Computer Physics Communications 269 (2021) 108123

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

Simulating low-energy neutrino interactions with MARLEY ☆,☆☆

Steven Gardiner [a,b,*]

[a] Fermi National Accelerator Laboratory, P.O. Box 500, Batavia, IL 60510, USA
[b] University of California, Davis, One Shields Avenue, Davis, CA 95616, USA

ARTICLE INFO

Article history:
Received 3 February 2021
Received in revised form 23 June 2021
Accepted 12 July 2021
Available online 4 August 2021

Keywords:
Event generator
Neutrino-nucleus scattering
Tens-of-MeV

ABSTRACT

Monte Carlo event generators are a critical tool for the interpretation of data obtained by neutrino experiments. Several modern event generators are available which are well-suited to the GeV energy scale used in studies of accelerator neutrinos. However, theoretical modeling differences make their immediate application to lower energies difficult. In this paper, I present a new event generator, MARLEY, which is designed to better address the simulation needs of the low-energy (tens of MeV and below) neutrino community. The code is written in C++14 with an optional interface to the popular ROOT data analysis framework. The current release of MARLEY (version 1.2.0) emphasizes simulations of the reaction $^{40}\mathrm{Ar}(\nu_e, e^-)^{40}\mathrm{K}^*$ but is extensible to other channels with suitable user input. This paper provides detailed documentation of MARLEY's implementation and usage, including guidance on how generated events may be analyzed and how MARLEY may be interfaced with external codes such as Geant4. Further information about MARLEY is available on the official website at http://www.marleygen.org.

Program summary

Program title: MARLEY 1.2.0
CPC Library link to program files: https://doi.org/10.17632/4v7zxnc8j3.1
Developer's respository link: http://github.com/MARLEY-MC/marley
Code Ocean capsule: https://codeocean.com/capsule/9868179
Licensing provisions: GNU General Public License 3.0
Programming language: C++14
External routines/libraries used: GNU Scientific Library [1,2] (required), ROOT [3,4] (optional)
Nature of problem: Simulation of neutrino-nucleus scattering events at energies of tens-of-MeV and below
Solution method: Initial two-to-two scattering kinematics are sampled using the allowed approximation differential cross section and tables of precomputed nuclear matrix elements. Subsequent de-excitations of the remnant nucleus are simulated using a Monte Carlo implementation of the Hauser-Feshbach statistical model and tabulated γ-ray decay schemes for discrete nuclear levels.
Additional comments including restrictions and unusual features: Input data are provided with the code that are suitable for producing simulations of the charged-current reaction $^{40}\mathrm{Ar}(\nu_e, e^-)^{40}\mathrm{K}^*$, coherent elastic neutrino-nucleus scattering on spin-zero target nuclei, and neutrino-electron elastic scattering on any atomic target. Preparation of new reaction input files (whose format is documented in Appendix B) would enable other reaction channels and nuclear targets to be handled by the existing code framework. Although there is no maximum neutrino energy enforced by the code itself, realistic neutrino-nucleus scattering events may be generated up to roughly 50 MeV. Above this energy, the effects of forbidden nuclear transitions, which are neglected in the current treatment of the cross sections (see section 2.1), become increasingly important.

☆ The review of this paper was arranged by Prof. Z. Was.
☆☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Correspondence to: Fermi National Accelerator Laboratory, P.O. Box 500, Batavia, IL 60510, USA.
E-mail address: gardiner@fnal.gov.

https://doi.org/10.1016/j.cpc.2021.108123

1. Introduction

Monte Carlo event generators are a widely-used tool in nuclear and particle physics. These computer programs implement probabilistic models of physics processes and produce corresponding sets of *events*: lists of particles (represented by their charges, 4-momenta, etc.) involved in simulated interactions.

While helpful as an aid to theoretical calculations, event generators are also often used by experimentalists for designing detectors, estimating efficiencies and backgrounds, assessing systematic uncertainties, and interpreting the results of measurements. Examples of event generators include PYTHIA [5] and Herwig [6] for high-energy particle collisions, HYDJET++ [7] for relativistic heavy ion collisions specifically, FREYA [8] for fission, and DECAY4 [9] for radioactive decays of unstable isotopes.

For studies of neutrinos, much community effort has been directed toward the development of event generators suitable for use by accelerator-based experiments at facilities like J-PARC [10] and Fermilab [11]. These experiments employ beams of primarily muon-flavor neutrinos which are produced over a broad energy range between the low hundreds of MeV to the tens of GeV. Neutrino scattering on atomic nuclei is the primary means of detection, and several modern event generators provide widely-used models of the relevant physics, including GENIE [12], GiBUU [13], NEUT [14], and NuWro [15].

Although important differences exist between each of these generators, all share a similar conceptual treatment of neutrino-nucleus interactions. Each scattering event on a complex nucleus is taken to involve a neutrino striking a single bound nucleon.[1] A removal energy and initial 3-momentum are assigned to the struck nucleon using a model of the nuclear ground state. Traditionally this is done using a variant of the Fermi gas model (e.g., that of Ref. [17]), but implementations of more sophisticated treatments, such as the Correlated Basis Function approach [18,19], are beginning to become available [20].

With the initial state fully defined, the generator then simulates production of the particles that emerge from the neutrino-nucleon interaction vertex. A variety of models must be used at this stage due to competition between multiple nucleon-level processes (e.g, quasi-elastic and deep inelastic scattering) which may occur at accelerator neutrino energies.

Because they are subject to the strong force, outgoing hadrons from the primary neutrino interaction will often rescatter within the nuclear medium. These *final-state interactions* (FSIs) can have a pronounced effect on the kinematics and multiplicities of the hadrons that ultimately exit the nucleus. Intranuclear hadron transport and FSIs are handled in GENIE, NEUT, and NuWro using variations of the intranuclear cascade (INC) model [21–24]. This model assumes that propagation of a hadron h through the nucleus may be described in terms of a mean free path

$$\lambda(h, E, r) = \frac{1}{\rho(r)\,\sigma_{hN}(E)}\,, \tag{1}$$

where E is the hadron energy, r is its radial position within the nucleus, and ρ is the number density of nucleons. The total cross section for h scattering on a free nucleon, σ_{hN}, may be used directly but is typically modified with approximate corrections for nuclear effects.

The GiBUU code simulates intranuclear hadron transport using a semi-classical model which considers the time evolution of the phase space density for each hadronic species. The equations describing the behavior of distinct kinds of hadrons are coupled through the common nuclear mean field and through a collision term which represents the influence of FSIs. The numerical implementation adopts a test-particle ansatz to solve the relativistic Boltzmann-Uehling-Uhlenbeck (BUU) equation. Extensive documentation of GiBUU's treatment of various nuclear reactions, including neutrino-nucleus scattering, is available in refs. [13,25]. The INC models used by other generators for FSIs may largely be regarded as approximate simplifications of the BUU approach [26, sec. 2].

When all produced particles either escape the nucleus or are re-absorbed by it, the hadron transport stage of the simulation is complete. This typically marks the end of the physics workflow needed to generate a single neutrino-nucleus scattering event. The finished events may be used both for standalone calculations of kinematic distributions and as input to later stages of an experiment's detector simulation chain.[2]

In addition to accelerator-based neutrino oscillation experiments, there is also worldwide interest in pursuing detailed measurements of lower-energy (tens-of-MeV and below) neutrinos produced by supernovae [28–30], by the Sun [31,32], and by terrestrial facilities via pion and muon decays at rest [33–37]. Several distinct reaction modes will enable neutrino detection in these measurements. Elastic scattering on electrons and protons [38] is flavor-blind but produces signal events down to arbitrarily low neutrino energies. Coherent elastic neutrino-nucleus scattering (CEvNS), a neutral-current (NC) process in which a neutrino scatters off of a complex nucleus and leaves it in its ground state [39], shares these properties and was recently observed for the first time by the COHERENT experiment [40,41]. Apart from reactions involving complex nuclei, the only inelastic channel available at tens-of-MeV energies is charged-current (CC) absorption of $\bar{\nu}_e$ on free protons via the inverse beta decay (IBD) reaction[3]

$$\bar{\nu}_e + p \to n + e^+\,. \tag{2}$$

The IBD cross section is precisely known [42,43] and dominates the expected signal for supernova neutrinos in water Cherenkov and liquid scintillator detectors [29].

The low-energy neutrino interaction modes listed so far do not present any special difficulties from an event generator perspective. Relatively simple expressions are adequate to compute differential cross sections for all but the most precise calculations of these processes, and, with the recent addition of a CEvNS model [44], the GENIE event generator currently provides an implementation of all of them that may be suitable for use by low-energy neutrino experiments.

[1] Two particle-two hole interactions, which involve a pair of nucleons, are included in modern generators [16] but neglected here for simplicity. I likewise neglect coherent processes, which involve the nucleus as a whole.

[2] Ref. [27] describes a GENIE-based example used by the NOvA experiment.

[3] The analogous reaction $\nu_e + n \to p + e^-$ is also possible. However, it is not practical for use in a neutrino detector due to the instability of free neutrons.

In contrast to other low-energy processes, inelastic neutrino scattering on all but the lightest complex nuclei (e.g., deuterium) is theoretically cumbersome, requiring an elaborate treatment of nuclear physics in order to fully describe the interactions. Despite the significant challenges involved, however, obtaining realistic simulations of low-energy inelastic neutrino-nucleus scattering is highly desirable for a variety of scientific applications. Principal among these is detection of low-energy astrophysical neutrinos by the upcoming Deep Underground Neutrino Experiment (DUNE) [45]. Thanks to the experiment's planned use of four ten-kiloton liquid argon time projection chambers (LArTPCs), DUNE has the potential to perform detailed measurements of these neutrinos via the charged-current reaction

$$\nu_e + {}^{40}\mathrm{Ar} \rightarrow e^- + {}^{40}\mathrm{K}^* \,. \tag{3}$$

This process is anticipated to provide most of the signal in DUNE for the neutrino burst associated with a galactic core-collapse supernova, granting the experiment the potential for unique sensitivity among large detectors to the ν_e component of the supernova neutrino flux [46,47]. DUNE also shows substantial promise as a detector for studying solar neutrinos [31].

Primary sensitivity to low-energy ν_e is shared with DUNE by the 79-ton Helium and Lead Observatory (HALO) [48–50], but in this case the detection technique is indirect. Inelastic neutrino reactions on lead nuclei, such as

$$\nu_e + {}^{208}\mathrm{Pb} \rightarrow e^- + {}^{208}\mathrm{Bi}^* \,, \tag{4}$$

will sometimes lead to the production of neutrino-induced neutrons (NINs) via de-excitations of the residual nucleus. HALO's lead neutrino target is instrumented with ^{3}He-based neutron counters, which will be used to search for NINs in the event of a nearby core-collapse supernova. Due to its widespread use as a radiation shielding material, lead is also a potential source of background NINs in precision CEvNS measurements. To better constrain theoretical modeling of this background, the COHERENT experiment is pursuing direct measurements of NIN production on lead, iron, and copper [36].

Beyond the examples mentioned here, inelastic scattering on a variety of other nuclear targets is of interest either as a means of low-energy neutrino detection[4] or as a source of background. Additional nuclei for which models of these processes have been studied in detail include carbon [52–57], oxygen [58–62], molybdenum [63–68], and xenon [69–71], among others [72–79].

Despite the successes of standard neutrino event generators in describing accelerator neutrino data, their prevailing treatment of inelastic neutrino-nucleus scattering is likely to be inadequate when applied to interactions at energies of tens of MeV and below. This is due in part to approximations made in their modeling of nuclear structure. For few-MeV neutrinos, inelastic neutrino-nucleus cross sections are governed by the energetically-accessible transitions to low-lying discrete energy levels of the daughter nucleus. At somewhat higher energies, excitations of collective vibrational modes of the nucleus, known as *giant resonances* [80,81], also begin to play a major role. Both of these details are entirely missing from the conventional Fermi gas model of the nuclear ground state. While these deficiencies of the Fermi gas model become less problematic as the neutrino energy increases to several hundred MeV and beyond, it has been pointed out that low-energy nuclear excitations are still expected to exert a noticeable influence on $\sim$1 GeV neutrino-nucleus differential cross sections at very forward scattering angles [82,83].

A second modeling difference that limits the suitability of typical neutrino event generators for the low-energy regime is the description of hadronic final-state interactions. Rather than taking an INC- or BUU-like dynamical approach to FSIs, in which intranuclear scattering is explicitly modeled, typical low-energy calculations [61,84–91] opt instead for a statistical treatment, in which only bulk properties of the nuclear system (such as its excitation energy and spin-parity) are employed to predict its behavior. The latter strategy is usually justified by assuming that the struck nucleon from the primary neutrino interaction will scatter repeatedly within the nuclear medium without being directly knocked out. These multiple intranuclear collisions lead to the transferred energy being widely shared among the constituent nucleons and thus to thermal equilibration: de-excitations of the resultant *compound nucleus* may be treated independently of the manner in which it was formed.

While theoretical predictions of neutrino-nucleus cross sections at high (low) energies tend to rely exclusively on a dynamical (statistical) model of FSIs, a more complete treatment may be achieved by combining the two approaches. The current release of GiBUU enables such calculations by providing an optional interface to the Statistical Multifragmentation Model (SMM) [92] code. Disintegration of the residual nucleus is simulated by SMM in a post-processing step which takes otherwise complete GiBUU events as input. Although the two codes have been used together to examine other processes [93–96], their joint application to neutrino interactions remains unstudied.

At present, native support for statistical nuclear de-excitation models in the other three neutrino generators mentioned above is limited to some simple approximations used in GENIE's treatment of nucleon emission [97]. However, official GENIE interfaces to INCL++ [98,99] and to the Bertini Cascade implementation [100] in Geant4 [101,102] are in the late stages of development [44]. These will provide enhancements to GENIE FSI modeling which are similar in scope to those obtained with SMM for GiBUU. Unofficial interfaces are also being explored by outside groups, with at least one attempt [103] having been made to apply the TALYS [104,105] nuclear de-excitation model to events generated using both GENIE and NuWro.

Although interfacing with these external tools provides an FSI treatment more compatible with standard low-energy approaches, a key omission remains problematic. With the exception of TALYS, the usual configurations of all of the remaining codes[5] lack a means of simulating discrete γ-ray transitions between low-lying nuclear energy levels.[6] Gamma-rays created in this way represent a major component of the final state for inelastic scattering of solar neutrinos on complex nuclei. For inelastic NC reactions, de-excitation γ-rays may often be the only final-state particles which are experimentally observable.

In this work, I present a new neutrino event generator, `MARLEY`,[7] which implements a model of inelastic neutrino-nucleus scattering designed specifically for the low-energy regime. The early version of `MARLEY` described herein is primarily focused on simulations of

[4] The pioneering Homestake solar neutrino experiment famously employed the $^{37}\mathrm{Cl}(\nu_e, e^-)^{37}\mathrm{Ar}$ reaction for this purpose [51].

[5] Initializing the main Geant4 Bertini Cascade class (`G4CascadeInterface`) using non-default settings [100, sec. 4.1] may enable simulation of discrete γ-ray emission via the `G4ExcitationHandler` class [106].

[6] I note, however, that both GENIE and NEUT directly implement simple models of de-excitation γ-ray production that are specific to ^{16}O.

[7] `MARLEY` is an acronym for *Model of Argon Reaction Low Energy Yields*. Although originally conceived as a tool to simulate the specific process $^{40}\mathrm{Ar}(\nu_e, e^-)^{40}\mathrm{K}^*$, `MARLEY` is moving toward becoming a more general-purpose low-energy neutrino interaction generator.

charged-current absorption of ν_e on ^{40}Ar [107]. However, preparation of additional input data would allow other reaction channels and nuclear targets to be handled without difficulty within the existing code framework. Section 2 provides an overview of the theoretical treatment of neutrino scattering used in MARLEY. Section 3 presents the MARLEY approach to random sampling, which makes ample use of modern improvements to the C++ language and, as discussed in section 3.3, implements a new inverse transform sampling algorithm [108] in a physics event generator for the first time. Section 4 discusses the MARLEY event generation workflow and implementation details. Sections 5–7 outline how to install, configure, run, and interpret the output of the code. Section 8 explains how MARLEY can be interfaced with external software toolkits, using the popular Geant4 [101,102] particle transport package as an example. Finally, section 9 considers prospects for future improvements to MARLEY.

2. MARLEY treatment of neutrino scattering

This section provides a brief overview of the physics models currently implemented in MARLEY, with more details available in Ref. [107]. Natural units ($\hbar = c = 1$) are used throughout.

As has been done in many previous calculations of low-energy neutrino cross sections [61,84–91], MARLEY treats neutrino-nucleus scattering events as proceeding via a two-step process. In the first step, a two-to-two scattering reaction involving the neutrino and the target nucleus is simulated, and the final nucleus is left in a state with a well-defined excitation energy, spin, and parity. In the second step, which is handled independently from the first, the final nucleus de-excites. At low excitation energies, where discrete level data are available, the de-excitations are modeled using tabulated γ-ray branching ratios. At higher excitation energies, where the final nucleus becomes unbound, the formation of a compound nucleus is assumed, and the decay widths for all open channels are calculated using the Hauser-Feshbach statistical model [109].

2.1. Nuclear $2 \to 2$ scattering model

To simulate two-to-two neutrino-nucleus scattering processes at low energies, MARLEY 1.2.0 evaluates the nuclear matrix elements in the long-wavelength limit (in which the four-momentum transfer $q \to 0$) and the slow-nucleon limit (in which $|\mathbf{p}_N|/m_N \to 0$, where $\mathbf{p}_N$ is the initial 3-momentum of the struck nucleon and m_N is its mass). The combination of these two limits is sometimes referred to as the *allowed approximation*. Under this approach, the differential cross section in the center-of-momentum (CM) frame for a transition to a particular nuclear final state is given by the expression

$$\frac{d\sigma}{d\cos\theta_\ell} = \frac{G_F^2}{2\pi}\mathcal{F}_{\mathrm{CC}}\left[\frac{E_i\,E_f}{s}\right]E_\ell\,|\mathbf{p}_\ell|\left[\left(1+\beta_\ell\cos\theta_\ell\right)B(\mathrm{F}) + \left(1-\frac{1}{3}\beta_\ell\cos\theta_\ell\right)B(\mathrm{GT})\right]. \tag{5}$$

Here G_F is the Fermi constant, Mandelstam s is the square of the total energy in the CM frame, and E_i (E_f) is the total energy of the initial (final) nucleus. The final-state lepton has total energy E_ℓ, 3-momentum $\mathbf{p}_\ell$, speed $\beta_\ell = |\mathbf{p}_\ell|/E_\ell$, and scattering angle θ_ℓ, which is defined with respect to the incident neutrino direction. The first factor in square brackets, E_iE_f/s, arises due to nuclear recoil and is often neglected. The symbol $\mathcal{F}_{\mathrm{CC}}$ is used to introduce extra factors needed when computing the cross section for charged-current scattering. Discussion of $\mathcal{F}_{\mathrm{CC}}$ is deferred to section 2.1.2.

