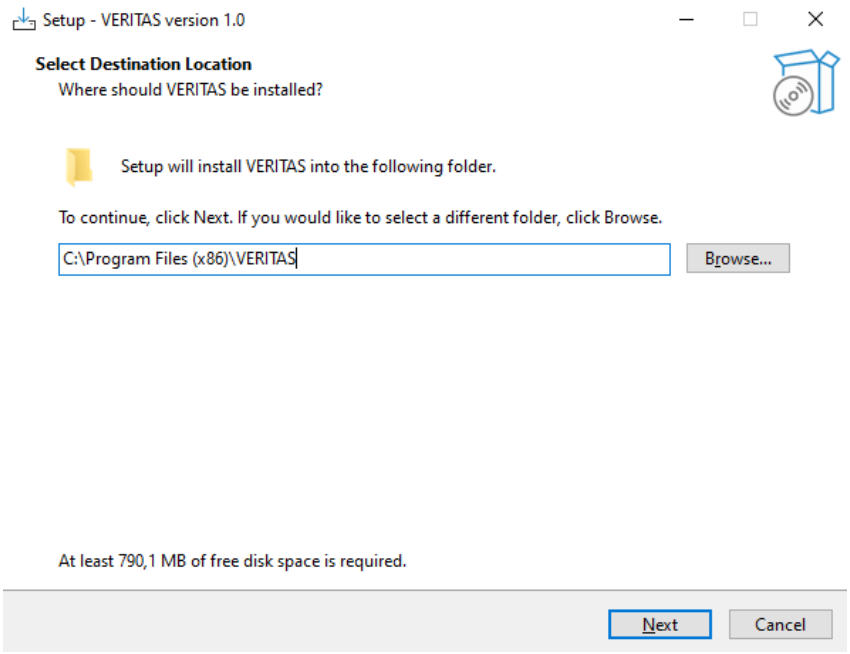

Figure 1: Installation path selection in Inno Setup.

2. **Nuclear Data (Cross-Sections):** The user specifies the location of existing ENDF/B-VIII cross-section data.
    - If no data is available, the user may opt to download a minimal library from the VERITAS website (approx. 4.5 GB) or proceed without global libraries, relying solely on custom-generated cross-sections[cite: 16]. Global libraries can be added later via the application settings.
3. **Workspace and Quotas:** The user defines the storage path for `setups` and `custom_materials`[cite: 19, 20].
    - Default path: `/openmc_setups/local_user/` within the installation directory.
    - The user must define a storage quota (in MB) to cap the maximum disk space utilized by simulation results and custom materials.
4. **Port Selection:** The user chooses the port on which the app's local server will run. The default port is 8300.
5. **Shortcuts:** Options are provided to generate shortcuts for the Desktop and the Start Menu.
6. **Docker Image Deployment [3]:** Upon confirming the installation summary, the installer initiates a PowerShell script to load the **VERITAS Docker image**.

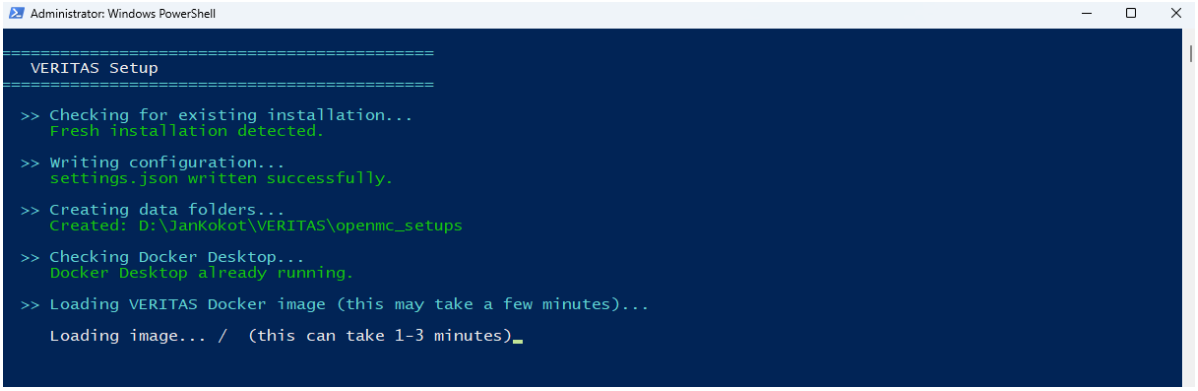

Figure 2: Docker image installation.

Critical Note: The PowerShell window must not be closed or interrupted, as this process registers the OpenMC environment with the local Docker daemon.

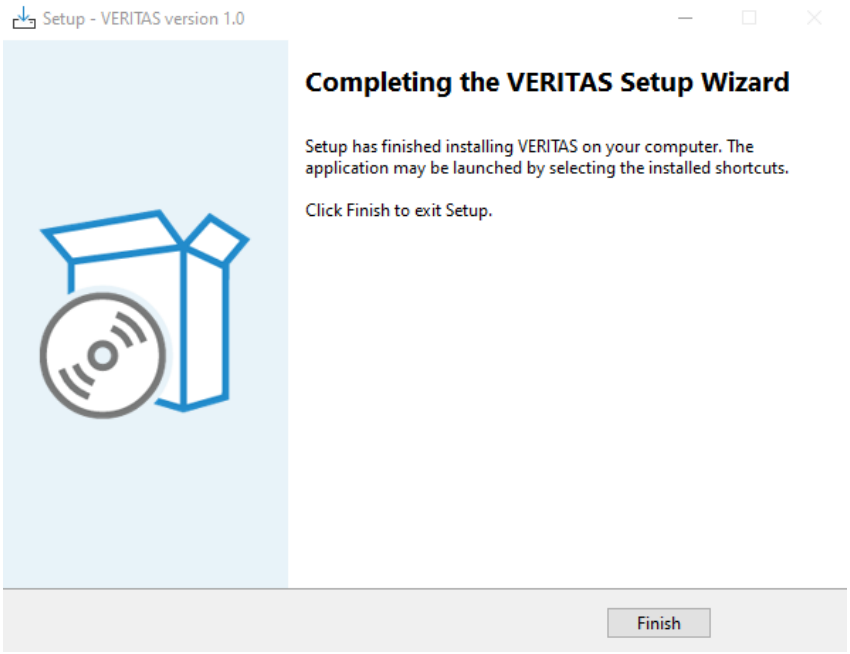

Figure 3: Installation finished screen.

2.3. Uninstallation

To remove the application, users must execute the `unins000.exe` utility located in the root installation directory. The uninstaller performs the following actions:

- Removes all program binaries and associated assets.
- Deletes the **VERITAS Docker image** from the system to recover disk space.
- **Optional Data Persistence:** The user is prompted to decide whether their local `setup` data (simulations and custom materials) should be preserved or permanently deleted.

3. Program Flow

The application utilizes a unified Python-based architecture for both the backend logic and frontend presentation. The interface is developed using **Plotly Dash** [5], which serves as a high-level alternative to standard HTML, CSS, and JavaScript, allowing for a fully reactive web-based environment.

3.1. Tech Stack and Build Logic

- **UI Layer:** Built with `Dash` and `Dash Bootstrap Components`. It manages the application layout and handles user interactions through a series of reactive callbacks.
- **Transport Engine:** OpenMC [1] executes the physics simulations. It is invoked as an asynchronous subprocess within a containerized Docker environment to ensure OS-level compatibility.
- **Data Processing:** For custom materials, the `pydecs` [6] module generates the necessary nuclear data tapes. These are then processed by **NJOY** [7] (using `RECONR` and `BROADR`) to produce ACE-formatted cross-sections.
- **Serialization:** Trajectory data produced by OpenMC (initially in `tracks.h5` format) is processed by the `data_creation` module. To optimize performance, this data is cached into Pickle files, significantly reducing latency during subsequent analysis and graphing.
- **Graphing:** All interactive visualizations are generated in the `graphing.py` module using Plotly and are seamlessly integrated into the Dash frontend.

3.2. Execution Workflow

The simulation pipeline follows a strictly sequenced data flow:

1. **Cross-Section Preparation:** Optionally generate virtual ACE files using the `pydecs` pipeline or select standard isotopes from the **ENDF/B-VIII** library[cite: 1, 2, 5].
2. **XML Export:** UI configurations are exported into OpenMC-compliant `geometry.xml`, `materials.xml`, and `settings.xml` files[cite: 1, 2, 3].
3. **Binary Execution:** The backend triggers the `openmc --track` command[cite: 2, 3]. Batch progress is parsed from the standard output and streamed to the UI in real-time using a periodic interval trigger.
4. **Caching Strategy:** To bypass the performance limitations of repeated HDF5 parsing, raw trajectory data is serialized into a `tracks.pkl` file. While the initial conversion adds a minor overhead to the first simulation run, subsequent visualization and filtering tasks are executed nearly instantaneously.