The spin-reduced Fermi and Gamow-Teller nuclear matrix elements may be written in the form

$$B(\mathrm{F}) \equiv \frac{g_V^2}{2J_i+1}\left|\left\langle J_f \,\|\, \mathcal{O}_{\mathrm{F}} \,\|\, J_i\right\rangle\right|^2 \tag{6}$$

$$B(\mathrm{GT}) \equiv \frac{g_A^2}{2J_i+1}\left|\left\langle J_f \,\|\, \mathcal{O}_{\mathrm{GT}} \,\|\, J_i\right\rangle\right|^2 \tag{7}$$

where g_V (g_A) is the vector (axial-vector) weak coupling constant of the nucleon, and J_i (J_f) is the initial (final) nuclear spin. The Fermi matrix element $B(\mathrm{F})$ is subject to the spin-parity selection rule

$$B(\mathrm{F}) = 0 \text{ unless } J_f = J_i \text{ and } \Pi_f = \Pi_i \tag{8}$$

where Π_i (Π_f) is the initial (final) nuclear parity. The Gamow-Teller matrix element $B(\mathrm{GT})$ likewise obeys the selection rule

$$B(\mathrm{GT}) = 0 \text{ unless } |J_i - 1| \le J_f \le J_i + 1 \text{ and } \Pi_f = \Pi_i\,. \tag{9}$$

The Fermi ($\mathcal{O}_{\mathrm{F}}$) and Gamow-Teller ($\mathcal{O}_{\mathrm{GT}}$) operators from eqs. (6) and (7) are defined for charged-current and neutral-current scattering processes via the relations

$$\mathcal{O}_{\mathrm{F}} \equiv \begin{cases} \sum_{n=1}^{A} t_\pm(n) & \mathrm{CC} \\ Q_W/2 & \mathrm{NC} \end{cases} \qquad \mathcal{O}_{\mathrm{GT}} \equiv \begin{cases} \sum_{n=1}^{A} \boldsymbol{\sigma}(n)\, t_\pm(n) & \mathrm{CC} \\ \sum_{n=1}^{A} \boldsymbol{\sigma}(n)\, t_3(n) & \mathrm{NC} \end{cases} \tag{10}$$

where $\boldsymbol{\sigma}$ is the Pauli vector, A is the nucleon number, and the isospin lowering (raising) operator t_- (t_+) should be chosen[8] for an incident neutrino (antineutrino). The symbol t_3 denotes the third component of isospin, and an operator suffixed by (n) is understood to act only on the nth nucleon. The weak nuclear charge Q_W for a nucleus with neutron number N and proton number Z is given in terms of the weak mixing angle θ_W by

$$Q_W = N - [1 - 4\sin^2\theta_W]\,Z\,. \tag{11}$$

[8] I take the neutron to be the isospin-up state of the nucleon, i.e., $t_-|n\rangle = |p\rangle$.

2.1.1. Coherent elastic neutrino-nucleus scattering

For NC reactions on a spin-zero ($J_i = 0$) target nucleus, the differential cross section from eq. (5) reduces to a particularly simple form when describing scattering that leaves the nucleus in its ground state. The Gamow-Teller selection rule (eq. (9)) ensures that $B(\mathrm{GT})$ vanishes identically, while the Fermi matrix element for the ground-state-to-ground-state transition becomes

$$B(\mathrm{F}) = \frac{g_V^2\, Q_W^2}{4}\,. \tag{12}$$

For this special case, eq. (5) may be rewritten in terms of the lab-frame kinetic energy T_f of the recoiling ground-state nucleus as

$$\frac{d\sigma_{\mathrm{CEvNS}}}{dT_f} = \frac{G_F^2\, Q_W^2\, g_V^2\, M}{4\pi} R(s)\left[1 - \frac{T_f}{T_f^{\max}}\right] \tag{13}$$

where M is the mass of the nuclear target. The maximum value of T_f allowed by the reaction kinematics is given in terms of the initial lab-frame neutrino energy E_ν by

$$T_f^{\max} = \frac{2E_\nu^2}{2E_\nu + M}\,. \tag{14}$$

This process is known as *coherent elastic neutrino-nucleus scattering* (CEvNS) [39,40]. The nuclear recoil correction factor

$$R(s) \equiv \frac{(s + M^2)^2}{4s^2} \tag{15}$$

is usually neglected[9] in the CEvNS literature (see, e.g., Ref. [110]). Due to its adoption of the allowed approximation, the `MARLEY` treatment of CEvNS currently does not include a q^2-dependent nuclear form factor that accounts for imperfect coherence in the cross section [111].

2.1.2. Coulomb corrections

In eq. (5), the symbol $\mathcal{F}_{\mathrm{CC}}$ is used to include extra factors needed solely for CC scattering. It is defined by

$$\mathcal{F}_{\mathrm{CC}} \equiv \begin{cases} |V_{ud}|^2\, F_C & \mathrm{CC} \\ 1 & \mathrm{NC} \end{cases} \tag{16}$$

where V_{ud} is the Cabibbo–Kobayashi–Maskawa matrix element connecting the up and down quarks. The Coulomb correction factor F_C accounts for the electromagnetic interaction between the outgoing lepton and nucleus in an approximate way. Three prescriptions for computing F_C are available in `MARLEY`: the Fermi function, the effective momentum approximation (EMA), and the modified effective momentum approximation (MEMA).

Fermi function. For very low energies of the outgoing lepton (such as those observed in beta decay), using the *Fermi function* [112,113] as the Coulomb correction factor is a standard approach. A minor complication emerges, however, because the derivation of the Fermi function[10] assumes that the final nucleus is at rest, while the differential cross section in eq. (5) is evaluated in the CM frame and accounts for nuclear recoil. To work around this discrepancy, `MARLEY` uses the relative speed β_{rel} of the two final-state particles [115]

$$\beta_{\mathrm{rel}} = \frac{\sqrt{(k' \cdot p')^2 - m_\ell^2\, m_f^2}}{k' \cdot p'} \qquad \gamma_{\mathrm{rel}} \equiv \left(1 - \beta_{\mathrm{rel}}^2\right)^{-1/2}, \tag{17}$$

to evaluate the Fermi function in the rest frame of the final nucleus using the Lorentz-invariant expression

$$F_{\mathrm{Fermi}} = \frac{2(1+S)}{\left[\Gamma(1+2S)\right]^2} \left(2\,\gamma_{\mathrm{rel}}\,\beta_{\mathrm{rel}}\, m_\ell\, R\right)^{2S-2} e^{-\pi\,\eta}\, \left|\Gamma\left(S - i\eta\right)\right|^2. \tag{18}$$

In eqs. (17) and (18), p', m_f, and Z_f denote the 4-momentum, mass, and proton number of the final nucleus. Likewise, k', m_ℓ, and z_ℓ represent the 4-momentum, mass, and electric charge (in units of the elementary charge) of the final-state lepton. The quantity S is defined in terms of the fine structure constant α by

$$S \equiv \sqrt{1 - \alpha^2 Z_f^2} \tag{19}$$

while

$$R \approx \frac{1.2\, A^{1/3}\,\mathrm{fm}}{\hbar\, c} \tag{20}$$

is the nuclear radius (in natural units), and the Sommerfeld parameter η is given by

$$\eta = \frac{\alpha\, Z_f\, z_\ell}{\beta_{\mathrm{rel}}}. \tag{21}$$

[9] Approximating $R(s) \approx 1$ is accurate to zeroth order in E_ν/M.

[10] See, e.g., Ref. [114].

Effective momentum approximation. For higher energies of the outgoing lepton, the Fermi function is known to overestimate the magnitude of the Coulomb corrections, and an alternative approach called the *effective momentum approximation* (EMA) becomes more appropriate [116]. Let the symbol $\mathcal{K}$ ($\mathcal{E}$) denote the momentum (total energy) of the outgoing lepton in the rest frame of the final nucleus:

$$\mathcal{E} \equiv \gamma_{\text{rel}}\, m_\ell \qquad \mathcal{K} \equiv \beta_{\text{rel}}\, \mathcal{E}\,. \tag{22}$$

Then the *effective* values of these variables

$$\mathcal{K}_{\text{eff}} \equiv \sqrt{\mathcal{E}_{\text{eff}}^2 - m_\ell^2} \qquad \mathcal{E}_{\text{eff}} \equiv \mathcal{E} - V_C(0), \tag{23}$$

are those that exist in the presence of the nuclear Coulomb potential, which is taken to be that at the center of a uniformly-charged sphere:

$$V_C(0) \approx \frac{3\, Z_f\, z_\ell\, \alpha}{2\, R}\,. \tag{24}$$

In the MARLEY implementation of the EMA, the Coulomb correction factor is given by the ratio[11]

$$F_{\text{EMA}} \equiv \frac{\mathcal{K}_{\text{eff}}}{\mathcal{K}}\,. \tag{25}$$

Modified effective momentum approximation. In Ref. [116], an adjustment to the standard EMA prescription is proposed which improves the accuracy of the approximation in cases where the final lepton mass cannot be neglected. Under this *modified effective momentum approximation* (MEMA), the Coulomb correction factor defined in eq. (25) is replaced by

$$F_{\text{MEMA}} = \frac{\mathcal{K}_{\text{eff}}\, \mathcal{E}_{\text{eff}}}{\mathcal{K}\, \mathcal{E}}\,. \tag{26}$$

Default behavior. The final-state Coulomb interaction increases the charged-current cross section for neutrinos and decreases it for antineutrinos. Because the (M)EMA is known to overestimate the size of this effect at low energies while the Fermi function does the same at high energies, previous calculations [64,117] have adopted a simple prescription for combining the two approaches: in any particular case, adopt the method that yields the smallest Coulomb correction. For MARLEY, this amounts to defining the Coulomb correction factor F_C by

$$F_C \equiv \begin{cases} F_{\text{Fermi}} & |F_{\text{Fermi}} - 1| < |F_{\text{MEMA}} - 1| \\ F_{\text{MEMA}} & \text{otherwise} \end{cases} \tag{27}$$

Although eq. (27) represents the default MARLEY approach to Coulomb corrections, this behavior may be altered by the user in the job configuration file (see section 6.7.5).

2.1.3. Total cross section

With the definitions given above, integration of eq. (5) over $\cos\theta_\ell$ becomes trivial, leading to the total cross section

$$\sigma = \frac{G_F^2}{\pi}\, \mathcal{F}_{\text{CC}} \left[\frac{E_i\, E_f}{s}\right] E_\ell\, |\mathbf{p}_\ell| \left[B(\text{F}) + B(\text{GT})\right]. \tag{28}$$

2.2. Nuclear de-excitation model

De-excitations from high-lying nuclear levels are simulated in MARLEY using the Hauser-Feshbach statistical model (HFSM) [109]. This model assumes that the decaying nuclear state may be adequately described as a thermally-equilibrated compound nucleus with a definite excitation energy (E_x), spin (J), and parity (Π). In the MARLEY HFSM implementation, emissions of γ-rays and light nuclear fragments ($1 \leq A \leq 4$) are treated as a sequence of binary decays while fission, production of heavy nucleon clusters ($A \geq 5$), and simultaneous multiparticle evaporation are neglected.

2.2.1. Nuclear fragment emission

According to the HFSM, the distribution of final excitation energies E_x' that may result from the emission of a fragment a (with parity π_a and separation energy S_a) from the compound nucleus is described by the differential decay width

$$\frac{d\Gamma_a}{dE_x'} = \frac{1}{2\,\pi\, \rho_i(E_x, J, \Pi)} \sum_{\ell=0}^{\ell_{\max}} \sum_{j=|\ell-s|}^{\ell+s} \sum_{J'=|J-j|}^{J+j} T_{\ell j}(\varepsilon)\, \rho_f(E_x', J', \Pi') \tag{29}$$

where J' is the final nuclear spin; s, ℓ, and j are the spin, orbital, and total angular momentum quantum numbers of the emitted fragment;

$$\Pi' = (-1)^\ell\, \pi_a\, \Pi \tag{30}$$

[11] The original EMA treatment also involves the use of an effective value of the 4-momentum transfer while computing the scattering amplitude. However, since the nuclear matrix elements in eq. (5) are already evaluated in the limit of zero momentum transfer, MARLEY neglects this additional correction.

is the value of the final-state nuclear parity needed to enforce parity conservation; and ε is the total kinetic energy of the decay products in the rest frame of the initial nucleus. The maximum accessible final-state excitation energy

$$E_x'^{\max} = E_x - S_a \tag{31}$$

is related to the total kinetic energy ε via

$$\varepsilon = E_x'^{\max} - E_x'. \tag{32}$$

The functions ρ_i and ρ_f represent the density of nuclear levels in the vicinity of the initial and final states, while the transmission coefficient $T_{\ell j}$ quantifies how readily the fragment may be emitted from the nucleus. Because the value of $T_{\ell j}$ at fixed ε falls off rapidly with increasing ℓ, MARLEY truncates the infinite sum over orbital angular momenta using an upper limit $\ell_{\max}$. By default, $\ell_{\max} = 5$ is used. This value may be adjusted by the user as described in section 6.7.4.

At low excitation energies, where individual nuclear levels can be resolved, MARLEY treats the level density ρ_f as a sum of delta functions, with one term per level. For a specific nuclear level, the partial decay width for emission of fragment a may be written in the form

$$\Gamma_a = \frac{1}{2\pi\,\rho_i(E_x, J, \Pi)} \sum_{j=|J-J'|}^{J+J'} \sum_{\ell=|j-s|}^{j+s} \delta_\pi^\ell\, T_{\ell j}(\varepsilon) \tag{33}$$

where the symbol δ_π^ℓ, which enforces parity conservation, is equal to one if eq. (30) is satisfied and zero if it is not.

At higher excitation energies, MARLEY computes ρ_f according to the Back-shifted Fermi gas model (BFM) from version 3 of the Reference Input Parameter Library (RIPL-3) [118]. The "BFM effective" values of the level density parameters for this model are adopted from a global fit of nuclear level data for 289 nuclides reported in Ref. [119]. A full description of the BFM as implemented in MARLEY is given in Appendix B of Ref. [107]. The initial level density ρ_i is always evaluated according to the BFM regardless of excitation energy.

The partial decay width for a transition to the continuum of nuclear levels via emission of a fragment a may be computed by integrating eq. (29) over the final excitation energy interval $[E_x'^{\min,c}, E_x'^{\max}]$. The lower bound of the continuum $E_x'^{\min,c}$ is taken by MARLEY to be the excitation energy of the highest tabulated discrete nuclear level for the nuclide of interest. In cases where no such level data are available, the continuum is taken to start at the nuclear ground state ($E_x'^{\min,c} = 0$).

2.2.2. Fragment transmission coefficients

The fragment transmission coefficients $T_{\ell j}$ that appear in eqs. (29) and (33) are computed by numerically solving the radial Schrödinger equation

$$\left[\frac{d^2}{dr^2} + k^2 - \frac{\ell(\ell+1)}{r^2} - \frac{k^2}{\varepsilon}\mathcal{U}(r, \varepsilon_{\text{lab}}, \ell, j)\right] u_{\ell j}(r) = 0 \tag{34}$$

where $u_{\ell j}$ is the fragment's radial wavefunction,

$$\varepsilon_{\text{lab}} = \frac{\varepsilon^2 + 2\,(m_a + M')\,\varepsilon}{2M'} \tag{35}$$

is its kinetic energy in the rest frame of the final nucleus,[12] and

$$k = \sqrt{\frac{(2\,m_a + \varepsilon_{\text{lab}})\,M'^2\,\varepsilon_{\text{lab}}}{(m_a + M')^2 + 2\,M'\,\varepsilon_{\text{lab}}}} \tag{36}$$

is the magnitude of its 3-momentum in the rest frame of the initial nucleus. In eqs. (35) and (36), m_a (M') denotes the mass of the emitted fragment (final nucleus).

The optical potential $\mathcal{U}$ used by MARLEY for nucleon emission is the global parameterization of Koning and Delaroche [120]. For complex nuclear fragments, a folding approach similar to that of Madland [121] is used to construct the optical potential by weighting the individual neutron and proton potentials. More details about the MARLEY nuclear optical potential are available in Appendix C of Ref. [107].

Far from the nucleus, the optical potential approaches the Coulomb potential, and the fragment radial wavefunction approaches the limiting form

$$u_{\ell j}(r) \to \frac{i}{2}\left[H_\ell^-(\eta, kr) - \langle S_{\ell j}\rangle H_\ell^+(\eta, kr)\right] \tag{37}$$

valid for large radii r. Here $H_\ell^\pm$ are the Coulomb wavefunctions [122, ch. 33]. The Sommerfeld parameter

$$\eta \equiv \frac{z\,Z'\,\alpha}{\beta_{\text{rel}}} \tag{38}$$

is evaluated in terms of the proton number z (Z') of the emitted fragment (final nucleus) and the relative speed

[12] The label *lab* for this quantity reflects it status as the laboratory-frame kinetic energy in the time-reversed process wherein the fragment is absorbed to form the compound nucleus. See Appendix A of Ref. [107].

$$\beta_{\text{rel}} = \frac{\sqrt{\varepsilon_{\text{lab}}^2 + 2\, m_a\, \varepsilon_{\text{lab}}}}{m_a + \varepsilon_{\text{lab}}}\,. \tag{39}$$

The energy-averaged S-matrix element $\langle S_{\ell j} \rangle$ that appears in eq. (37) is related to the transmission coefficient $T_{\ell j}$ via

$$T_{\ell j} \equiv 1 - \left| \langle S_{\ell j} \rangle \right|^2 . \tag{40}$$

To approximate $\langle S_{\ell j} \rangle$, MARLEY first obtains a numerical solution $u_{\ell j}(r)$ of eq. (34) using Numerov's method [123–125]. This method computes $u_{\ell j}$ iteratively on a regular grid with fixed radial step size Δ. If one defines the function $a_{\ell j}(r)$ to be equal to the non-derivative terms enclosed in square brackets in eq. (34), i.e.,

$$a_{\ell j}(r) \equiv k^2 - \frac{\ell(\ell+1)}{r^2} - \frac{k^2}{\varepsilon}\, \mathcal{U}(r, \varepsilon_{\text{lab}}, \ell, j), \tag{41}$$

then the Numerov solution (accurate to order Δ^4) $u_n \approx u_{\ell j}(r_n)$ at the nth grid point

$$r_n = n\,\Delta \qquad n \in \{0, 1, 2, \dots\} \tag{42}$$

is given by the recurrence relation

$$u_n = \frac{(2 - 10\, h\, a_{n-1})\, u_{n-1} - (1 + h\, a_{n-2})\, u_{n-2}}{(1 + h\, a_n)} \qquad n \geq 2 \tag{43}$$

and boundary conditions

$$u_0 = 0 \tag{44}$$

$$u_1 = \Delta^{\ell+1}. \tag{45}$$

Here I have defined the abbreviations $h \equiv \Delta^2/12$ and $a_n \equiv a_{\ell j}(r_n)$ for $n \geq 1$, with $a_0 \equiv 0$.

The MARLEY calculation of $\langle S_{\ell j} \rangle$ proceeds through iterations of the Numerov method until the first grid point r_A is encountered such that

$$\left| \mathcal{U}(r_A) - V_C(r_A) \right| \leq V_{\text{thresh}}\,, \tag{46}$$

that is, the difference between the nuclear optical potential $\mathcal{U}$ and the Coulomb potential V_C falls below a small threshold V_{thresh}. Iterations continue further until a second grid point r_B is encountered such that

$$r_B \geq \mathcal{S}\, r_A\,. \tag{47}$$

In MARLEY 1.2.0, the parameter values $\Delta = 0.1$ fm$/\hbar c$, $V_{\text{thresh}} = 1$ keV and $\mathcal{S} = 1.2$ are used.

Comparing the full solution obtained in this way to the asymptotic form from eq. (37) at r_A and r_B leads to the expression

$$\langle S_{\ell j} \rangle \approx \frac{u_{\ell j}(r_A)\, H^-(\eta, k r_B) - u_{\ell j}(r_B)\, H^-(\eta, k r_A)}{u_{\ell j}(r_A)\, H^+(\eta, k r_B) - u_{\ell j}(r_B)\, H^+(\eta, k r_A)}\,. \tag{48}$$

Numerical values of the Coulomb wavefunctions are obtained by interfacing with the GNU Scientific Library [1,2].

2.2.3. Gamma-ray emission

In the Hauser-Feshbach formalism, γ-ray emission is described by the differential decay width

$$\frac{d\Gamma_\gamma}{dE_x'} = \frac{1}{2\pi\, \rho_i(E_x, J, \Pi)} \sum_{\lambda=1}^{\lambda_{\max}} \sum_{J'=|J-\lambda|}^{J+\lambda} \sum_{\Pi' \in \{-1,1\}} T_{X\lambda}(E_\gamma)\, \rho_f(E_x', J', \Pi') \tag{49}$$

where $\lambda \geq 1$ is the multipolarity, $E_\gamma \approx E_x - E_x'$ is the energy of the emitted γ-ray,[13] and

$$X = \begin{cases} \text{E} & \Pi = (-1)^\lambda\, \Pi' \\ \text{M} & \Pi = (-1)^{\lambda+1}\, \Pi' \end{cases} \tag{50}$$

labels the type of transition as either electric (E) or magnetic (M). The infinite sum over multipolarities in eq. (49) is truncated at $\lambda_{\max}$. The default cutoff value $\lambda_{\max} = 5$ may be configured by the user as described in section 6.7.4.

The calculation of the level densities ρ_i and ρ_f is identical to the approach used for nuclear fragment emission (see section 2.2.1). In particular, the level density ρ_f used for γ-ray transitions to discrete levels is once again treated as a sum of delta functions. The partial decay width for γ-ray emission to a particular nuclear level then becomes

[13] Although MARLEY uses the approximate expression for E_γ given here to compute γ-ray decay widths, corrections for nuclear recoil are handled exactly when the actual emission is simulated.

Table 1
Coupling constants used in calculations of neutrino-electron elastic scattering. Values are given in terms of the weak mixing angle θ_W.

Neutrino	g_1	g_2
ν_e	$1/2 + \sin^2\theta_W$	$\sin^2\theta_W$
$\bar{\nu}_e$	$\sin^2\theta_W$	$1/2 + \sin^2\theta_W$
ν_μ, ν_τ	$-1/2 + \sin^2\theta_W$	$\sin^2\theta_W$
$\bar{\nu}_\mu, \bar{\nu}_\tau$	$\sin^2\theta_W$	$-1/2 + \sin^2\theta_W$

$$\Gamma_\gamma = \frac{1}{2\pi\,\rho_i(E_x, J, \Pi)} \sum_{\lambda=\max(1,|J-J'|)}^{J+J'} T_{X\lambda}(E_\gamma)\,. \tag{51}$$

If $J + J' < 1$, the width Γ_γ vanishes.

Similarly to the fragment emission case, calculation of the partial decay width for γ-ray transitions to the continuum of nuclear levels is performed by integrating eq. (49) over the interval $[E_x'^{\min,c}, E_x'^{\max}]$. The continuum lower bound $E_x'^{\min,c}$ is handled as in section 2.2.1, while the upper bound becomes

$$E_x'^{\max} = E_x\,. \tag{52}$$

2.2.4. Gamma-ray transmission coefficients

The transmission coefficients $T_{X\lambda}$ that appear in eqs. (49) and (51) are typically expressed in terms of a *strength function* $f_{X\lambda}(E_\gamma)$ such that

$$T_{X\lambda}(E_\gamma) = 2\pi\,E_\gamma^{2\lambda+1}\,f_{X\lambda}(E_\gamma)\,. \tag{53}$$

In `MARLEY` 1.2.0, the expression used to evaluate the strength function

$$f_{X\lambda}(E_\gamma) = \frac{\sigma_{X\lambda}}{(2\lambda+1)\,\pi^2}\left[\frac{\Gamma_{X\lambda}^2\,E_\gamma^{3-2\lambda}}{(E_\gamma^2 - E_{X\lambda}^2)^2 + E_\gamma^2\,\Gamma_{X\lambda}^2}\right] \tag{54}$$

is taken from the RIPL-3 Standard Lorentzian model [118]. According to this model, γ-ray emissions of type $X\lambda$ are assumed to take place via de-excitation of the corresponding giant multipole resonance, which has centroid excitation energy $E_{X\lambda}$, width $\Gamma_{X\lambda}$, and peak cross section $\sigma_{X\lambda}$. A full listing of the values of these parameters is given in Table II of Ref. [107].

2.3. Neutrino-electron elastic scattering

In addition to neutrino-nucleus scattering, `MARLEY` is also capable of simulating neutrino-electron elastic scattering. For a target atom with proton number Z, the differential cross section in the CM frame for this process is computed according to

$$\frac{d\sigma}{d\cos\theta_\nu} = \frac{2ZG_F^2E_\nu^2}{\pi}\left[g_1^2 + \frac{g_1\,g_2\,m_e^2}{s}\,(\cos\theta_\nu - 1) + g_2^2\left(1 + \frac{1}{2}\left[1 - \frac{m_e^2}{s}\right]\left[\cos\theta_\nu - 1\right]\right)^2\right] \tag{55}$$

where m_e is the electron mass and E_ν (θ_ν) is the energy (scattering angle) of the neutrino. The coupling constants g_1 and g_2 depend on the neutrino species and are given in Table 1. Electron binding energies are neglected.

3. Random sampling implementation

Like any other Monte Carlo event generator, `MARLEY` must make extensive use of pseudorandom numbers and produce samples from a variety of discrete and continuous probability distributions. With the advent of C++11, a suite of high-quality random number generation tools was adopted as part of the C++ standard library [126], and `MARLEY` relies heavily on these new features. All random numbers used by `MARLEY` are obtained using the C++ standard library object `std::mt19937_64`, which provides a 64-bit implementation of the Mersenne Twister algorithm developed by M. Matsumoto and T. Nishimura [127,128].

3.1. Discrete distributions

All sampling from discrete distributions is handled using instances of the C++ standard library object `std::discrete_distribution`. In cases where the sampling weights for each possible outcome already exist in memory, the usual method for initializing this object is to supply iterators that point to the beginning and end of a collection of sampling weights. For example, line 5 of the code snippet[14]

[14] The examples given in this section make use of a C++17 feature (class template argument deduction) in order to avoid code clutter that is unimportant for a conceptual understanding. In the `MARLEY` 1.2.0 source code, the `std::discrete_distribution` and `marley::IteratorToMember` class templates are used in a manner that is compatible with C++14.

```
1  #include <random>
2  #include <vector>
3
4  std::vector<double> weights = { 1., 2. };
5  std::discrete_distribution dist1( weights.begin(), weights.end() );
```

initializes a `std::discrete_distribution` object `dist1` which will sample `int` values of 0 (1) with probability 1/3 (2/3).

When the sampling weights are stored as object data members, however, initializing the distribution becomes more complicated. In particular, a naïve attempt using the approach from the example above

```
6  struct A {
7    A(double w) : weight(w) {}
8    double weight;
9  };
10
11 std::vector<A> As = { A(1.), A(2.) };
12 std::discrete_distribution dist2( As.begin(), As.end() );
```

triggers a compilation error on line 12. Because the iterators returned by `As.begin()` and `As.end()` refer to objects of type `A` instead of the weights (of type `double`), they cannot be used to initialize `dist2`.

`MARLEY` works around this difficulty by implementing the `iterator_to_member` interface proposed by T. Becker [129]. This interface converts an iterator that points to an object (`A`) into an iterator that points to one of that object's data members (`weight`). In this example, including the appropriate `MARLEY` header file (`include/marley/IteratorToMember.hh`) and replacing line 12 with

```
12 marley::IteratorToMember w_begin( As.begin(), &A::weight );
13 marley::IteratorToMember w_end( As.end(), &A::weight );
14 std::discrete_distribution dist2( w_begin, w_end );
```

compiles successfully and yields the same sampling behavior for `dist2` as for `dist1`. For cases in which the input iterators refer to object pointers instead of the objects themsleves, `MARLEY` provides a similar interface via the `marley::IteratorToPointerMember` class template, which is compatible with both bare (e.g., `A*`) and smart pointers (e.g., `std::unique_ptr<A>`).

This approach is used in several places in the `MARLEY` source code to initialize discrete distributions using sampling weights stored as object data members, e.g., relative intensities owned by `marley::Gamma` objects representing distinct γ-ray de-excitations from a particular nuclear energy level.

3.2. Continuous 1D distributions: accept/reject approach

To sample from continuous one-dimensional distributions, `MARLEY` implements two general schemes, both of which take the bounds of a sampling interval $[a, b]$ and an arbitrary probability density function $f(x)$ (for which no particular normalization is assumed) as input. The first scheme uses a simple rejection method which relies on an accurate knowledge of the global maximum $f_{\max}$ of $f(x)$ within the sampling interval. If $f_{\max}$ is known in advance, it may be supplied along with the other input parameters. Otherwise, it is estimated within a specified tolerance by minimizing $-f(x)$ using Brent's method [130]. Once the value of $f_{\max}$ has been obtained, pairs of uniformly-distributed variables $x \in [a, b]$ and $y \in [0, f_{\max}]$ are repeatedly sampled. This continues until $y \leq f(x)$, at which point the sampled x value is accepted.

3.3. Continuous 1D distributions: inverse transform approach

In cases where the global maximum $f_{\max}$ is unknown and its estimation via Brent's method is either unreliable (because a local maximum may be found instead of the global one) or inefficient (because f is computationally expensive to evaluate), `MARLEY` employs a second sampling scheme based on the "fast inverse transform sampling" algorithm originally proposed in ref. [108]. Although `MARLEY` uses numerical techniques similar to those in the original MATLAB code [131], which was written as an extension of the Chebfun package [132,133], the C++ implementation described herein is original. To the author's knowledge, `MARLEY` represents the first application of this algorithm to physics event generation.

3.3.1. Algorithm

To obtain a random sample x from $f(x)$ using inverse transform sampling, `MARLEY` first constructs a polynomial approximant $\tilde{f}$ of f using a grid of $N+1$ ordered pairs (x_j, f_j) for $j \in \{0, 1, \ldots, N\}$, where the x_j are the Chebyshev points of the second kind

$$x_j = \frac{a + b + (b - a)\cos(\pi j/N)}{2} \tag{56}$$

and the f_j are the function values

$$f_j \equiv f(x_j). \tag{57}$$

At points other than the x_j (where f_j is used directly), the polynomial approximant $\tilde{f}$ is given by the barycentric formula [134]

$$\tilde{f}(x) \equiv \left[\sum_{j=0}^{N} \frac{(-1)^j w_j^N}{x - x_j} f_j\right] \Bigg/ \left[\sum_{j=0}^{N} \frac{(-1)^j w_j^N}{x - x_j}\right] \tag{58}$$

with weights

$$w_j^N \equiv \begin{cases} 1/2 & j = 0 \text{ or } j = N \\ 1 & \text{otherwise.} \end{cases} \tag{59}$$

The f_j are related to the coefficients α_k that appear in an Nth order expansion of f in Chebyshev polynomials T_k, i.e.,

$$f(x) \approx \sum_{k=0}^{N} w_k^N \, \alpha_k \, T_k(u(x)) \qquad u(x) \equiv 2\left(\frac{x-a}{b-a}\right) - 1, \tag{60}$$

by the type-I discrete cosine transform (DCT-I)

$$\alpha_k = \frac{f_0 + (-1)^k f_N}{N} + \frac{2}{N} \sum_{j=1}^{N-1} f_j \cos\left(\frac{\pi jk}{N}\right). \tag{61}$$

To approximate the cumulative density function (CDF)

$$F(z) \equiv \int_a^z f(x)\, dx, \quad z \in [a, b] \tag{62}$$

MARLEY uses the formulas

$$\int_{-1}^{y} T_k(u)\, du = \begin{cases} T_1(y) + 1 & k = 0 \\ \frac{1}{4}\left[T_2(y) - 1\right] & k = 1 \\ \frac{1}{2}\left[\frac{T_{k+1}(y)}{k+1} - \frac{T_{k-1}(y)}{k-1}\right] + \frac{(-1)^{k+1}}{k^2-1} & k \geq 2 \end{cases} \tag{63}$$

to integrate eq. (60) term-by-term, yielding an $(N+1)$th order Chebyshev expansion

$$F(z) \approx \sum_{k=0}^{N+1} w_k^{N+1} \, \beta_k \, T_k(u(z)) \tag{64}$$

where the coefficients β_k are given by

$$\beta_k = \left(\frac{b-a}{2}\right) B_k \tag{65}$$

with

$$B_k \equiv \begin{cases} \alpha_0 - \frac{1}{2}\alpha_1 + 2\sum_{j=2}^{N} \frac{\alpha_j (-1)^j}{1-j^2} & k = 0 \\ \frac{1}{2k}\left(\alpha_{k-1} - \alpha_{k+1}\right) & k > 0 \text{ and } k < N \\ \frac{1}{2k}\alpha_{k-1} & k = N \text{ or } k = N+1. \end{cases} \tag{66}$$

One may obtain a polynomial approximant $\tilde{F}$ of the cumulative density function F by applying a second DCT-I (which is its own inverse) to the expansion coefficients β_k:

$$F_\ell = \frac{\beta_0 + (-1)^\ell \beta_{N+1}}{2} + \sum_{k=1}^{N} \beta_k \cos\left(\frac{\pi k \ell}{N+1}\right). \tag{67}$$

Using the $N+1$ Chebyshev points

$$x_\ell = \frac{a + b + (b-a)\cos(\pi\,\ell/[N+1])}{2}, \tag{68}$$

one may approximate $F(x)$ for any $x \in [a, b]$ using eq. (58) with the substitutions $f \to F$, $j \to \ell$, and $N \to N+1$.

With the approximate CDF $\tilde{F}(x)$ constructed in this manner, MARLEY obtains a random sample x from $f(x)$ by generating a uniform random number $\xi \in [0, 1]$. Bisection is then used to find the sampled value of $x \in [a, b]$ which satisfies the relation

$$\xi = \tilde{F}(x) \big/ \tilde{F}(b)\,. \tag{69}$$

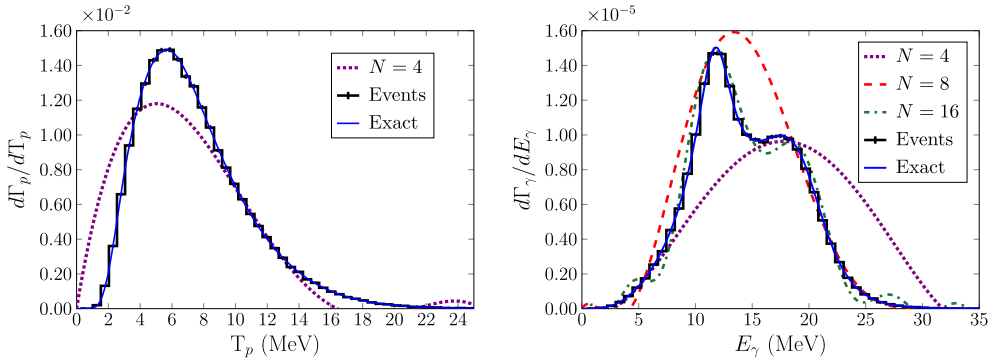

Fig. 1. Validation of MARLEY's inverse transform sampling algorithm using simulations of proton (left) and γ-ray (right) emission to the continuum from a highly-excited ^{40}K state. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

3.3.2. Validation

Fig. 1 shows an application of this sampling technique to the modeling of compound nuclear decays in MARLEY. In the left-hand plot, the black histogram shows the spectrum of proton kinetic energies T_p obtained from a simulation of 200,000 decays $^{40}\mathrm{K} \rightarrow p + {}^{39}\mathrm{Ar}$ to the excitation energy continuum of the daughter ^{39}Ar nucleus. The initial ^{40}K state had excitation energy $E_x = 45$ MeV and spin-parity $J^\pi = 4^+$. Decay modes other than proton emission to the continuum were switched off for simplicity, and the kinetic energy T_p is reported in the rest frame of the mother ^{40}K nucleus. The contents of each histogram bin are normalized to provide a calculation of the differential decay width $d\Gamma_p/d\mathrm{T}_p$, where the average value of this quantity in the wth bin is approximated by the Monte Carlo estimator

$$\left\langle \frac{d\Gamma_p}{d\mathrm{T}_p} \right\rangle_w \approx \frac{n_w \, \Gamma_p}{\mathrm{N_{events}} \, \Delta\mathrm{T}_p^w} \, . \tag{70}$$

Here $\mathrm{N_{events}} = 200{,}000$ is the total number of simulated decay events, n_w is the number of these that fall within the wth bin, $\Delta\mathrm{T}_p^w$ is the width of the wth bin, and Γ_p is the total width for proton emission to the continuum.

The blue curve shows a direct calculation of the differential decay width via

$$\frac{d\Gamma_p}{d\mathrm{T}_p} = \frac{M}{M'} \frac{d\Gamma_p}{dE_x'} \tag{71}$$

where M (M') is the mass of the initial (final) nucleus and $d\Gamma_p/dE_x'$ is evaluated according to eq. (29) with the fragment species $a = p$. The agreement seen between the exact calculation and the simulated events is achieved by sampling the latter from a CDF constructed using a Chebyshev polynomial approximant to $d\Gamma_p/dE_x'$ with grid size $N = 64$. The polynomial approximant with $N = 64$ and the exact calculation are indistinguishable on the scale of the plot. For reference, a lower-order ($N = 4$) Chebyshev polynomial approximation to the distribution is also shown by the dotted purple line.

The right-hand plot in Fig. 1 shows simulation results obtained using an identical procedure, except that the decay process $^{40}\mathrm{K} \rightarrow \gamma + {}^{40}\mathrm{K}$ is considered using the differential width from eq. (49). Three lower-order Chebyshev approximants are shown which provide improved agreement with the exact calculation as the grid size N grows.

3.3.3. Implementation details

Although the barycentric interpolation scheme described here is used in MARLEY solely for inverse transform sampling, the C++ implementation is very general and may find useful applications elsewhere. The `marley::ChebyshevInterpolatingFunction` class constructs the polynomial interpolant $\tilde{f}$ for an arbitrary input function $f(x)$, represented by a `std::function<double(double)>`. If the grid size N is not specified in the constructor, then an adaptive technique is used to choose a grid size sufficiently large to represent $f(x)$ at close to machine precision. Starting with $N = 2$, the value of N is doubled (and the Chebyshev expansion coefficients are recomputed) until the stopping criterion

$$\alpha_m < 2\,\varepsilon \max(\alpha_0, \alpha_1, \ldots, \alpha_N) \qquad m = N, N-1 \tag{72}$$

(with machine epsilon ε) is satisfied or N reaches a large maximum value. The `evaluate(double x)` member function returns the value of $\tilde{f}(x)$, and the `cdf()` member function returns a new `marley::ChebyshevInterpolatingFunction` representing $\tilde{F}(x)$. The DCT-I calculations in eqs. (61) and (67) are carried out using the fast Fourier transform C library `FFTPACK4` [135], which is included in the MARLEY source code distribution. This library is based on the original `FFTPACK` Fortran code developed by P. Swarztrauber [136,137].

No simultaneous sampling of multiple variables from a joint probability distribution is needed to implement the physics models in the current version of MARLEY.

4. Event generation workflow

The flowchart in Fig. 2 illustrates the procedure used by MARLEY to generate events. In the following paragraphs, each stage in the process will be described. Unless otherwise noted by providing an explicit namespace specifier (e.g., `std::`), all C++ classes referred to using `typewriter font` in this section are defined within the `marley` namespace.

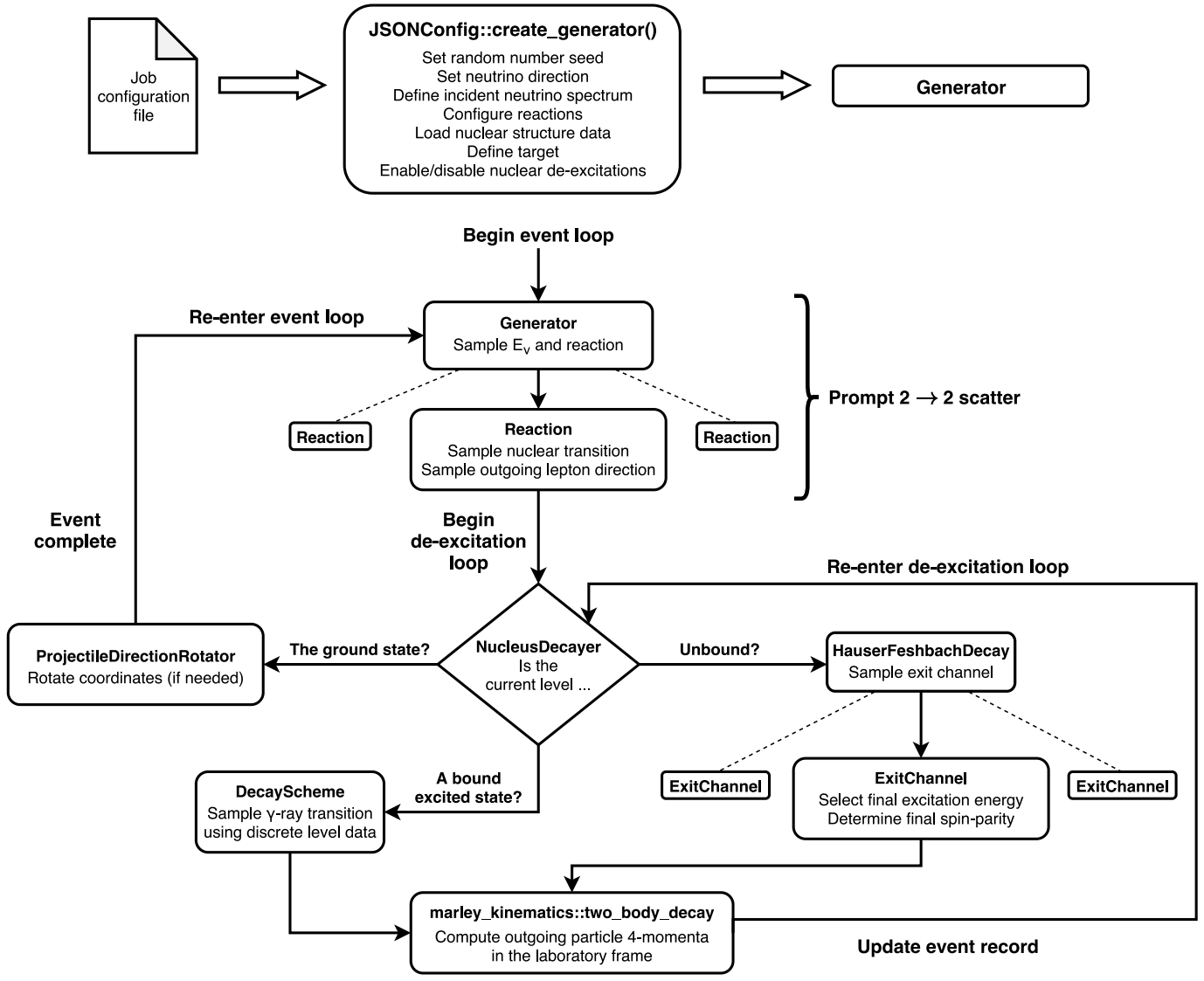

Fig. 2. Illustration of the workflow used by `MARLEY` to generate neutrino-nucleus scattering events.