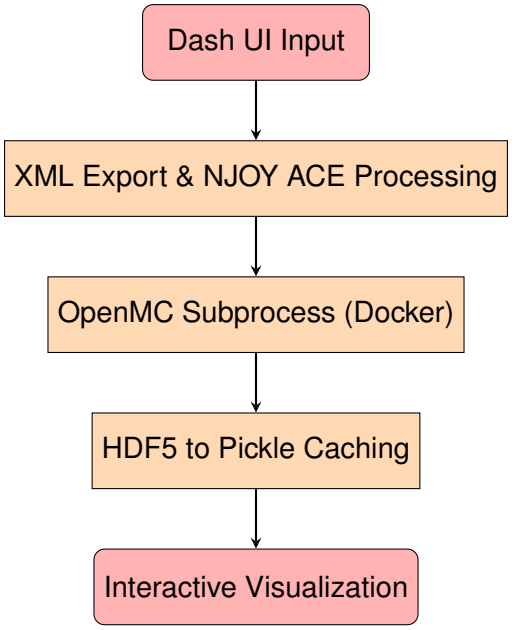

Figure 4: The high-level program flow of the VERITAS simulator.

4. The `pydecs.py` Module: Idealized nuclide generation

The `pydecs.py` module (recreated from the Fortran code `idecs.f90`, original author L. Plevnik [6]) serves as the idealized cross-section data creator for the VERITAS simulator. Its primary purpose is to generate ENDF-6 formatted tapes and NJOY [7] input decks based on idealized theoretical parameters rather than relying on pre-existing nuclear data libraries.

4.1. Multi-Pass Writing Strategy

The ENDF-6 format requires that the "File 1" metadata (MF1, MT451) contains an accurate directory of the total number of records (lines) in every subsequent section. To satisfy this requirement, the module implements a two-pass execution logic within the `_write_endf` function:

- **Pass 0 (Dry Run):** The module executes all generation functions with the internal write-flag (`_s["wro"]`) set to `False`. During this pass, it calculates the exact line counts for every MF/MT section.
- **Pass 1 (Emission):** After the counts are finalized, the module runs a second time with the write-flag set to `True`. It uses the counts from Pass 0 to populate the headers and emits the formatted 80-character strings into an ASCII buffer.

4.2. Fixed-Width Formatting: `cxfp`

The most critical low-level component of the module is the `cxfp` function. The ENDF-6 standard mandates that floating-point numbers must fit exactly into 11-character columns. Standard Python `f-strings` cannot handle the unique requirements of this format (e.g., omitting the 'e' in scientific notation to save space). The `cxfp` logic:

- Forces a specific mantissa length depending on the magnitude of the exponent.
- Manages the sign bit and ensures that the total width never exceeds 11 characters.
- Converts values such as `0.0253` into the rigid `2.530000-2` format required by NJOY and OpenMC.

4.3. Theoretical Physics Implementation

The module uses analytical models to derive realistic nuclear properties for virtual isotopes:

- **Binding Energy (E_B):** The `_eb` function calculates the binding energy using the Semi-Empirical Mass Formula (Liquid Drop Model):

$$E_B(Z, A) = a_v A - a_s A^{2/3} - a_c \frac{Z^2}{A^{1/3}} - a_a \frac{(A - 2Z)^2}{A} + \delta \tag{1}$$

  where A is the atomic mass number and Z is the atomic number. This is used to compute reaction Q-values for radiative capture (MT102) and fission (MT18).

- **Stability Curve:** The `_compute_za_nz` function determines the most probable atomic number (Z) for a given Atomic Weight Ratio (AWR) using the nuclear stability valley approximation, ensuring the virtual isotope is physically plausible.

4.4. Cross-Section Modeling (MF3)

The module populates the "File 3" section of the ENDF tape with energy-dependent cross-sections ($\sigma(E)$). It supports three primary idealized behaviors:

1. **Generalized $1/v$ Law:** Modeled as $\sigma(E) = \sigma_0 \left(E_{\min}/E\right)^{\alpha}$, where $\alpha = 0.5$ recovers the classical $1/v$ absorption law. The exponent α is a configurable parameter.
2. **Constant Cross Section:** A flat cross-section profile independent of energy.
3. **Absorption Windows:** Used for the `scatterer_abs` material type. The `_tab1_abs_windows` function defines energy intervals in which absorption is elevated, using duplicate x-points to enforce sharp edges under linear-linear interpolation.