4.1. Generator initialization

At the beginning of an event generation job, a `JSONConfig` object is used to parse and interpret the settings stored in a configuration file (whose format is described in section 6). The `create_generator` member function is then used to construct a `Generator` object which handles the actual simulation of events. The steps below are followed to initialize the `Generator`.

4.1.1. Set random number seed

If the user has specified an integer value for the random number seed in the configuration file (see section 6.1), then this value is used to initialize a `std::mt19937_64` object owned by the `Generator`. If not, then the system time since the Unix epoch is used as a seed.

4.1.2. Set neutrino direction

By default, `MARLEY` generates events in a reference frame in which the incident neutrino is traveling in the positive z direction. If the user has specified a different neutrino direction (see section 6.2), then this information is passed to a `ProjectileDirectionRotator` object owned by the `Generator`. A rotation matrix is precalculated which will allow the coordinate system to be appropriately transformed at the end of each iteration of the event loop.

4.1.3. Define incident neutrino spectrum

Based on the user's description of the neutrino energy spectrum (see section 6.5), one of several derived classes of the `NeutrinoSource` abstract base class is instantiated and stored as a member of the `Generator` object. In typical `MARLEY` use cases, the `NeutrinoSource` describes the energy distribution of *incident* neutrinos. This behavior is desirable so that the energy spectrum $P(E_\nu)$ of *reacting* neutrinos

$$P(E_\nu) = \frac{\phi(E_\nu)\,\sigma(E_\nu)}{\int_{E_\nu^{\min}}^{E_\nu^{\max}} \phi(E_\nu)\,\sigma(E_\nu)\,dE_\nu} \quad E_\nu^{\min} \le E_\nu \le E_\nu^{\max} \tag{73}$$

where $\phi(E_\nu)$ is the `NeutrinoSource` spectrum and $\sigma(E_\nu)$ is the total cross section, is fully consistent with the `MARLEY` physics models. In unusual situations where cross section weighting is not wanted (e.g., event generation with a uniform distribution of reacting neutrino energies), it may be disabled in the job configuration file as described in section 6.7.3. This corresponds to the substitution $\sigma(E_\nu) \to 1$ in eq. (73).

4.1.4. Configure reactions

Each distinct $2 \to 2$ scattering mode that may be simulated during a `MARLEY` job is represented by an object that implements the abstract base class `Reaction`. This class includes member functions called `total_xs` and `diff_xs`, which respectively compute (in units of MeV^{-2} per target atom) the reaction total cross section $\sigma_r(E_\nu)$ and differential cross section $d\sigma_r(E_\nu)/d\cos\theta_\ell$, where θ_ℓ is the lepton scattering angle in the CM frame. The `create_event` member function simulates the scattering process using these quantities (see sections 4.2.3 and 4.2.4) and returns the results in a newly-created `Event` object.

Based on the set of one or more scattering modes enabled by the user in the configuration file (see section 6.4), a vector of pointers to `Reaction` objects is initialized during job startup and stored as the `reactions_` member of the `Generator`. This vector is populated by calls to the factory method `Reaction::load_from_file`, which processes a reaction specification given in an input file (see section 6.4 and Appendix B). After all reaction input files have been fully processed, the abundance-weighted sum $\sigma(E_\nu)$ of the total cross sections $\sigma_r(E_\nu)$ for all of the enabled reactions is used together with the `NeutrinoSource` spectrum $\phi(E_\nu)$ to construct the probability density function for reacting neutrino energies $P(E_\nu)$ shown in eq. (73). The weights needed to compute $\sigma(E_\nu)$ are the nuclide fractions in the neutrino target, as described in section 4.1.6.

In `MARLEY` 1.2.0, two concrete derived classes of `Reaction` are implemented. The `NuclearReaction` class implements the neutrino-nucleus scattering model described in section 2.1, while the `ElectronReaction` class does the same for the neutrino-electron elastic scattering model from section 2.3.

While simulating a neutrino-nucleus scattering event, `MARLEY` represents the nuclear state in terms of the following quantities: the proton number Z, nucleon number A, excitation energy above the ground state E_x, total spin J, and parity Π. The values of these variables are tracked throughout the event loop, both during $2 \to 2$ scattering (section 4.2) and during simulation of nuclear de-excitations (section 4.3).

A distinction is made in `MARLEY` between bound versus unbound nuclear states. A nuclear state is considered to be unbound if the excitation energy E_x exceeds the separation energy for at least one nuclear fragment with mass number $A \le 4$. Separation energies are computed using atomic and particle mass data from refs. [138,139] tabulated in the file `data/mass_table.js`. The singleton class `MassTable` provides a C++ API for accessing and manipulating these data. If an atomic mass value is not tabulated for a particular nuclide, the `MassTable` class computes an estimate using a formula for the liquid-drop model mass excess developed by Myers and Swiatecki [119,140].

A set of transition matrix elements to the final nuclear states that may be populated by means of a $2 \to 2$ neutrino-nucleus scatter are represented in `MARLEY` by a vector of `MatrixElement` objects. Each of these objects is labeled as either a Fermi or Gamow-Teller matrix element (see section 2.1) using the `type_` member variable, while the `strength_` member stores the corresponding B(F) or B(GT) value. These quantities are given in the reaction input file together with the final nuclear excitation energy, which is stored as the `MatrixElement` member `level_energy_`. A `NuclearReaction` is constructed using a `std::shared_ptr` to a vector of `MatrixElement` objects, allowing shared ownership between multiple reactions as appropriate.

While parsing an input file repesenting a neutrino-nucleus reaction, the static method `Reaction::load_from_file` constructs one `MatrixElement` object for each listed nuclear transition. After the full list of transitions has been processed, a `StructureDatabase` object owned by the `Generator` is consulted to determine whether discrete level data are available for the final-state nucleus of interest. The first such query will trigger loading of these data in the form of `DecayScheme` objects as described in section 4.1.5. If a suitable `DecayScheme` is available, then every `MatrixElement` which represents a transition to a bound nuclear final state is matched to a `Level` object owned by the `DecayScheme`. This matching is performed by selecting the `Level` whose excitation energy most closely matches the value of the `MatrixElement`'s member variable `level_energy_`. Duplicate matches occurring for two `MatrixElement` objects belonging to the same `NuclearReaction` will lead to an exception being thrown. A successful match will result in the `MatrixElement`'s member variable `final_level_` being loaded with a pointer to the matched `Level`. If the spin-parity of the matched `Level` is not compatible with the relevant selection rules (from either eq. (8) or eq. (9)), then a warning message will be printed to the screen.

Matching of this kind is not performed for `MatrixElement` objects representing transitions to unbound nuclear states. In such cases (or when an appropriate `DecayScheme` is not available), a null pointer is assigned to the `final_level_` member.

4.1.5. Load nuclear structure data

To simulate nuclear transitions at low excitation energies, `MARLEY` relies on tabulated nuclear structure data files. These files contain listings of discrete nuclear energy levels and branching ratios for γ-ray transitions between them. The file format is documented in Appendix A.

For `MARLEY` 1.2.0, the recommended structure data files are largely taken (with reformatting) from version 1.6 of the TALYS nuclear reaction code [104,105]. The data are organized by element in the folder `data/structure/`. This folder also includes a text file named `nuclide_index.txt` which serves as an index for the entire dataset: each line of the text file has the format

```
NucPDG  DataFileName
```

where `NucPDG` is an integer PDG code (see section 7.1) representing a nuclear species and `DataFileName` is the name of the corresponding file (assumed to appear in `data/structure/`) in which its discrete level data are given.

A `StructureDatabase` object owned by the `Generator` provides an interface for accessing and manipulating nuclear structure data within `MARLEY`. The `StructureDatabase` manages a lookup table indexed by nuclear PDG code that stores pointers to `DecayScheme` objects. Each `DecayScheme` represents a table of nuclear levels and γ-ray branching ratios for a particular nuclide. A fully-initialized `DecayScheme` owns one `Level` object for every discrete level listed for the nuclide of interest in the associated structure data file. Each `Level` itself owns one `Gamma` object for each listed γ-ray transition that may originate from it.

External access to the `DecayScheme` objects is provided by the `StructureDatabase` member function `get_decay_scheme`. This function takes a nuclear PDG code or a (Z, A) pair as input. If a corresponding `DecayScheme` already exists in the lookup table, a pointer to it is immediately returned. If a suitable `DecayScheme` has not been constructed yet, the `StructureDatabase` consults the nuclide index (which is automatically loaded from `nuclide_index.txt` when needed and cached for repeated use) in an attempt to find a matching structure data file. If a match is found, `DecayScheme` objects are constructed and added to the lookup table for every nuclide listed in the file. A pointer to the requested `DecayScheme` is then returned if it was successfully loaded. A null pointer is stored in the lookup table and returned when the attempt to load the requested `DecayScheme` fails.

Additional information indexed by nuclear PDG code is also stored in the `StructureDatabase`, including (1) ground-state nuclear spin-parities[15] loaded from the file `data/structure/gs_spin_parity_table.txt` and (2) three varieties of objects (represented by the abstract base classes `LevelDensityModel`, `OpticalModel`, and `GammaStrengthFunctionModel`) used to compute quantities of interest for the Hauser-Feshbach statistical model (see sections 2.2 and 4.3.1). In `MARLEY` 1.2.0, these three abstract base classes each have a single concrete implementation which is used for calculations.[16] The `BackshiftedFermiGasModel` class is derived from `LevelDensityModel` and computes nuclear level densities (e.g., ρ_i and ρ_f in eq. (29)) as described in section 2.2.1. The `Koning DelarocheOpticalModel` class is derived from `OpticalModel` and computes nuclear fragment transmission coefficients using the procedure outlined in section 2.2.2. Finally, the `StandardLorentzianModel` class is derived from `GammaStrengthFunctionModel` and computes the γ-ray strength function from eq. (54) as described in section 2.2.4.

Beyond the nuclide-specific items managed by the `StructureDatabase`, there are also two settings which are common to all nuclei: the cutoff values $\ell_{\max}$ and $\lambda_{\max}$ used to truncate the sums in eq. (29) and eq. (49), respectively, (see sections 2.2.1, 2.2.3 and 6.7.4) and a list (elements of which are represented by the `Fragment` class) of the nuclear fragments to consider when simulating decays using the Hauser-Feshbach model.

4.1.6. Define target

In a `MARLEY` simulation, the isotopic composition of the material illuminated by the incident neutrinos is represented by a `Target` object owned by the `Generator`. In the absence of an explicit list of abundances specified by the user (see section 6.3), `MARLEY` assumes equal amounts of each nuclide that appears in the initial state of at least one configured `Reaction`. Using the information stored in the `Target` object, the `Generator` computes the abundance-weighted total cross section per target atom via

$$\sigma(E_\nu) = \sum_r f_r \sigma_r(E_\nu) \tag{74}$$

where f_r is the nuclide fraction for the initial-state atom involved in the rth `Reaction`.

4.1.7. Enable/disable nuclear de-excitations

By default, `MARLEY` simulates both the prompt $2 \to 2$ scattering reaction and the subsequent nuclear de-excitations for every event. In applications where only the prompt reaction is important, the user may disable de-excitations as described in section 6.7.2. The choice of whether or not to simulate de-excitations is represented as a boolean member variable owned by the `Generator` object.

4.2. Event loop: $2 \to 2$ scattering

After the `Generator` object has been fully initialized, the `marley` command-line executable enters an event loop. This loop iterates until a fixed number of events requested by the user (see section 6.6) has been reached or the loop is interrupted (by an error condition or by the user pressing `ctrl+C`). A single event loop iteration corresponds to a call to the `Generator` member function `create_event`. This function returns an `Event` object constructed according to the following steps.

4.2.1. Neutrino energy selection

In the first step of the event loop, the `Generator` employs rejection sampling (see section 3.2) to select a reacting neutrino energy from the probability distribution $P(E_\nu)$ defined in eq. (73). By default, Brent's method is used during the first event loop iteration to obtain a numerical estimate $P_{\max}$ of the maximum value of the probability density function $P(E_\nu)$. The value of $P_{\max}$ is cached and reused during neutrino energy sampling for subsequent events.

In cases where Brent's method fails to converge to the global maximum of $P(E_\nu)$, the resulting distribution of E_ν in the generated events will be biased, with too few events produced in energy regions where $P(E_\nu)$ is larger than $P_{\max}$. At the start of each pass through the event loop, `MARLEY` verifies that all values of $P(E_\nu)$ computed during rejection sampling never exceed $P_{\max}$. If the cached value of $P_{\max}$ is ever found to be an underestimate of the true global maximum, an error message is printed (see Listing 2 in section 5.4.1 for an example), and the value of $P_{\max}$ is updated to match the largest value of $P(E_\nu)$ encountered.

Users can override the default use of Brent's method by supplying their own values of $P_{\max}$ in the job configuration file. Instructions for doing so can be found in section 6.7.7. For convenience in troubleshooting, the error message printed by `MARLEY` in response to a rejection sampling problem contains a recommended value of $P_{\max}$ to use in this way.

[15] Also obtained from nuclear structure data files included with TALYS 1.6.

[16] A second `GammaStrengthFunctionModel` implementation, `WeisskopfSingleParticleModel`, is preserved in the code base for historical interest but has been deprecated.

4.2.2. Reaction mode sampling

Once a reacting neutrino energy E_ν has been chosen, the `Generator` samples a specific reaction mode r from the discrete probability distribution

$$P(r) = \frac{f_r \sigma_r(E_\nu)}{\sigma(E_\nu)}. \tag{75}$$

Control then passes to the `create_event` member function of the rth `Reaction` object owned by the `Generator`. This function handles selection of $2 \to 2$ scattering kinematics and initializes an `Event` object.

4.2.3. Selection of a nuclear transition

If the sampled reaction mode involves scattering on a nuclear target, then simulation of the prompt $2 \to 2$ reaction is handled by the `NuclearReaction` class, and the outgoing nucleus may be left in one of potentially many final states. A specific final state is chosen for a nuclear reaction by selecting a `MatrixElement` according to the discrete distribution

$$P(\Lambda) = \frac{\sigma_r(E_\nu, \Lambda)}{\sigma_r(E_\nu)} \tag{76}$$

where $\sigma_r(E_\nu, \Lambda)$ is the partial reaction cross section computed for the Λth `MatrixElement` owned by the `NuclearReaction` of interest. That is, $\sigma_r(E_\nu, \Lambda)$ is given by the expression in eq. (28) with the quantity $B(\mathrm{F}) + B(\mathrm{GT})$ set equal to the `strength_` member of the Λth `MatrixElement`. The total cross section $\sigma_r(E_\nu)$ is obtained by summing over all of the owned `MatrixElement` objects which represent energetically-accessible nuclear transitions:

$$\sigma_r(E_\nu) = \sum_\Lambda \sigma_r(E_\nu, \Lambda) \tag{77}$$

If the sampled `MatrixElement` has been matched to a known discrete nuclear `Level` (i.e., if its member pointer `final_level_` is non-null, see section 4.1.4), then the excitation energy, spin, and parity of the final nucleus are determined directly from the `Level` object.

If a match is not available (e.g., because the associated nuclear state is unbound), then the final nuclear excitation energy E_x is set equal to the sampled `MatrixElement`'s member variable `level_energy_`. The final-state spin J and parity Π are determined according to the selection rules given in either eq. (8) or eq. (9) as appropriate for the `MatrixElement` type. In the case of a Gamow-Teller transition involving an initial nucleus with nonzero spin (as determined using the table of ground-state spin-parities managed by the `Structure Database`, see section 4.1.5), multiple J values will satisfy the spin selection rule in eq. (9). In such cases, `MARLEY` 1.2.0 makes a rough approximation: equipartition of spin is assumed, and a definite value of J is sampled from the discrete distribution

$$P(J) = \frac{\rho(E_x, J, \Pi)}{\sum_K \rho(E_x, K, \Pi)} \tag{78}$$

where ρ (which is calculated using the same level density treatment as in section 2.2.1) is the density of final-state nuclear levels with spin J and parity Π in the vicinity of excitation energy E_x. In eq. (78), the sum in the denominator runs over all final spin values K that satisfy the Gamow-Teller spin selection rule from eq. (9).

4.2.4. Outgoing lepton direction

Due to the simple kinematics of $2 \to 2$ scattering, the 4-momenta of the outgoing particles are fully determined by specifying their masses (including the final nuclear excitation energy) and the direction of the outgoing lepton. This direction is represented in `MARLEY` using the lepton's azimuthal scattering angle ϕ_ℓ and polar scattering cosine $\cos\theta_\ell$, both measured with respect to the incident neutrino direction in the center-of-momentum (CM) frame.

For all reactions currently implemented in `MARLEY` 1.2.0, the differential cross section is independent of ϕ_ℓ, and the value of this variable is therefore sampled uniformly on the half-open interval $[0, 2\pi)$.

For neutrino-nucleus reactions, a value of $\cos\theta_\ell$ is obtained via rejection sampling (see section 3.2) on the interval $[-1, 1]$ from the probability density function

$$P(\cos\theta_\ell) = \frac{1}{\sigma_r(E_\nu, \Lambda)} \frac{d\sigma_r(E_\nu, \Lambda)}{d\cos\theta_\ell} \tag{79}$$

where $d\sigma_r(E_\nu, \Lambda)/d\cos\theta_\ell$ is the CM-frame differential cross section for the reaction r proceeding via a nuclear transition described by the Λth `MatrixElement`. This differential cross section is computed as in eq. (5), except that only one of the two matrix element terms is used. For a Fermi transition, $B(\mathrm{F})$ is set equal to the `strength_` member variable of the `MatrixElement`, while $B(\mathrm{GT})$ is set to zero. For Gamow-Teller transitions, the reverse is done. In the case of neutrino-electron elastic scattering, rejection sampling is still performed using the distribution from eq. (79), but the differential cross section is calculated as in eq. (55).

For nuclear transitions proceeding purely via a Fermi or Gamow-Teller operator, the angular dependence in the CM frame reduces to (see eq. (5))

$$P(\cos\theta_\ell) \propto \begin{cases} 1 + \beta_\ell \cos\theta_\ell & \text{Fermi} \\ 1 - \frac{1}{3}\beta_\ell \cos\theta_\ell & \text{Gamow-Teller} \end{cases} \tag{80}$$

where $\beta_\ell \in [0, 1]$ is the speed of the outgoing lepton. The linear expressions seen here for either case allow the maximum of the distribution (needed for rejection sampling) to be found analytically. Nuclear transitions that do not correspond to one of these two cases are

neglected in the current version of MARLEY. The maximum of the $\cos\theta_\ell$ distribution for neutrino-electron elastic scattering is also found analytically.

After a CM frame direction is sampled for the outgoing lepton, the active Reaction initializes Particle objects with the initial and final 4-momenta for the $2 \to 2$ scattering event. The values of these in the laboratory frame are stored in a newly-constructed Event object together with the excitation energy and spin-parity of the final energy level of the target.

At this point, program control passes back to the Generator, which applies two additional operations to the Event object before the event loop is complete. Once an Event object has been created by a Reaction, further operations on it are handled by derived instances of the EventProcessor abstract base class. These define a member function called process_event which takes references to the Event and to the Generator as arguments.

4.3. Event loop: nuclear de-excitations

For reactions involving a transition to an excited state of a nuclear target, the subsequent de-excitations are simulated using an EventProcessor called NucleusDecayer. This class manages a de-excitation loop that is initialized using the proton number (Z), nucleon number (A), excitation energy (E_x), spin (J), and parity (Π) of the outgoing nuclear state stored in the Event object. The values of these variables are updated during each iteration of the de-excitation loop, which simulates a sequence of binary decays until the nuclear ground state ($E_x = 0$) is reached.

Each binary decay step begins with a check to determine whether the current nuclear state is bound or unbound (see section 4.1.4).

4.3.1. Unbound nuclear states

In the case of an unbound nuclear state, the NucleusDecayer constructs a HauserFeshbachDecay object, which provides a Monte Carlo implementation of the Hauser-Feshbach statistical model (see section 2.2) for decays of a compound nucleus. Upon construction, the HauserFeshbachDecay object uses the properties of the initial nuclear state (Z, A, E_x, J, Π) and information from the Generator's owned StructureDatabase to initialize a member vector of pointers to ExitChannel objects.

The abstract base class ExitChannel represents a nuclear de-excitation via emission of a γ-ray or a particular nuclear fragment. Two pairs of abstract derived classes virtually inherit from ExitChannel. The classes in the first pair, DiscreteExitChannel and ContinuumExitChannel, specialize to handling transitions to a specific discrete nuclear level and to the continuum, respectively. The classes in the second pair, FragmentExitChannel and GammaExitChannel, respectively represent de-excitations involving fragment and γ-ray emission. Concrete ExitChannel objects belong to a class that inherits from exactly one member of each pair, with the four possibilities being FragmentDiscreteExitChannel, FragmentContinuumExitChannel, GammaDiscreteExitChannel, and GammaContinuumExitChannel.

Every ExitChannel object owns a member variable called width_, which is initialized during construction with the partial decay width of interest (in MeV^{-1}) via a call to the protected member function compute_total_width. The FragmentDiscreteExitChannel and GammaDiscreteExitChannel classes implement this function using the expressions from eqs. (33) and (51), respectively. The FragmentContinuumExitChannel and GammaContinuumExitChannel classes both share the ContinuumExitChannel::compute_total_width implementation, which integrates the differential decay width over the energetically-accessible continuum (see sections 2.2.1 and 2.2.3) to obtain the width_ value

$$\Gamma_u = \int_{E_x'^{\mathrm{min,c}}}^{E_x'^{\mathrm{max}}} \frac{d\Gamma_u}{dE_x'}\, dE_x' \tag{81}$$

where the emitted particle species $u \in \{p, n, d, t, h, \alpha, \gamma\}$. The numerical integration is performed using Clenshaw-Curtis quadrature [141] by an instance of the Integrator class. Evaluation of the differential decay width $d\Gamma_u/dE_x'$ is delegated to the pure virtual member function differential_width, which is implemented in FragmentContinuumExitChannel and GammaContinuumExitChannel according to the expressions given in eq. (29) and eq. (49), respectively.

When the HauserFeshbachDecay object has been fully initialized, its owned vector of ExitChannel pointers will include an individual object derived from DiscreteExitChannel for each accessible transition to a discrete nuclear level present in the StructureDatabase. It will also contain a pointer to a single object derived from ContinuumExitChannel for each particle species that may be emitted via a transition to the continuum.

To simulate a binary decay step, the NucleusDecayer calls the do_decay method of the new HauserFeshbachDecay object. This causes a particular ExitChannel e to be sampled with probability

$$P(e) = \frac{\Gamma_e}{\sum_\beta \Gamma_\beta}. \tag{82}$$

Here Γ_e is the partial width for the eth ExitChannel (given by the width_ member variable) and the sum in the denominator runs over all of the owned ExitChannel objects. Control then passes to the do_decay member function of the sampled ExitChannel to determine the properties of the final nuclear state. If a DiscreteExitChannel has been sampled, the final nuclear excitation energy (E_x'), spin (J'), and parity (Π') are retrieved from the associated Level object.

For a ContinuumExitChannel, a ChebyshevInterpolatingFunction approximation to the differential decay width $d\Gamma_u/dE_x'$ is used to select a final excitation energy $E_x' \in [E_x'^{\mathrm{min,c}}, E_x'^{\mathrm{max}}]$ via inverse transform sampling (see section 3.3). The differential decay width is then evaluated (via a call to the differential_width member function) for the selected final excitation energy E_x'. Each individual term τ_b in the differential decay width sum (see either eq. (29) or eq. (49) as appropriate for the emitted particle species) is stored in a SpinParityWidth object together with its corresponding J' and Π' values. The final nuclear spin-parity is determined by choosing the bth SpinParityWidth object with probability

$$P(b) = \frac{\tau_b}{\sum_c \tau_c} = \frac{\tau_b}{d\Gamma_u / dE'_x} \tag{83}$$

where the differential decay width in the denominator is evaluated at the sampled value of E'_x.

Once values of the variables defining the final nuclear state (E'_x, J', and Π') have been determined, the procedure required to complete an iteration of the event loop (see section 4.3.3) is similar to that used for decays of bound nuclear levels.

4.3.2. Bound nuclear states

If the nucleus is in a bound excited state, i.e., it has a nonzero excitation energy E_x which is below all of the nuclear fragment emission thresholds, then the `NucleusDecayer` consults the `StructureDatabase` to determine whether a `DecayScheme` has been previously configured for the nuclide of interest (see section 4.1.5). If this is the case, then simulation of the remaining de-excitations is delegated to the `DecayScheme` member function `do_cascade`. During each binary decay step handled by this function, a γ-ray transition γ_j originating from the current `Level` is chosen with probability

$$P(\gamma_j) = \frac{I_j}{\sum_k I_k} \tag{84}$$

where I_j is the relative intensity of the jth `Gamma` owned by the current `Level` and the sum in the denominator runs over all of the owned `Gamma` objects. The properties of the final nuclear state, known immediately from the `Level` pointed to by the sampled `Gamma` object (via its `end_level_` member variable) are used to complete the de-excitation loop iteration as described in section 4.3.3.

If a suitable `DecayScheme` cannot be found in the `StructureDatabase`, then `MARLEY` follows the same procedure as for unbound nuclear levels (see section 4.3.1).

4.3.3. Finishing a de-excitation step

At the end of each iteration of the de-excitation loop, a direction for the emitted γ-ray or nuclear fragment is chosen by sampling a polar cosine $\cos\theta$ and an azimuthal angle ϕ isotropically in the rest frame of the decaying nucleus. This information is used together with the daughter particle PDG codes and masses to initialize two new `Particle` objects via a call to the utility function `marley_kinematics::two_body_decay`. This function handles the elementary kinematical calculations needed to obtain the outgoing particle 4-momenta in the laboratory frame. The `Particle` representing the emitted γ-ray or fragment is appended to the vector of final particles owned by the `Event` object being processed. The `Particle` representing the daughter nucleus, on the other hand, replaces the mother nucleus `Particle` in the `Event`. After these adjustments have been made to the `Event` object and the values of the variables tracked in the de-excitation loop (Z, A, E_x, J, Π) have been updated, simulation of the current binary decay step is complete.

The de-excitation loop will continue to simulate additional binary decay steps until the nuclear ground state is reached, or, in cases where no discrete level data are available for a particular final-state nucleus, until the excitation energy falls below the cutoff $E_x^{\text{cut}} = 1$ keV.

4.4. Event loop: rotation of coordinates

For convenience during internal calculations, all `MARLEY` events are initially generated in a frame in which the incident neutrino direction lies along the positive z axis. If the user has specified a different neutrino direction in the job configuration file (see section 6.2), then an instance of the `ProjectileDirectionRotator` class (which inherits from `EventProcessor`) is used to process the `Event` after the de-excitation loop terminates. All particle 3-momenta in the event are rotated into a new reference frame in which the neutrino is traveling along the direction specified by the user. After this rotation is applied, generation of the new `MARLEY` event is complete.

5. Installation and usage

The current release of `MARLEY` has been tested on both Linux and macOS platforms and is expected to work in any Unix-like environment in which the prerequisites are installed. Building and running `MARLEY` on Windows is not currently supported. The installation instructions presented in this section assume use of the Bash shell [142] and the availability of several standard command-line tools.

5.1. Prerequisites

The following prerequisites are required to build the `MARLEY` source code:

- A C++-14-compliant compiler. Two compilers are officially supported:
  - The GNU Compiler Collection (GCC) [143], version 4.9.4 or greater
  - The Clang frontend for the LLVM compiler infrastructure [144], version 3.5.2 or greater
- GNU Make [145]
- The GNU Scientific Library (GSL) [1,2]

On Linux architectures, all of these prerequisites will likely be available for installation through the standard package manager. On macOS, the use of Homebrew [146] to install GSL is recommended. This may be done by executing the terminal command

```
brew install gsl
```

after Homebrew has been installed on the target system.

Although it is not required in order to build or use `MARLEY`, the popular ROOT data analysis framework [3,4] provides convenient tools for plotting and analyzing simulation results. Users who wish to use the optional interface between the two codes (see sections 7.2.4, 7.3.4 and 7.3.5) should ensure that ROOT is installed before building `MARLEY`.

Other than GSL, `MARLEY`'s only required external dependencies are the C++ standard library and the symbols defined in the `dirent.h` and `sys/stat.h` headers of the C POSIX library [147].

5.2. Obtaining and building the code

The source code for MARLEY 1.2.0 may be downloaded as a compressed archive file from the releases webpage (https://github.com/MARLEY-MC/marley/releases). Both `zip` and `tar.gz` file formats are available. After downloading the source code, the user should unpack the archive file in the desired installation folder. For the `tar.gz` format, this may be done via the command

```
tar xvfz marley-1.2.0.tar.gz
```

After unpacking the source code, the user may build MARLEY by navigating to the `build/` subdirectory and invoking GNU Make:

```
cd marley-1.2.0/build
make
```

At build time, MARLEY verifies that GSL is installed by checking for the presence of the `gsl-config` script on the system path. Similarly, the optional MARLEY interface to ROOT is automatically enabled or disabled based on whether `root-config` script is present. Users who have an existing installation of ROOT but desire to build MARLEY without ROOT support may manually disable the interface by setting the `IGNORE_ROOT` variable while invoking `make`, e.g., via

```
make IGNORE_ROOT=yes
```

If the build is successful, then executing

```
./marley --version
```

from within the `build/` folder should yield the following console output:

```
MARLEY (Model of Argon Reaction Low Energy Yields) 1.2.0
Copyright (C) 2016-2020 Steven Gardiner
License: GNU GPL version 3 <http://opensource.org/licenses/GPL-3.0>
This is free software: you are free to change and redistribute it.
```

Users wishing to contribute improvements to MARLEY may prefer to clone the official source code repository instead of downloading a release archive file. Instructions for doing so are available in the "developer documentation" section of the official website [148].

5.3. Configuring the runtime environment

At runtime, the `marley` command-line executable relies on the system environment variable `MARLEY` to store the path to the root folder of the source code. If generation of events is attempted without this variable being set, then the program will halt after printing the error message

```
[ERROR]: The MARLEY environment variable is not set. Please set it (e.g., by sourcing the setup_marley.sh
    script) and try again.
```

Although the user may manually set the value of the `MARLEY` variable, use of the Bash shell script `setup_marley.sh` to configure the system environment is recommended. In addition to storing the path to the source code, this script makes several other changes to environment variables for user convenience, including adding the `build/` folder to the system search paths for executables[17] and dynamic libraries.[18] The `setup_marley.sh` script appears in the root source code folder and should be executed ("sourced") using the `source` command. From within the `build/` folder, for example, one should source the script via

```
source ../setup_marley.sh
```

Sourcing the setup script does not produce any console output. The instructions given in the remainder of this section assume that the `setup_marley.sh` script has already been run in the current terminal session.

5.4. Running a simulation

The typical procedure for running a MARLEY simulation is to invoke the executable via a command of the form[19]

```
marley CONFIG_FILE
```

where `CONFIG_FILE` is the name (with any needed path specification) of a job configuration file. Section 6 gives a full description of the file format and configuration parameters. Three example job configuration files are included with MARLEY in the `examples/config/` folder:

`annotated.js` A heavily-commented example that provides documentation of the configuration file format similar to the contents of section 6

`COPY_ME.js` An example intended to serve as the basis for new job configuration files written by users

[17] The `PATH` environment variable.

[18] Either the `LD_LIBRARY_PATH` (Linux) or the `DYLD_LIBRARY_PATH` (macOS) environment variable.

[19] The `marley` executable may also be invoked with the command-line option `--marley`. This is best done using a terminal window set to display at least 80 columns and 53 rows in a small font.

```
Event Count = 4001/10000 (40.0% complete, 300.4 events / s)
Elapsed time: 00:00:13 (Estimated total run time: 00:00:33)
Data written to events.ascii 3.80 MB
MARLEY is estimated to terminate on Mon Jun  1 23:54:52 2020 CDT
```

Listing 1: Example status display printed by the `marley` executable.

```
1 [WARNING]: PDF value f(x) = 0.00133991 at x = 48.9882 exceeded the estimated maximum fmax = 0.001 during
      rejection sampling
2 [WARNING]: A new estimate fmax = 0.00135331 will now be adopted.
3 [ERROR]: Estimation of the maximum PDF value failed when using a rejection method to sample reacting
      neutrino energies.
4 This may occur when, e.g., an incident neutrino flux is used that includes multiple sharp peaks.
5 To avoid biasing the energy distribution, please rerun the simulation after adding the following line to
      the MARLEY job configuration file:
6     energy_pdf_max: 0.00135331,
7 If this error message persists after raising energy_pdf_max to a relatively high value, please contact
      the MARLEY developers for troubleshooting help.
```

Listing 2: Example warning and error messages printed in response to a rejection sampling problem encountered when selecting reacting neutrino energies.

`minimal.js` An example of the simplest possible job configuration: default settings are used for all parameters except those that must be explicitly specified

As it executes, the `marley` program will print a variety of logging messages to the screen. Several rows at the bottom of the screen are reserved for a status display that tracks the progress of the simulation. Listing 1 shows the format of the status display. Following the first two rows, which report the current event count and the elapsed time since the simulation began, zero or more rows record the total amount of data written to each output file. The final row displays an estimate of the time at which the simulation job will be completed.

If it becomes necessary to end the simulation before all requested events have been generated, the user may interrupt program execution by pressing ctrl+C. In response, the `marley` executable will terminate gracefully after writing the current event to any open output files. As discussed in section 6.6, two of the available output file formats allow for a simulation job interrupted in this way to be resumed from where it left off.

5.4.1. Troubleshooting rejection sampling problems

To select the energy E_ν of the reacting neutrino in each event, `MARLEY` employs the rejection sampling technique discussed in section 3.2 and the probability density function $P(E_\nu)$ given in section 4.1.3. The validity of the technique depends on obtaining an accurate estimate of the global maximum $P_{\max}$ of $P(E_\nu)$ within the sampling region of interest. Underestimates of $P_{\max}$ will lead to bias in the neutrino energy distribution of the simulated events, while significant overestimates will adversely impact sampling efficiency.

Although numerical estimation of $P_{\max}$ in `MARLEY` is reasonably robust for a variety of realistic neutrino spectra, it is not foolproof: there exist pathological energy distributions (e.g., those with multiple sharp peaks) for which automatic detection of the global maximum often fails. To protect against this problem, `MARLEY` checks that each calculation of $P(E_\nu)$ performed during neutrino energy sampling yields a value that does not exceed $P_{\max}$. If a value larger than $P_{\max}$ is ever encountered, a set of warning and error messages similar to those in Listing 2 will be printed to the screen. A new estimate of $P_{\max}$, given by the problematic value of $P(E_\nu)$ increased by a small "safety factor," will be adopted in subsequent event generation.

While this adaptive approach to improving estimation of $P_{\max}$ may resolve rejection sampling problems in the later part of the simulation, it cannot correct for bias in the energy distribution of the events that have already been generated. A simple strategy for overcoming this difficulty is for the user to restart the simulation from the beginning. By adding a line to the job configuration file containing the `energy_pdf_max` key (see section 6.7.7) and the improved estimate of $P_{\max}$ recommended by the rejection sampling error message (see line 6 of Listing 2), automatic estimation of $P_{\max}$ will be disabled in favor of adopting the user-specified value.

For severe underestimations of $P_{\max}$, several repetitions of this "interrupt and rerun" strategy (each with a larger value of `energy_pdf_max` than the one before) may be needed before correct behavior is achieved. If rejection sampling errors persist across multiple fix attempts, or if the corrected value of $P_{\max}$ results in prohibitively slow performance, a different approach to running the simulation (e.g., splitting a problematic spectrum into energy regions which are handled by separate `MARLEY` jobs) should be pursued.

5.5. The `marley-config` utility

For a variety of applications, e.g., analyzing the standard `MARLEY` output files (section 7.3) and interfacing `MARLEY` with external codes (section 8.1), it may be desirable to make use of `MARLEY` classes within an external C++ program. To facilitate compilation of such programs, a Bash script named `marley-config` is placed in the `build/` folder whenever `MARLEY` is successfully built.[20] This script may be queried to obtain various pieces of information about the build.

Each `marley-config` query is executed by providing the script with one or more command-line options, each beginning with a prefix of two hyphens (`--`). The command-line options may appear in any order and may be repeated. If the `--help` option is present on the command line, then all other arguments are ignored, and a multi-line usage message is printed. All other recognized options produce

[20] The `marley-config` script is loosely patterned after `pkg-config` [149]. Similar scripts are used by ROOT, GENIE, GSL, and Geant4.

Table 2
Directory options recognized by the `marley-config` script.

Option	Directory contents	Relative path
`--bindir`	executables	`build/`
`--datadir`	input data	`data/`
`--incdir`	header files	`include`
`--libdir`	libraries	`build/`
`--srcdir`	source files	`src/`
`--topdir`	top-level `MARLEY` directory	`./`

console output that will be combined into a single line and printed in the order that the corresponding options appeared on the command line.

Three categories of command-line options are recognized by the `marley-config` script. The *compilation options* are used to retrieve compiler flags (compatible with both GCC and Clang) and other helpful information for building external programs that link to the `MARLEY` shared libraries (see section 5.5.1). Brief descriptions of the four compilation options currently recognized by `marley-config` are given below.

`--cflags` Prints the C++ compiler flags needed to build code that includes `MARLEY` header files. If `MARLEY` has been built with ROOT support, then the compiler flags will include `-DUSE_ROOT`, which defines the preprocessor macro `USE_ROOT`. This macro may be used together with the preprocessor directives `#ifdef` or `#ifndef` to test for ROOT-dependent `MARLEY` features in compiled code. An example is given in section 5.5.1.

`--cxx-std` Prints the version of the C++ Standard used by the compiler when `MARLEY` was built. The format used is the same as for parameters passed via the `-std` flag to the compiler. For example, if `MARLEY` was built using the `-std=c++14` flag, then `marley-config` will print `c++14` for this option.

`--libs` Prints the compiler flags needed for linking to the `MARLEY` shared libraries.

`--use-root` Prints the string `yes` if `MARLEY` was built with ROOT support and `no` if it was not.