4.5. Supported Material Types

The `generate()` function accepts a `material_type` argument that selects the physical model used to construct the ENDF tape. Six idealized material types are supported, each mapping to an internal type index `nt` and a fissile flag `lfi`:

`absorber` **($nt = 1$):** A pure neutron absorber with no scattering resonances. The total, elastic, and radiative capture cross-sections (MT1, MT2, MT102) follow either a generalized $1/v$ law, a constant profile, or a step function, selected via the `xs_shape` parameter. The NJOY pipeline applies full RECONR and BROADR processing.

`scatterer` **($nt = 3$):** A pure elastic scatterer with a flat, potential-scattering cross-section $\sigma = \pi a_p^2$, where a_p is the scattering radius. No resonance region is defined (LRU$= 0$ in MF2), so RECONR passes MF3 through unchanged.

`resonance` **($nt = 2$, $lfi = 0$):** A resonance absorber and scatterer. MF2 contains explicit Breit-Wigner resonance parameters (energy E_r, spin J, partial widths Γ_n, Γ_γ). RECONR reconstructs the pointwise cross-section from these parameters and BROADR applies Doppler broadening.

`fissile` **($nt = 2$, $lfi = 1$):** Extends `resonance` with fission channels. Adds MT452 (prompt neutron multiplicity $\bar{\nu}$), MT18 (fission cross-section), and MT18 in MF5 (Watt fission energy spectrum with configurable parameters E_{fl} and E_{fh}). Additional partial widths Γ_{fa} and Γ_{fb} are included in the resonance table.

`scatterer_res` **($nt = 4$):** An elastic scatterer with explicit resonance parameters in MF2 (LRU$=1$, LRF$=2$). The potential cross-section baseline is derived from the scattering radius. Full RECONR and BROADR processing applies.

`scatterer_abs` **($nt = 5$):** An elastic scatterer with configurable absorption windows. MF2 is set to LRU$=0$, causing RECONR to skip resonance reconstruction entirely and pass MF3 through as the final cross-section. Absorption is non-zero only within user-defined energy intervals $(E_{\text{low}}, E_{\text{high}}, \sigma_{\text{abs}})$, specified via the `abs_windows` parameter. BROADR is omitted as Doppler broadening of step-function cross-sections is not physically meaningful.

Table 1 summarizes the ENDF sections generated for each material type.

Table 1: ENDF sections generated per material type.

Type	nt	MF1/451	MF1/452	MF2/151	MF3	MF4	MF5/18	BROADR
`absorber`	1	✓	–	LRU=1	1,2,102	✓	–	✓
`scatterer`	3	✓	–	LRU=0	1,2	✓	–	✓
`resonance`	2	✓	–	LRU=1	1,2,102	✓	–	✓
`fissile`	2	✓	✓	LRU=1	1,2,18,102	✓	✓	✓
`scatterer_res`	4	✓	–	LRU=1	1,2	✓	–	✓
`scatterer_abs`	5	✓	–	LRU=0	1,2,102	✓	–	–

Only the `absorber`, `scatterer` and `scatterer_abs` material types are used in **VERITAS**.

4.6. NJOY Integration (`acer_inp`)

Finally, the `_write_acer_inp` function generates the instruction set for NJOY. This script automates the transformation of the virtual ENDF tape into a production-grade ACE library for OpenMC using the following sequence:

- **RECONR:** Reconstructs resonance cross-sections into pointwise form from resonance parameters.
- **BROADR:** Applies Doppler broadening to the specified simulation temperature (e.g., 300 K). This step is omitted for `scatterer_abs` materials (`nt=5`), as Doppler broadening corrupts the generated step function even at low temperatures.
- **ACER:** Generates the final ACE-formatted file and `xsdir` entries for use by OpenMC.

5. File and Folder Structure

The project is architected to enforce a strict separation between the UI framework, transport logic, and runtime assets. This modularity ensures that the physics engine remains decoupled from the presentation layer.