The *directory options* are used to print the full paths to various subfolders of the `MARLEY` installation. Table 2 lists the available directory options in the first column. The second and third columns list, respectively, a description of the directory's contents and its path relative to the root `MARLEY` folder. For an installation of `MARLEY` in which the top-level directory of the source code is `/home/marley`, executing

```
marley-config --topdir --incdir --bindir
```

will produce the output

```
/home/marley /home/marley/include /home/marley/build
```

A final `marley-config` option category, the *version options*, may be used to identify the installed version of `MARLEY`. Two options in this category are currently recognized by the `marley-config` script: `--git-revision` and `--version`.

The `--git-revision` option is used to print a unique hash value reported by Git at build time. If `MARLEY` was built using a cloned Git repository (see section 5.2), and the `git` executable is present on the system path when GNU Make is invoked, then the hash identifier for the latest commit will be retrieved[21] and saved in the `marley-config` script. If uncommitted changes exist in the Git index or working tree when `MARLEY` is built, then the string `-dirty` will be appended to the hash value. In cases where a hash value is unavailable (e.g., because `MARLEY` was installed without cloning a Git repository), the `marley-config` script will print the string `unknown version` in response to this option.

If a tagged release of `MARLEY` was built,[22] then the `--version` option may be used to print the corresponding version number (e.g., `1.2.0`). Otherwise, the output produced by `marley-config` in response to `--version` is the same as for `--git-revision`.

5.5.1. Compiling programs using `marley-config`

To illustrate usage of the `marley-config` utility, three simple C++ programs that employ `MARLEY` classes are provided in the folder `examples/executables/minimal/`. Each one of these may be compiled using GCC by executing a command of the form

```
g++ -o EXECUTABLE_NAME $(marley-config --cflags --libs) SOURCE_NAME
```

from within `examples/executables/minimal/`. Here `EXECUTABLE_NAME` is the desired file name for the compiled executable and `SOURCE_NAME` is the name of one of the example C++ source files. The `marley-config` script may also be used in a `Makefile` as part of a more complicated build. Interested users should consult the files `examples/executables/build/Makefile` and `examples/marg4/build/Makefile` for concrete examples.

Listing 3 shows the source code for `mass_40Ar.cc`, one of the three example programs included in `examples/executables/minimal/`. The other two, `efr.cc` and `evgen.cc`, are discussed in sections 7.3.1 and 8.1, respectively. On lines 12–13 of Listing 3, a constant reference to the singleton instance of the `marley::MassTable` class is retrieved and used to compute the atomic mass of ^{40}Ar. Lines 15–19 print a message to standard output indicating whether `MARLEY` was built with ROOT support. The `USE_ROOT` preprocessor macro mentioned in section 5.5 is used to determine whether line 17 should be included in the compiled program. A final message containing the atomic mass of ^{40}Ar is printed to standard output on line 21.

[21] Via the command `git rev-parse --short HEAD`.
[22] This is determined by the presence of a file named `.VERSION` in the top-level `MARLEY` folder at build time.

```
1  // Standard library includes
2  #include <iostream>
3
4  // MARLEY includes
5  #include "marley/MassTable.hh"
6
7  constexpr int Z = 18; // Ar
8  constexpr int A = 40;
9
10 int main() {
11
12   const marley::MassTable& mt = marley::MassTable::Instance();
13   double mass_Ar40 = mt.get_atomic_mass( Z, A );
14
15   std::cout << "MARLEY was";
16   #ifndef USE_ROOT
17     std::cout << " not";
18   #endif
19   std::cout << " built with ROOT support.\n";
20
21   std::cout << "The atomic mass of 40Ar is " << mass_Ar40 << " MeV/c^2\n";
22
23   return 0;
24 }
```

Listing 3: The C++ source code for `examples/executables/minimal/mass_40Ar.cc`, a trivial example program that uses the `marley::MassTable` class.

6. Generator configuration

Before entering the event loop, the `MARLEY` executable reads in various settings (see section 4.1) from a job configuration file. The format of the job configuration file is based on JavaScript Object Notation (JSON) [150], which represents data structures as a nested hierarchy of key-value pairs. A JSON *object* is an unordered set of key-value pairs surrounded by curly braces (`{}`). Each key is an arbitrary string delimited by double quotes (`""`) and separated from its corresponding value by a colon. Each value may be a JSON object itself, a double-quoted string, a number, an array, or one of the words `true`, `false`, or `null` (without surrounding quotes). A JSON *array* is an ordered collection of values surrounded by square brackets (`[]`). Elements of an array need not have the same data type, e.g., the values `true` and `1.05` may be members of the same array. Array elements are separated by commas, as are object key-value pairs. A valid JSON document is itself an object and is thus delimited by a pair of curly braces.

For the convenience of users, the job configuration file format used by `MARLEY` permits three minor deviations from the JSON standard:

- Single-word keys (no whitespace) may be given without surrounding double quotes.
- C++-style comments (`//` and `/* */`) are allowed anywhere in the file.
- A trailing comma is allowed after the final element in objects and arrays.

A filename extension of `.js` is recommended for `MARLEY` job configuration files because the default JavaScript syntax highlighting used by many text editors is well-suited to the JSON-like format.[23]

To parse job configuration files, `MARLEY` relies on a modified version of the SimpleJSON library [151], which was incorporated into the code base as the class `marley::JSON`. An example `MARLEY` job configuration file is shown in Listing 4. Explanations of each parameter needed to fully configure the generator are given in the following subsections.

6.1. Random number seed

The optional `seed` key may be used to specify an integer seed for the random number generator. Any value corresponding to a C++ `long int` (guaranteed by the standard to be at least 32 bits long) may be used as a seed. This includes negative integers, although users should note that these will be reinterpreted as unsigned integers by `MARLEY`. If the `seed` key is omitted from the job configuration file, then the current Unix time will be used as the random number seed.

6.2. Neutrino direction

For simplicity, `MARLEY` initially generates all events in a frame where the incident neutrino is traveling along the positive z direction (see section 4.4). However, it is often useful to simulate events for neutrinos traveling in an arbitrary direction in the lab frame. This functionality is enabled via use of the `direction` key to define a lab-frame direction 3-vector for the incident neutrinos. The Cartesian components of this vector are specified as shown on line 7 of Listing 4. They need not have any particular normalization, but at least one component must be nonzero. If the `direction` key is omitted, then the positive z direction is assumed.

[23] Some email services do not allow the attachment of files with the `.js` extension. To enable sharing of a MARLEY job configuration file via email, it is usually sufficient to change the filename extension to `.txt` before attaching the file.

```
// The full configuration is enclosed in a set of curly braces
{
  // Random number seed
  seed: 123456,

  // Incident neutrino direction
  direction: { x: 0.0, y: 0.0, z: 1.0 },

  // Target specification
  target: {
    nuclides: [ 1000180400 ],
    atom_fractions: [ 1.0 ],
  },

  // Reaction input files
  reactions: [ "ve40ArCC_Bhattacharya2009.react", "ES.react" ],

  // Neutrino source specification
  source: {
    neutrino: "ve", // The source produces electron neutrinos
    type: "monoenergetic",
    energy: 15.0,   // MeV
  },

  // Settings for MARLEY command-line executable
  executable_settings: {
    // The number of events to generate
    events: 10000,

    // Event output configuration
    output: [ { file: "events.ascii", format: "ascii", mode: "overwrite" } ],
  },
}
```

Listing 4: Example `MARLEY` job configuration file.

In certain special cases, e.g., studies of the Diffuse Supernova Neutrino Background (DSNB) [152], it may be desirable to generate `MARLEY` events with an isotropic distribution of initial neutrino directions. This behavior may be enabled by providing the string `"isotropic"` as the value for the `direction` key instead of a direction 3-vector.

6.3. Target composition

The nuclidic composition of the neutrino target material in a `MARLEY` simulation may be specified by defining a JSON object in the job configuration file labeled with the `target` key. The JSON object must define two arrays of equal size. The first of these, which is labeled with the `nuclides` key, includes one or more nuclear PDG codes (see section 7.1) with one code per species. The second array, which is labeled with the `atom_fractions` key, contains the corresponding nuclide fractions in the target material. After being automatically normalized to sum to unity, these will be used as the abundance weights f_r that appear in eqs. (74) and (75). If any of the elements of the `atom_fractions` array is negative, `MARLEY` will halt and print an error message.

Lines 10–13 of Listing 4 provide an example configuration for a neutrino target composed of pure ^{40}Ar. If the `target` key is omitted from the job configuration file, then the default behavior mentioned in section 4.1.6 will be used: every distinct nuclide that appears in the initial state of at least one configured reaction (see section 6.4) will be included in the neutrino target with equal abundance.

6.4. Reactions

To define the set of neutrino scattering processes that should be considered in a `MARLEY` simulation, the user must specify a list of reaction input files as a JSON string array labeled by the `reactions` key. This key is required to be present in all `MARLEY` job configuration files. For neutrino-nucleus reactions, each reaction input file provides the values of the allowed nuclear matrix elements $B(\mathrm{F})$ and $B(\mathrm{GT})$ needed to compute the differential cross section in eq. (5) for a particular target nucleus and interaction mode. For neutrino-electron elastic scattering, the reaction input file provides a list of nuclides for which this process should be enabled. Details of the file format are discussed in Appendix B.

In `MARLEY` 1.2.0, the code is capable of simulating reactions belonging to four distinct interaction modes: charged-current neutrino-nucleus scattering (ν CC), charged-current antineutrino-nucleus scattering ($\bar{\nu}$ CC), neutral-current (anti)neutrino-nucleus scattering (NC), and (anti)neutrino-electron elastic scattering (ES). As shown in eq. (10), the nuclear scattering modes may be distinguished by the isospin operator that appears in the nuclear matrix elements (e.g., t_- for ν CC).

6.4.1. Example reaction input files

Three example reaction input files for ν CC on ^{40}Ar, intended for use in simulations of the process $^{40}\mathrm{Ar}(\nu_e, e^-)^{40}\mathrm{K}^*$, are included in the `data/react` folder as part of the standard `MARLEY` distribution. At high excitation energies, all three files include the same set of

Table 3
Allowed values for the `neutrino` key in the neutrino source specification.

Neutrino	PDG code	string
ν_e	12	`"ve"`
$\bar{\nu}_e$	−12	`"vebar"`
ν_μ	14	`"vu"`
$\bar{\nu}_\mu$	−14	`"vubar"`
ν_τ	16	`"vt"`
$\bar{\nu}_\tau$	−16	`"vtbar"`

theoretical matrix elements taken from a QRPA[24] calculation by Cheoun, Ha, and Kajino [153]. At low excitation energies, the files contain different sets of nuclear matrix elements extracted from experimental data. The references used to obtain the data-driven matrix elements for each file are as follows:

`ve40ArCC_Bhattacharya1998.react` Measurement of ^{40}Ti β^+ decay reported in Ref. [154]
`ve40ArCC_Liu1998.react` Independent ^{40}Ti β^+ decay measurement reported in Ref. [155]
`ve40ArCC_Bhattacharya2009.react` Forward-angle (p, n) scattering data reported in Ref. [156].

A description of the data evaluation procedure used to prepare these files is available in Ref. [107, sec. III].

Two other reaction input files are included in the `data/react` folder. The file `ES.react` enables simulations of neutrino-electron scattering on an atomic ^{40}Ar target. Simulation of this process for other nuclides may be enabled in `ES.react` by appending additional nuclear PDG codes.

The file `CEvNS40Ar.react` provides a simple example of an NC configuration. Although `MARLEY` is capable of simulating inelastic NC reactions if provided with suitable input, `CEvNS40Ar.react` includes only one B(F) matrix element appropriate for simulating coherent elastic neutrino nucleus scattering on ^{40}Ar (see section 2.1.1). Altering the nuclear PDG code given in this file (see Appendix B) will immediately enable this process to be simulated for any spin-zero target nucleus.

6.4.2. Configuration example

Line 16 of Listing 4 provides an example `reactions` setting that enables ν CC and ES reactions on ^{40}Ar to be simulated simultaneously. As shown in the example, a full path specification is not needed for files that appear in the `${MARLEY}/data/react/` folder, where `${MARLEY}` is the value of the `MARLEY` environment variable at runtime (see section 5.3). If multiple files representing the same reaction mode and the same target nuclide are listed for the `reactions` key, a warning message will be printed and only the configuration from the first such file will be used in the simulation.

Every element of the `reactions` array must be a simple string literal containing a single file name. The use of wildcard characters, regular expressions, and environment variables is not currently allowed by the `MARLEY` job configuration file format.

6.5. Neutrino source

The required `source` key allows the user to provide a description of the incident neutrino energy spectrum using a JSON object. This object must always contain at least two keys: `neutrino` and `type`.

The `neutrino` key represents the neutrino species to be used as the projectile. This may be specified using either an integer PDG code or a string, as shown in Table 3.

The `type` key may be used to request one of several built-in neutrino spectra or to indicate to `MARLEY` that a user-defined spectrum should be used. In the latter case, users should typically not apply any cross section weighting themselves to the neutrino spectrum, as `MARLEY` will weight the incident spectrum using the appropriate reaction cross section at runtime (see section 4.1.3). In unusual cases where this behavior is not desirable, the user may disable cross section weighting by following the procedure described in section 6.7.3.

The following paragraphs describe each of the allowed options for the `type` key, together with any additional parameters needed for each source type.

6.5.1. Monoenergetic

A neutrino source whose `type` is `mono` or `monoenergetic` always produces a neutrino with a fixed energy E_ν. This energy is specified in MeV using the `energy` key. Lines 19–23 of Listing 4 provide an example configuration for a monoenergetic ν_e source with $E_\nu = 15$ MeV.

6.5.2. Fermi-Dirac distribution

The `fermi-dirac` or `fd` source type emits neutrinos whose energies are drawn from a Fermi-Dirac distribution with temperature T (MeV) and pinching parameter η. The incident neutrino spectrum is given by

$$\phi(E_\nu) = \frac{C\,E_\nu^2}{T^4\left[1+\exp\left(\frac{E_\nu}{T}-\eta\right)\right]} \qquad E_\nu^{\min} \le E_\nu \le E_\nu^{\max} \tag{85}$$

where C is a normalization constant chosen so that

[24] Quasiparticle Random Phase Approximation.

```
source: {
  type: "fermi-dirac",
  neutrino: "ve",
  Emin: 0,           // Minimum neutrino energy (MeV)
  Emax: 60,          // Maximum neutrino energy (MeV)
  temperature: 3.5,  // Temperature (MeV)
  eta: 4             // Pinching parameter (default 0)
},
```

Listing 5: Example Fermi-Dirac source specification.

```
source: {
  type: "beta-fit",
  neutrino: "ve",
  Emin: 0,   // Minimum neutrino energy (MeV)
  Emax: 60,  // Maximum neutrino energy (MeV)
  Emean: 15, // Mean neutrino energy (MeV)
  beta: 3.0, // Pinching parameter (default 4.5)
},
```

Listing 6: Example "beta fit" source specification.

```
source: {
  type: "dar",
  neutrino: "ve",
},
```

Listing 7: Example muon decay-at-rest source specification.

$$\int_{E_\nu^{\min}}^{E_\nu^{\max}} \phi(E_\nu)\, dE_\nu = 1. \tag{86}$$

The values of the parameters $E_\nu^{\min}$, $E_\nu^{\max}$, T, and η are specified using the keys `Emin`, `Emax`, `temperature`, and `eta`, respectively. Omitting the `eta` key will yield the default value $\eta = 0$. The other three parameters must be given explicitly. Listing 5 shows an example Fermi-Dirac source specification in which the parameter values $E_\nu^{\min} = 0$ MeV, $E_\nu^{\max} = 60$ MeV, $T = 3.5$ MeV, and $\eta = 4$ have been chosen.

The form for the pinched Fermi-Dirac distribution used by `MARLEY` is based on that of the "Livermore model" of the supernova neutrino energy spectrum [157].

6.5.3. "Beta fit" spectrum

The `beta-fit` or `bf` source type produces neutrinos with energies drawn from the spectrum

$$\phi(E_\nu) = C \left(E_\nu / \langle E_\nu \rangle\right)^{\beta-1} \exp\left(-\beta\, E_\nu / \langle E_\nu \rangle\right) \qquad E_\nu^{\min} \le E_\nu \le E_\nu^{\max} \tag{87}$$

with mean neutrino energy $\langle E_\nu \rangle$ (MeV) and dimensionless pinching parameter $\beta \ge 0$. The normalization constant C is chosen to satisfy eq. (86).

The values of the parameters $E_\nu^{\min}$, $E_\nu^{\max}$, $\langle E_\nu \rangle$, and β are specified using the keys `Emin`, `Emax`, `Emean`, and `beta`, respectively. Omitting the `beta` key will yield the default value $\beta = 4.5$. The other three parameters must be given explicitly. Negative β values are unphysical and will cause `MARLEY` to halt and issue an error message. Listing 6 shows an example "beta fit" source specification in which the parameter values $E_\nu^{\min} = 0$ MeV, $E_\nu^{\max} = 60$ MeV, $\langle E_\nu \rangle = 15$ MeV, and $\beta = 3$ have been chosen.

The "beta fit" distribution has been employed in a recent theoretical study [158] of supernova neutrino detection and is equivalent to the widely-used "Garching model" parameterization of supernova neutrino spectra [159]. The sole difference between the two parameterizations is in the convention for the pinching parameter. The Garching model employs a parameter α and makes the substitution $\beta \to \alpha + 1$ in eq. (87). For $\alpha = \beta - 1$, the distributions used by the two approaches are identical.

6.5.4. Muon decay-at-rest

The `decay-at-rest` or `dar` source type emits neutrinos according to a Michel spectrum for a muon decaying at rest in the lab frame. For a ν_e source (corresponding to the decay $\mu^+ \to e^+ + \nu_e + \bar{\nu}_\mu$), the energy spectrum is given by [160]

$$\phi(E_\nu) = 96\, E_\nu^2\, m_\mu^{-4}\, (m_\mu - 2E_\nu) \qquad 0 < E_\nu < m_\mu/2, \tag{88}$$

where m_μ is the muon mass. A $\bar{\nu}_\mu$ source has the spectrum

$$\phi(E_\nu) = 16\, E_\nu^2\, m_\mu^{-4}\, (3m_\mu - 4E_\nu) \qquad 0 < E_\nu < m_\mu/2. \tag{89}$$

Production of $\bar{\nu}_e$ and ν_μ (from $\mu^- \to e^- + \bar{\nu}_e + \nu_\mu$) may also be simulated using a `dar` neutrino source. Because the decay kinematics fully determine the shape of the Michel spectrum, no additional input parameters are needed for this source type. Listing 7 shows an example configuration for a muon decay-at-rest ν_e source.

```
source: {
  type: "histogram",
  neutrino: "ve",
  E_bin_lefts: [ 7., 8., 9. ], // Low edges of energy bins (MeV)
  weights: [ 0.2, 0.5, 0.3 ],  // Bin weights (dimensionless)
  Emax: 10.,                   // Upper edge of the final bin (MeV)
},
```

Listing 8: Example histogram source specification.

```
source: {
  type: "grid",
  neutrino: "ve",
  energies: [ 10., 15., 20. ],     // Energy grid points (MeV)
  prob_densities: [ 0., 1., 0. ],  // Probability densities
  rule: "lin", // Interpolation rule ("lin" default)
},
```

Listing 9: Example grid source specification.

Table 4
Allowed values of the `rule` key for the `grid` neutrino source type.

Rule	Interpretation
`"const"`	histogram-like
`"lin"`	linear in both E_ν and $\phi(E_\nu)$
`"linlog"`	linear in E_ν, logarithmic in $\phi(E_\nu)$
`"loglin"`	logarithmic in E_ν, linear in $\phi(E_\nu)$
`"log"`	logarithmic in both E_ν and $\phi(E_\nu)$

Neutrinos from muon decay-at-rest produced at facilities like the Oak Ridge Spallation Neutron Source (SNS) [161] and the Fermilab Neutrinos at the Main Injector (NuMI) beamline [37] offer a valuable opportunity to study neutrino-nucleus scattering at tens-of-MeV energies with a terrestrial source.

6.5.5. Histogram

The `histogram` or `hist` source type allows the user to define the incident neutrino spectrum as a histogram with one or more energy bins.

The n bins are specified by their lower edges $E_\nu(k)$ and weights $W(k)$ for $k \in \{1, 2, \ldots, n\}$. Within bin k, incident neutrino energies are distributed uniformly on the half-closed interval $[E_\nu(k), E_\nu(k+1))$. Weights that do not sum to unity will be automatically renormalized by the code. The keys `E_bin_lefts` and `weights` are used to provide JSON arrays of $E_\nu(k)$ and $W(k)$ values, respectively. The sizes of these arrays must be equal and are used by `MARLEY` to determine the bin count n. The energy bin edges $E_\nu(k)$ must be given in increasing order. The upper edge of the final energy bin $E_\nu(n+1)$ must also be specified using the key `Emax`.

Listing 8 shows an example histogram source specification in which three equally-spaced bins are defined between 7 and 10 MeV.

6.5.6. Grid

The `grid` source type emits neutrinos with energies drawn from a tabulated probability density function $\phi(E_\nu)$. To define this function, a grid of at least two monotonically increasing E_ν values is specified as a JSON array via the `energies` key. The corresponding $\phi(E_\nu)$ values (which need not be normalized by the user to integrate to unity) must also be given in an array of the same size using the `prob_densities` key.

To evaluate $\phi(E_\nu)$ at energies between the grid points, `MARLEY` employs an interpolation rule specified by the `rule` key. The default `"lin"` rule (linear in both E_ν and ϕ) is typically most useful, but `MARLEY` is also capable of interpolating logarithmically along one or both axes. A `"const"` interpolation rule is also available which treats $\phi(E_\nu)$ as a step function at each energy grid point. Table 4 lists all allowed values of the `rule` key together with their interpretations.

Listing 9 shows an example grid source specification which defines a symmetric triangular distribution between 10 and 20 MeV.

6.5.7. TH1 and TGraph

If `MARLEY`'s optional interface to the ROOT [3] data analysis framework is enabled (see section 5), then two additional neutrino source types, `th1` and `tgraph`, are allowed.

The `th1` type takes a one-dimensional ROOT histogram object (of C++ type `TH1`) as input and converts it into a `histogram` neutrino source using `MARLEY`'s native representation of the distribution.[25] The `tgraph` source type is similar, except that it converts a ROOT `TGraph` object into a `grid` neutrino source.

To use either of the `th1` or `tgraph` source types, the user must provide the name of an input ROOT file (including any needed path specification) using the `tfile` key. The string value given for the `namecycle` key, which must also be provided, will be used by `MARLEY` in a call to `TDirectoryFile::Get(const char* namecycle)` in order to retrieve the object of interest from the input ROOT file. Units of MeV should always be used for neutrino energies when preparing a `TH1` or `TGraph` object for use with `MARLEY`.

[25] Both histogram and grid neutrino sources are implemented using the `marley::InterpolationGrid` class.

```
source: {
  type: "th1",
  neutrino: "ve",
  tfile: "my_root_file.root",  // Name of the ROOT file
                               // containing the TH1 object

  namecycle: "MyHist",         // Name under which the TH1 object appears
                               // in the file (used to retrieve it)
},
```

Listing 10: Example th1 source specification.

Listing 10 shows an example th1 source specification which uses a neutrino energy histogram named `"MyHist"` stored in a ROOT file in the working directory called `my_root_file.root`.

6.6. Executable settings

The optional `executable_settings` key is used to provide a JSON object that controls the `marley` command-line executable. However, the entirety of the executable settings will be ignored if the configuration file is used to initialize `MARLEY` in a different context (e.g., from within a Geant4 application that links to the `MARLEY` shared libraries, see section 8).

Within the executable settings object, the `events` key is used to specify the number of events that the `marley` executable should generate before terminating. Because the `MARLEY` JSON parser expects the associated value to be an integer literal, using scientific notation to specify the number of events (e.g., `1e4` instead of `10000`) is not currently allowed. If the `events` key is omitted, a default value of 1000 will be assumed.

The `output` key in the executable settings object is used to describe zero or more output files that will receive the generated events. The corresponding value should be a JSON array containing one JSON object per output file. Each object in the array should define the following keys:

file The name of the output file (with any needed path specification).

format The format to use when storing events in the output file. Valid values for this key are `"ascii"`, `"hepevt"`, `"json"`, and (if `MARLEY`'s ROOT interface is active) `"root"`. Descriptions of each of the allowed output file formats are given in section 7.2.

mode The approach that the `marley` executable should use if the output file is not initially empty. For the ASCII and HEPEVT formats, valid values are `"overwrite"` (erase any previously existing file contents) and `"append"` (continue output immediately after any existing file contents). For the JSON and ROOT formats, valid values are `"overwrite"` and `"resume"`. If the `"resume"` mode is chosen, the generator will restore its previous state from an incomplete run (e.g., a run that was interrupted by the user by pressing ctrl+C) that was saved to the output file and continue from where it left off. If the `"resume"` mode is selected for more than one output file, then `MARLEY` will halt and print an error message.

Two additional keys may be used in specific contexts:

force Boolean value used only for the `"overwrite"` mode. If it is `true`, then the marley executable will not prompt the user before overwriting existing data. If this key is omitted, a value of `false` is assumed.

indent Nonnegative integer value used only for the JSON output format. It represents the number of spaces that should be used as a tab stop when formatting the JSON output. If this key is omitted, all unnecessary whitespace will be suppressed. This default behavior results in the most compact JSON-format output files.

If the `output` key in the `executable_settings` object is omitted entirely, then the default configuration

```
output: [ { file: "events.ascii", format: "ascii", mode: "overwrite" } ]
```

will be used.

Lines 26–32 of Listing 4 define an example `executable_settings` object that includes the default settings described in this section.

6.7. Less commonly-used parameters

Although the job configuration file shown in Listing 4 provides usage examples for all of the key-value pairs typically needed to define a `MARLEY` simulation, a number of additional parameters are recognized by the configuration parser.[26] These parameters, which are intended for advanced users or for addressing unusual situations, are documented in the remainder of this section.

6.7.1. Logging

The singleton `marley::Logger` class provides rudimentary support for writing diagnostic messages to zero or more output streams including standard output, standard error, and plaintext log files. This is done in a prioritized way by configuring a *logging level* for each output stream. Messages are likewise categorized by a logging level. In order of severity, the logging levels recognized by `marley::Logger` are `debug`, `info`, `warning`, `error`, and `disabled`, with the last available for output streams but not for messages. When a

[26] That is, the `marley::JSONConfig` class.

message with logging level `MSG_LEVEL` is passed by `MARLEY` to the `Logger`, the message will be ignored by all streams whose logging levels are more severe than `MSG_LEVEL`. The message will be written to all other configured streams.

Nearly all output written to the terminal by the `marley` executable is handled by the `Logger` class. The main exception is the status display discussed in section 5.4 (see Listing 1).

The default behavior of the `Logger` class may be adjusted via the optional `log` key in the job configuration file. This key is used to specify an array of JSON objects with one element per output stream. Each object in the array may define the following keys:

file A string giving the name of a text file that will receive the output logging messages. If the string value is `"stdout"` (`"stderr"`), then the messages will be written to standard output (standard error) rather than a text file.

level A string specifying the logging level that should be used as a threshold for writing messages to the stream. Valid values are `"debug"`, `"info"`, `"warning"`, `"error"`, and `"disabled"`.

overwrite A boolean value indicating whether new messages should be appended to the end of the output file (`false`) or whether any previously existing file contents should be erased (`true`). This key is ignored when writing to standard output and standard error. A default value of `false` is assumed when this key is not present.

If the `log` key is omitted from the job configuration file, then `MARLEY` will use default settings equivalent to

```
log: [ { file: "stdout", level: "info" } ]
```

6.7.2. Disabling simulation of nuclear de-excitations

As described in section 2, `MARLEY` models neutrino-nucleus scattering as proceeding via a prompt $2 \to 2$ reaction and a subsequent sequence of zero or more binary decays of the remnant nucleus. For certain calculations,[27] it may be convenient to simulate events in which only the primary $2 \to 2$ scattering process is considered. This behavior may be enabled in the job configuration file by using the optional `do_deexcitations` key together with a boolean value. If the accompanying value is `true`, then the `MARLEY` event loop will proceed normally, and nuclear de-excitations will be simulated. If it is `false`, then the de-excitation loop will be skipped.

6.7.3. Disabling cross-section weighting of the neutrino spectrum

By default, `MARLEY` samples the reacting neutrino energy E_ν for each event from a probability density function $P(E_\nu) \propto \sigma(E_\nu)\phi(E_\nu)$, where $\sigma(E_\nu)$ is the abundance-weighted total cross section and $\phi(E_\nu)$ is the incident neutrino spectrum (see section 4.1.3). For some use cases, it may be desirable to sample E_ν directly from $\phi(E_\nu)$ without any cross-section weighting. This may be achieved by adding the optional `weight_flux` key, which takes a boolean value, to the contents of the `source` JSON object (section 6.5). If the `weight_flux` value is set to `true`, then E_ν will be sampled from the usual probability density function $P(E_\nu)$. If the `weight_flux` value is set to `false`, then E_ν will be drawn directly from $\phi(E_\nu)$.

6.7.4. Orbital angular momentum and multipolarity cutoffs

When computing differential decay widths for nuclear fragment (γ-ray) emission from a compound nucleus, `MARLEY` truncates the infinite sum over orbital angular momenta (multipolarities) that appears in eq. (29) (eq. (49)) by imposing a cutoff value $\ell_{\max}$ ($\lambda_{\max}$). The user may modify the default value ($\ell_{\max} = \lambda_{\max} = 5$) by including two keys in the job configuration file. The `fragment_lmax` key is used to specify a nonnegative integer which will be used as the maximum orbital angular momentum $\ell_{\max}$ to consider in nuclear fragment emission. Similarly, the `gamma_lmax` key is used to specify a positive integer which will serve as the maximum multipolarity $\lambda_{\max}$ considered in γ-ray emission.

6.7.5. Coulomb correction factor

As described in section 2.1.2, `MARLEY` accounts for final-state Coulomb effects in charged-current neutrino-nucleus interactions by the inclusion of a Coulomb correction factor F_C in the expression for the differential cross section. By default, this factor is computed according to eq. (27), which uses the smaller of the two corrections given by the Fermi function (eq. (18)) and the modified effective momentum approximation (MEMA, eq. (26)). The `MARLEY` prescription for the Coulomb correction factor may be controlled in the job configuration file by use of the `coulomb_mode` key. The following string values are allowed for this key:

`"none"` No Coulomb corrections will be applied ($F_C = 1$ in all cases).

`"Fermi"` The Fermi function defined in eq. (18) will always be used to compute the Coulomb correction factor ($F_C = F_{\text{Fermi}}$).

`"EMA"` The effective momentum approximation (EMA) will be used for Coulomb corrections ($F_C = F_{\text{EMA}}$, see eq. (25)).

`"MEMA"` The modified effective momentum approximation (MEMA) will be used for Coulomb corrections ($F_C = F_{\text{MEMA}}$, see eq. (26)).

`"Fermi-EMA"` The Fermi function will be compared to the EMA, and the approach that yields the smaller correction (i.e., the F_C value closer to unity) will be used. The Coulomb correction factor F_C is calculated as in eq. (27) with the substitution $F_{\text{MEMA}} \to F_{\text{EMA}}$.

`"Fermi-MEMA"` Similar to `"Fermi-EMA"`, except that the MEMA is used instead of the EMA. The Coulomb correction factor F_C is calculated as in eq. (27). This is `MARLEY`'s default approach.

6.7.6. Status update interval

As discussed in section 5.4, the `marley` executable prints a text-based status display at the bottom of the screen. By default, this display is updated after each set of $n_{\text{update}} = 100$ events has been generated. The user may modify the value of n_{update} by adding the `status_update_interval` key to the `executable_settings` object in the job configuration file (see section 6.6). Only positive integer values of n_{update} may be specified for this key.

[27] For example, those in which only the final-state lepton is of interest.

Table 5
The most commonly encountered PDG codes used by MARLEY to identify kinds of particles.

PDG code	Particle
11	e^-
12	ν_e
13	μ^-
14	ν_μ
13	τ^-
14	ν_τ
22	γ
2112	n
2212	p
1000010020	d
1000010030	t
1000020030	h
1000020040	α

6.7.7. *Manually specifying a maximum value of the neutrino energy PDF*

To select a reacting neutrino energy for each event, MARLEY relies on rejection sampling from the probability density function given in eq. (73). In situations where automatic estimation of the global maximum $P_{\max}$ of this PDF is found to be inadequate, MARLEY will print a set of messages to alert the user (see section 5.4.1). These messages will include an improved estimate of $P_{\max}$ (see line 6 of Listing 2) which may be manually supplied by the user in the job configuration file as a floating-point value associated with the `energy_pdf_max` key. Doing so will disable numerical estimation of $P_{\max}$ in favor of using the specified value, which may help to resolve some rejection sampling problems.

6.7.8. *Keys specific to `mardumpxs`*

Several job configuration file keys are available to modify the default behavior of the `mardumpxs` utility described in section 7.4.2. They are otherwise ignored by MARLEY. The `mardumpxs`-specific keys are listed below together with their interpretations:

`xsec_dump_pdg` Labels an integer value that specifies the PDG code (see the second column of Table 3) for the projectile of interest. If this key is not present, then a ν_e (PDG code 12) is assumed by `mardumpxs`.

`xsec_dump_KEmin` Used to set the minimum projectile kinetic energy (MeV) that should be included in the output. By default, `mardumpxs` sets the minimum kinetic energy to 0 MeV.

`xsec_dump_KEmax` Used to set the maximum projectile kinetic energy (MeV) that should be included in the output. By default, `mardumpxs` sets the maximum kinetic energy to 100 MeV.

`xsec_dump_steps` Used to set the number of equally-spaced points at which the total cross section should be evaluated as a function of projectile kinetic energy. Only positive integer values are allowed. If this key is absent, then `mardumpxs` will use a default value of 10,000.

7. Interpreting the output

The neutrino scattering events generated by the `marley` command-line executable may be saved to disk in four distinct output formats. Section 6.6 explains how one or more of these output formats may be selected by the user in the job configuration file. Following a brief discussion of the particle numbering scheme used by MARLEY in section 7.1, documentation for each of the output file formats is provided in section 7.2. Section 7.3 presents a C++-based API which may be used to parse all of the standard output formats via a unified interface. Section 7.4 provides guidance on how histograms of quantities of interest from the generated events may be converted into easily interpretable physics distributions, e.g., differential cross sections and event rates.

7.1. *Particle numbering scheme*

Like many other modern physics event generators, MARLEY identifies particle species using a standard set of integer codes defined by the Particle Data Group (PDG) [162, sec. 44, pp. 661–664]. Each kind of particle is assigned a unique positive integer, and the corresponding antiparticle is assigned a negative integer with the same absolute value. These *PDG codes* are used by MARLEY both internally and in output files.

Table 5 provides the interpretation of the most common PDG codes that appear in MARLEY events. In general, a nucleus with proton number Z and mass number A is labeled with the PDG code

$$\mathrm{P_{ID}} = 10{,}000\,Z + 10\,A + 1{,}000{,}000{,}000\,. \tag{90}$$

The only two exceptions to the rule given in eq. (90) are for a free neutron and ^{1}H, which are represented by the codes 2112 and 2212, respectively.

7.2. *Output file formats*

In the current version of MARLEY, generated events may be saved to disk using four possible file formats:

```
1  5.98368867447267264e-19
2  2 4 3.79748000000000019e+00 2 +
3  12 1.00000000000000000e+01 0.00000000000000000e+00 0.00000000000000000e+00 1.00000000000000000e+01 0.00000000000000000e
       +00 0
4  1000180400 3.72247225431518091e+04 0.00000000000000000e+00 0.00000000000000000e+00 0.00000000000000000e+00
       3.72247225431518091e+04 0
5  11 5.20690886815266740e+00 -4.63535385338761508e+00 1.35706546730579314e+00 -1.87687187323011373e+00
       5.10998927645907708e-01 -1
6  1000190400 3.72257175068044089e+04 4.98066184879143847e+00 -2.27058729207187593e+00 9.28456410767741858e+00
       3.72257159465162476e+04 1
7  22 1.53694352434620662e+00 -1.11400145523226901e+00 9.50751758100756628e-01 4.66119350851738223e-01 0.00000000000000000
       e+00 0
8  22 2.26118395490673985e+00 7.68693459828445835e-01 -3.72299333346743089e-02 2.12618841470095354e+00 0.00000000000000000
       e+00 0
9  2 3 1.03964200000000009e+01 2 +
10 12 2.99304885549511290e+01 0.00000000000000000e+00 0.00000000000000000e+00 2.99304885549511290e+01 0.00000000000000000e
       +00 0
11 1000180400 3.72247225431518091e+04 0.00000000000000000e+00 0.00000000000000000e+00 0.00000000000000000e+00
       3.72247225431518091e+04 0
12 11 1.85223477954320721e+01 -9.86795370224160173e+00 -1.55256312663347771e+01 -2.09630900945355991e+00
       5.10998927645907708e-01 -1
13 1000190390 3.62940260743929030e+04 -1.74322423197586254e+01 -4.21863040207651636e+01 5.85225771236273147e+01
       3.62939501877290750e+04 1
14 2112 9.42104609518426400e+02 2.73001960220002289e+01 5.77119352870999407e+01 -2.64957795592226226e+01
       9.39565378653339735e+02 0
```

Listing 11: Example ASCII-format output file.

ASCII The native text-based format for `MARLEY` events.

HEPEVT A legacy format for interfacing event generators with each other and with other software. Despite its relative age and emergence in the specific context of QCD event generation for Large Electron-Positron Collider experiments [163], compatibility with this format is maintained in many modern high energy physics software tools. Examples include the HepMC3 [164] event record library and (via its `TextFileGen` module) the LArSoft toolkit discussed in section 8.3.

JSON In addition to serving as the input language for `MARLEY` job configuration files (with slight extensions to the standard grammar, see section 6 for details), JSON is also available as an output file format.

ROOT If `MARLEY`'s interface to ROOT is active, a binary representation of the full `marley::Event` objects may be saved to disk in an output `TTree`. A simplified "flat" form of the ROOT output that can be read without recourse to the `MARLEY` shared libraries may also be produced using the `marsum` command-line tool (see section 7.3.5).

7.2.1. ASCII file format

An ASCII-format output file begins with the line

```
FluxAvgXsec
```

in which `FluxAvgXsec` is the flux-averaged total reaction cross section in natural units (MeV^{-2}/atom, see section 7.4). This line is followed by one or more event records, each of which begins with the header

```
Ni Nf Ex twoJ P
```

where `Ni` (`Nf`) is the number of particles in the initial (final) state. The three remaining fields in the event header report properties of the recoiling nucleus immediately following the prompt $2 \to 2$ scattering reaction. The quantity `Ex` is the excitation energy (MeV), `twoJ` is an integer equal to two times the total spin,[28] and `P` is a single character representing either positive (`+`) or negative (`-`) parity.

Following the event header, each of the `Ni` initial-state particles is described by a single line of the form

```
PDG Etot Px Py Pz M Q
```

where `PDG` is the PDG code (see section 7.1) identifying the particle species and `Etot`, `Px`, `Py`, and `Pz` are the Cartesian components of the particle 4-momentum (in MeV). The particle mass `M` (MeV) and (net) electric charge `Q` (in units of the elementary charge) are also included in each line. To complete the event record, `Nf` lines describing the final-state particles appear in the same format used for the initial-state particles.

To preserve full numerical precision when converting back and forth between `marley::Event` objects held in memory and ASCII-format event files, all floating point numbers are output in scientific notation with the required number of base-10 digits needed to uniquely represent all distinct values of the C++ type `double`.[29] However, this level of precision is not enforced by the code when reading events as input from an ASCII file.

Listing 11 shows an example `MARLEY` output file in ASCII format. Line 1 gives the value of 5.984×10^{-19} $\text{MeV}^{-2} = 2.330 \times 10^{-40}$ cm^2 for the flux-averaged total cross section. Line 2 begins the record for the first event, which involves a transition to the ^{40}K nuclear level

[28] This allows representation of half-integer spins using a C++ `int`.