- **Core Application Logic:**
  - `app.py`: The primary entry point for the Dash [5] application. It initializes the Flask server, defines the root layout, and configures the environment, including paths for NJOY [7] and the global `OPENMC_CROSS_SECTIONS`. The default application port is 8300 but it can be changed in the installation menu and later on in the app.
  - `data_creation.py`: Manages the interface between the UI and OpenMC [1]. It contains logic for dynamically building geometry and materials, managing the OpenMC subprocess, and executing the HDF5-to-Pickle serialization pipeline.

  - `pydecs.py`: The module responsible for generating ENDF-6 tapes and NJOY input decks for idealized material models.

- **Frontend and Visualization:**
  - `graphing.py`: A library of functions for generating interactive Plotly visualizations, including 3D particle tracks, energy-loss heatmaps, and spectral distributions.
  - `graphing_callbacks.py`: Manages the reactive logic required to update visualizations based on user-defined filters, such as particle selection or thermal energy cutoffs.
  - `layout_callbacks.py`: Handles high-level UI orchestration, including page navigation and modal toggles.
  - `user_layout.py`: Defines the structural framework and Bootstrap-based components for the application interface.

- **Utility and Helper Modules:**
  - `helper.py`: Contains general-purpose functions for safety checks, directory management, and unit conversion (e.g., slider-to-eV mappings).
  - `input_utilities.py`: Implements robust parsing logic to convert user-supplied strings (e.g., particle ranges) into functional data structures for the backend.
  - `layout_utilities.py`: Provides a library of custom reusable UI components and standardized messaging alerts.
  - `njoy_utilities.py`: Specialized helpers for managing the NJOY processing lifecycle and temporary file storage.

- **Data Storage and Assets:**
  - `/openmc_setups/`: A dynamic workspace where each simulation instance is assigned a unique directory to store its XML configurations, binary results, and cached Pickle data[cite: 1, 3].
  - `/assets/`: Houses static resources, including custom CSS stylesheets, additional JavaScript for UI enhancements, and images/icons used throughout the application[cite: 1, 3].

5.1. Installer Package Contents

The VERITAS executable is a compressed archive compiled using **Inno Setup** [4] that bundles all the necessary components to deploy the containerized simulation environment on a host system. The following files are packaged within the installer and deployed to the root installation directory:

- `veritas.tar`**:** The primary payload of the distribution. This is a pre-built Docker [3] image containing the full Ubuntu 24.04 runtime, compiled OpenMC and NJOY binaries, and the Python Dash application stack.

- `setup.ps1`**:** A post-installation PowerShell script. Its primary role is to initialize the environment by generating the `settings.json` file and executing the `docker load` command to register the image with the host's Docker daemon.

- `VERITAS.bat`**:** The script used to launch the application. It performs pre-launch checks for Docker Desktop, reads user-defined paths from `settings.json`, and handles the `docker run` command with appropriate volume mounts.

- `VERITAS.vbs`**:** Hides the Command Prompt window that is created by the `VERITAS.bat`.
- `veritas_loading.html`**:** A standalone HTML and JavaScript interface used as a loading screen. It automatically redirects the browser to the local server once the application is ready.
- `favicon.ico`**:** The application's visual asset used for Windows desktop and Start Menu shortcuts.
- `settings.json`**:** Although not directly bundled, this file is generated by the `setup.ps1` script during installation to store persistent user configurations, such as the cross-section library path and simulation data storage limits.
- `unins000.exe`**:** The uninstallation executable. After double-clicking it takes the user through the uninstall process described in the Uninstallation subsection 2.3.
- `unins000.dat`**:** The uninstallation database. It records everything the installer did so it can be undone cleanly.

References

[1] OpenMC Development Team. *OpenMC Documentation*, 2024. Version 0.15.0; Git Commit 26b1308.

[2] Microsoft Corporation. *Windows Subsystem for Linux Documentation*, 2026.

[3] Docker Inc. *Docker Desktop Documentation*, 2024.

[4] Jordan Russell and Martijn Laan. *Inno Setup Documentation*, 2024.

[5] Plotly Technologies Inc. *Dash Python Framework*, 2024.

[6] Lucijan Plevnik, Gašper Žerovnik, Bor Kos, and Luka Snoj. A computer code for generating idealized nuclear data. IJS delovno poročilo 12357, Inštitut Jožef Stefan, Ljubljana, 2017. 79 str. COBISS.SI-ID 31254823.

[7] Los Alamos National Laboratory. *NJOY2016: Nuclear Data Processing System*, 2016.