[29] This number is called `std::numeric_limits<double>::max_digits10` in the C++ standard library.

```
1  0 7
2  3 12 0 0 0 0 0.0000000000000000e+00 0.0000000000000000e+00 1.0000000000000002e-02 1.0000000000000002e-02
       0.0000000000000000e+00 0. 0. 0. 0.
3  3 1000180400 0 0 0 0 0.0000000000000000e+00 0.0000000000000000e+00 0.0000000000000000e+00 3.7224722543151806 1e+01
       3.7224722543151806 1e+01 0. 0. 0. 0.
4  11 0 2 1 0 0 0.0000000000000000e+00 0.0000000000000000e+00 0.0000000000000000e+00 3.7974800000000019e+00
       5.9836886744726726 4e-19 0. 0. 0. 0.
5  1 11 0 0 0 0 -4.63535385338761496e-03 1.35706546730579320e-03 -1.87687187323011366e-03 5.20690886815266749e-03
       5.1099892764590771 0e-04 0. 0. 0. 0.
6  1 1000190400 0 0 0 0 4.98066184879143812e-03 -2.27058729207187593e-03 9.28456410767741942e-03 3.7225717506804407 7e+01
       3.7225715946516245 9e+01 0. 0. 0. 0.
7  1 22 0 0 0 0 -1.11400145523226908e-03 9.50751758100756655e-04 4.66119350851738206e-04 1.53694352434620668e-03
       0.0000000000000000e+00 0. 0. 0. 0.
8  1 22 0 0 0 0 7.68693459828445808e-04 -3.72299333346743093e-05 2.12618841470095347e-03 2.26118395490674000e-03
       0.0000000000000000e+00 0. 0. 0. 0.
9  0 6
10 3 12 0 0 0 0 0.0000000000000000e+00 0.0000000000000000e+00 2.99304885549511283e-02 2.99304885549511283e-02
       0.0000000000000000e+00 0. 0. 0. 0.
11 3 1000180400 0 0 0 0 0.0000000000000000e+00 0.0000000000000000e+00 0.0000000000000000e+00 3.7224722543151806 1e+01
       3.7224722543151806 1e+01 0. 0. 0. 0.
12 11 0 2 1 0 0 0.0000000000000000e+00 0.0000000000000000e+00 0.0000000000000000e+00 1.0396420000000009e+01
       5.9836886744726726 4e-19 0. 0. 0. 0.
13 1 11 0 0 0 0 -9.86795370224160216e-03 -1.55256312663347770e-02 -2.09630900945356009e-03 1.85223477954320724e-02
       5.1099892764590771 0e-04 0. 0. 0. 0.
14 1 1000190390 0 0 0 0 -1.74322423197586264e-02 -4.21863040207651613e-02 5.85225771236273160e-02 3.6294026074392903 1e+01
       3.6293950187729073 8e+01 0. 0. 0. 0.
15 1 2112 0 0 0 0 2.73001960220002303e-02 5.77119352870999400e-02 -2.64957795592226236e-02 9.42104609518426450e-01
       9.39565378653339778e-01 0. 0. 0. 0.
```

Listing 12: Example HEPEVT-format output file.

with an excitation energy of 3.797 MeV above the ground state and spin-parity 1^+. Lines 3–4 describe the initial state: a 10 MeV ν_e traveling in the $+z$ direction toward a ^{40}Ar atom at rest. Lines 5–8 describe the final-state particles: a 5.2 MeV electron, the recoiling ^{40}K ion, and two de-excitation γ-rays with energies of 1.54 and 2.26 MeV. The second and final event in the file, which appears on lines 9–14, involves a ν_e-^{40}Ar collision which produces an electron, a ^{39}K ion, and a neutron.

7.2.2. HEPEVT file format

For the sake of brevity, only those aspects of the HEPEVT format needed to interpret `MARLEY` output are discussed in this section. A full description of the HEPEVT standard is available in Ref. [163, pp. 327–330].

A HEPEVT-format output file consists of one or more text-based event records. Each of these records begins with the header

```
NEVHEP NHEP
```

where `NEVHEP` is the event number (untracked by `MARLEY` and thus always set to zero) and `NHEP` is the number of particles in the event. The header is followed by `NHEP` lines, each representing a single particle. These have the format

```
ISTHEP IDHEP JMOHEP1 JMOHEP2 JDAHEP1 JDAHEP2 PHEP1 PHEP2 PHEP3 PHEP4 PHEP5 VHEP1 VHEP2 VHEP3 VHEP4
```

where `ISTHEP` is an integer code identifying the particle status and `IDHEP` is the PDG particle ID code. In agreement with the HEPEVT standard, `MARLEY` uses status code 1 for the final-state particles and 3 for the initial-state particles, the latter of which are not considered part of the event history [163]. The `JMOHEP1`, `JMOHEP2`, `JDAHEP1`, and `JDAHEP2` entries record the indices j ($1 \leq j \leq$ `NHEP`) of particles in the event record that correspond to the first mother, second mother, first daughter, and last daughter of the current particle, respectively. These indices are set to zero in cases where they do not apply (e.g., a particle which has not decayed will have `JDAHEP1` = `JDAHEP2` = 0). Entries `PHEP1` through `PHEP3` record the x, y, and z components of the particle 3-momentum, while `PHEP4` gives the total energy and `PHEP5` gives the particle mass (all in GeV). Entries `VHEP1` through `VHEP3` store the x, y, and z positions of the particle production vertex (mm), and `VHEP4` gives the production time (mm/c).

Because `MARLEY` currently treats all nuclear de-excitations as instantaneous and does not perform any particle tracking, `VHEP1` through `VHEP4` are always identically zero in HEPEVT output files. Intermediate de-excitation steps are also not currently stored in the event record, so `JMOHEP1`, `JMOHEP2`, `JDAHEP1`, and `JDAHEP2` are also identically zero in most cases.

In addition to the initial- and final-state particles, `MARLEY` adds a dummy particle with `ISTHEP` = 11 to each HEPEVT event record. All data fields are zero for this particle except for (1) `JMOHEP1`, which contains the nuclear spin multiplied by two, (2) `JMOHEP2`, which reports the parity of the nucleus as an integer, (3) `PHEP4`, which gives the excitation energy of the nucleus (MeV), and (4) `PHEP5`, which records the flux-averaged total cross section (see section 7.4) in units of MeV^{-2}/atom. As is the case for the ASCII format, the excitation energy, spin, and parity values refer to the nuclear state that existed immediately after the initial $2 \rightarrow 2$ scattering reaction.

Listing 12 shows an example `MARLEY` output file in HEPEVT format. The same two events from the ASCII-format example file (see Listing 11) are used for easy comparison of the formats.

7.2.3. JSON file format

Unlike `MARLEY` job configuration files, which allow a few non-standard JSON language extensions (see section 6 for details), the JSON-format output files fully conform to the standard grammar. Each output file includes two top-level keys: `events`, which is associated with an array of event objects, and `gen_state`, which stores a JSON object representation of the generator state at the moment that the file was created.

```
// Standard library includes
#include <iostream>
#include <string>

// ROOT includes
#include "TFile.h"
#include "TParameter.h"

void print_metadata() {

  TFile event_file( "events.root", "read" );

  std::string* config = NULL;
  std::string* state = NULL;
  std::string* seed = NULL;
  TParameter<double>* xsec = NULL;

  event_file.GetObject( "MARLEY_config", config );
  event_file.GetObject( "MARLEY_state", state );
  event_file.GetObject( "MARLEY_seed", seed );
  event_file.GetObject( "MARLEY_flux_avg_xsec", xsec );

  std::cout << "MARLEY configuration: " << *config << '\n';
  std::cout << "Generator internal state: " << *state << '\n';
  std::cout << "Random number seed: " << *seed << '\n';
  std::cout << "Flux-averaged total cross section: " << xsec->GetVal()
    << " MeV^{-2} / atom\n";

}
```

Listing 13: Example access to `MARLEY` metadata stored in a ROOT-format output file.

Each element of the `events` array is a JSON object containing five key-value pairs. The first three of these, `Ex`, `twoJ`, and `parity`, provide the excitation energy (MeV), two times the total spin, and the parity of the final nucleus after the initial $2 \rightarrow 2$ scattering reaction but before any de-excitations have taken place. The other two keys, `initial_particles` and `final_particles`, are used to store arrays of particles represented as JSON objects. Each particle object defines the following keys: (1) `charge`: the (net) electric charge in units of the elementary charge, (2) `pdg`: the PDG code identifying the particle type, (3) `E`: the total energy, (4) `px`: the x momentum component, (5) `py`: the y momentum component, (6) `pz`: the z momentum component, and (7) `mass`: the particle mass. The particle 4-momentum components and mass are all given in natural units (MeV).

The `gen_state` JSON object includes several key-value pairs. The `config` key refers to a JSON object which reproduces the full contents (except for comments) of the job configuration file used to generate the events. The `event_count`, `flux_avg_xsec`, and `seed` keys label the total number of events generated at the time the file was written, the flux-averaged total cross section (MeV^{-2}/atom, see section 7.4), and the integer random number seed used to initialize the event generator. A final key, `generator_state_string`, records a string representation of the internal state of the `std::mt19937_64` object used by `MARLEY` to obtain pseudorandom numbers.

An example `MARLEY` output file in JSON format (`example.json`) is included in the supplemental materials.

7.2.4. ROOT file format

If `MARLEY` has been built against the appropriate shared libraries from the ROOT data analysis toolkit (see section 5), then the `marley` command-line executable may also produce output files in the standard ROOT compressed binary format.

Within a ROOT-format output file, access to the generated events is managed by an instance of the ROOT `TTree` class, which provides a hierarchical data structure for storing many objects belonging to the same C++ type. In general, a `TTree` may own one or more branches (represented by the `TBranch` class), each of which owns one or more leaves (represented by `TLeaf`). Branches may be read from a file independently of one another, allowing for efficient access to elements of a large dataset stored in a suitably-organized `TTree` [165].

The ROOT-format output files generated by `MARLEY` contain a single `TTree` called `MARLEY_event_tree`. This `TTree` contains a single branch called `event`, which stores one `marley::Event` object per tree entry. Although direct access to the events is possible by manipulating the `MARLEY_event_tree`, use of the simplified C++ API described in section 7.3 is recommended instead.

In addition to the `MARLEY` events themselves, four pieces of metadata are stored in a ROOT-format output file:

MARLEY_config (*std::string*) A JSON-format string which stores the contents (except for comments) of the job configuration file used to generate the events (see section 6)

MARLEY_state (*std::string*) Serialized internal state (obtained using the stream insertion operator `<<`) of the `std::mt19937_64` object (see section 3) owned by the `marley::Generator` object used to create the events

MARLEY_seed (*std::string*) A string representation of the integer random number seed used to initialize the simulation

MARLEY_flux_avg_xsec (*TParameter<double>*) The flux-averaged total cross section (MeV^{-2}/atom) needed to normalize physics distributions computed from the events (see section 7.4)

In Listing 13, an example C++ function is shown which retrieves all of these metadata objects from a file called `events.root` and prints their contents to standard output.

```
1  #include <iostream>
2
3  #include "marley/Event.hh"
4  #include "marley/EventFileReader.hh"
5
6  int main(int argc, char** argv) {
7
8    if ( argc < 2 ) return 1;
9
10   std::string input_file_name( argv[1] );
11
12   marley::EventFileReader efr( input_file_name );
13   marley::Event event;
14
15   double avg_xsec = efr.flux_averaged_xsec();
16   std::cout << "flux-averaged total cross section = "
17     << avg_xsec << " * 10^{-42} cm^2 / atom\n";
18
19   while ( efr >> event ) {
20     std::cout << event << '\n';
21   }
22
23   return 0;
24 }
```

Listing 14: The source code for `examples/executables/minimal/efr.cc`, an example C++ program that uses the `marley::EventFileReader` class.

The ROOT output files described above may be converted into an alternative "flat" format which is readable by ROOT without a need for the `MARLEY` shared libraries. This is done by running the `marsum` command-line tool described in section 7.3.5.

7.3. C++ analysis API

Although the file format descriptions given in section 7.2 should be sufficient to enable processing of `MARLEY` events using any programming language, a C++ API allowing manipulation of events stored in any of the standard output formats has been developed for the convenience of users. The API is usable within compiled code as well as via the interactive C++ interpreters included with ROOT.[30]

7.3.1. The EventFileReader class

Programmatic access to `MARLEY` event records stored in a file is provided by the `marley::EventFileReader` class. The constructor of this class takes as its sole argument a `std::string` containing the name (including any needed path specification) of the file to be parsed. Events may be read one-by-one from the file using the stream extraction operator `>>`, which returns a boolean value indicating whether a new event was successfully loaded.

Listing 14 shows the source code for `examples/executables/minimal/efr.cc`, an example C++ program that illustrates the recommended usage of the `EventFileReader` class. Line 12 creates a new `EventFileReader` object that will read `MARLEY` events from a file whose name is given as the first command-line argument when the program is run. The while loop on lines 19–21 streams all events in the file to standard output, where they will be printed in ASCII format. Line 15 contains a call to the `EventFileReader` member function `flux_averaged_xsec`, which returns the flux-averaged total cross section (see section 7.4) used to generate the events in units of 10^{-42} cm^2/atom. Passing a boolean value of `true` to this function will cause it to return the cross section in natural units (MeV^{-2}/atom) instead.

To build the `efr.cc` example program, it is necessary to link the executable to the `MARLEY` shared library. This is most easily done using the `marley-config` script with the syntax described in section 5.5.1.

The `EventFileReader` class can parse `MARLEY` output files written in the ASCII, HEPEVT, and JSON formats. If `MARLEY` has been built against ROOT (see section 5), then support for the ROOT output format in addition to the others is available via the `RootEventFileReader` class. This class is derived from `EventFileReader` and implements an identical user interface. Making the replacement `EventFileReader` → `RootEventFileReader` on lines 4 and 12 of Listing 14 will yield a program that is capable of reading events from any `MARLEY` output file. Compiling this modified program requires linking to the shared library containing the `MARLEY` interface to ROOT. The build command given in section 5.5.1 will automatically include the necessary compiler flags for linking to ROOT if `MARLEY` was successfully built with ROOT support.

7.3.2. Accessing information from an event record

As discussed in section 2, `MARLEY` conceives of each scattering event as consisting of a primary $2 \to 2$ reaction possibly followed by nuclear de-excitations. The particle content of the primary reaction may be written in the general form

$$a + b \to c + d \tag{91}$$

[30] The CINT [166] interpreter is distributed with version 5 of ROOT, while version 6 uses Cling [167].

where particles a, b, c, and d, are respectively termed the *projectile*, *target*, *ejectile*, and *residue*. Where a mass difference exists, MARLEY chooses the projectile (ejectile) to be the lighter of the two particles in the initial (final) state. Otherwise, the choice is arbitrary. All four-vector components stored in a MARLEY event record are expressed in the laboratory frame, i.e., the rest frame of the target.

A full MARLEY event record is represented in the code itself by the `marley::Event` class. Instances of this class own two vectors of pointers to `marley::Particle` objects, with one (`initial_particles_`) describing the initial state and the other (`final_particles_`) describing the final state. Within the `initial_particles_` vector, the first and second elements always correspond to the projectile and target, respectively. A similar ordering (ejectile followed by residue) is enforced for the first two elements of the `final_particles_` vector. If present, elements of `final_particles_` beyond the second correspond to nuclear de-excitation products, which are listed in the order that they were emitted. If nuclear de-excitations were enabled in the simulation (as they are by default, see section 4.1.7), then the `Particle` object for the residue stores its properties after it has reached the ground state.

The `Event` class provides the following member functions for accessing `Particle` objects stored in the event record:

projectile(), target(), ejectile(), residue() Returns a reference to a specific particle involved in the primary $2 \to 2$ reaction
get_initial_particles() Returns a reference to the owned vector of initial particles
get_final_particles() Returns a reference to the owned vector of final particles
initial_particle_count() Returns the number of initial-state particles in the event
final_particle_count() Returns the number of final-state particles in the event
initial_particle(size_t idx) Returns a constant reference to the initial-state particle at position `idx`
final_particle(size_t idx) Returns a constant reference to the final-state particle at position `idx`

The `Particle` class stores a 4-momentum (represented as a C-style array of type `double[4]`) together with variables representing particle properties. It implements the following member functions for data retrieval:

charge() Electric charge (in units of the proton charge)
mass() Mass (MeV)
pdg_code() PDG code [162, sec. 44, pp. 661–664] representing the particle species
px() 3-momentum x-component (MeV)
py() 3-momentum y-component (MeV)
pz() 3-momentum z-component (MeV)
kinetic_energy() Kinetic energy (MeV)
total_energy() Total energy (MeV)

All of these functions return a value of type `double` except for `pdg_code()`, which returns an `int`.

In addition to `Particle` objects, the `Event` class also stores information about the state of the residue immediately after its creation by the prompt $2 \to 2$ reaction. Access to this information is provided by these member functions:

Ex() Initial residue excitation energy (`double`, MeV), measured with respect to its own ground state
twoJ() Two times the initial residue spin (`int`)
parity() Initial residue parity (`marley::Parity`)

The `Parity` object returned by `Event::parity()` provides a type-safe representation of a parity value ± 1. It may be converted to a boolean (`true` $\leftrightarrow +1$, `false` $\leftrightarrow -1$) or integer representation via an explicit cast. A `Parity` object may also be converted to a `char` value (`+` $\leftrightarrow +1$, `-` $\leftrightarrow -1$) via the member function `to_char()`.

Full technical documentation for the `Event`, `Particle`, and `Parity` classes is available on the official MARLEY website [148]. Similar HTML documentation may be automatically generated for offline use from the MARLEY source code using Doxygen [168]. After Doxygen has been installed, one may build the MARLEY API documentation files by executing

```
make doxygen
```

from within the `build/` folder. The generated documentation may then be viewed using any web browser by opening the file `docs/_build/html/doxygen/index.html`

7.3.3. Executable example: `marprint`

The `marprint` program (`examples/executables/marprint.cc`) included with MARLEY provides a detailed example of how the C++ analysis API may be used in compiled code. After instantiating either an `EventFileReader` or a `RootEventFileReader` object (depending on whether MARLEY was built against ROOT), the program prints all information stored in each event to standard output using a human-readable format. Usage examples for many of the functions listed in section 7.3.2 are provided within the `marprint.cc` source file. After sourcing the `setup_marley.sh` script, one may build the `marprint` example program via the commands

```
cd ${MARLEY}/build
make marprint
```

The `marprint` executable takes the names of one or more files containing MARLEY events as command-line arguments. For example, one may print all of the events stored in the files `/home/events1.ascii` and `/home/events2.json` by executing

```
marprint /home/events1.ascii /home/events2.json
```

Listing 15 shows the printout generated by `marprint` while parsing the first event given in the example ASCII-format output file shown above (Listing 11).

```
*** Event 0 has 2 initial particles and 4 final particles. ***
The residual nucleus initially had excitation energy 3.79748 MeV and spin-parity 1+
Initial particles
  particle with PDG code = 12 has total energy 10 MeV,
    3-momentum = (0 MeV, 0 MeV, 10 MeV),
    mass = 0 MeV, and charge = 0 times the proton charge.
  particle with PDG code = 1000180400 has total energy 37224.7 MeV,
    3-momentum = (0 MeV, 0 MeV, 0 MeV),
    mass = 37224.7 MeV, and charge = 0 times the proton charge.
Final particles
  particle with PDG code = 11 has total energy 5.20691 MeV,
    3-momentum = (-4.63535 MeV, 1.35707 MeV, -1.87687 MeV),
    mass = 0.510999 MeV, and charge = -1 times the proton charge.
  particle with PDG code = 1000190400 has total energy 37225.7 MeV,
    3-momentum = (4.98066 MeV, -2.27059 MeV, 9.28456 MeV),
    mass = 37225.7 MeV, and charge = 1 times the proton charge.
  particle with PDG code = 22 has total energy 1.53694 MeV,
    3-momentum = (-1.114 MeV, 0.950752 MeV, 0.466119 MeV),
    mass = 0 MeV, and charge = 0 times the proton charge.
  particle with PDG code = 22 has total energy 2.26118 MeV,
    3-momentum = (0.768693 MeV, -0.0372299 MeV, 2.12619 MeV),
    mass = 0 MeV, and charge = 0 times the proton charge.
```

Listing 15: Example output generated by the `marprint` program.

The commands needed to build `marprint` (and a second example program, `mardumpxs`, described in section 7.4.2) against the `MARLEY` shared libraries are given in the `Makefile` located in the folder `examples/executables/build/`. Users are encouraged to adapt this `Makefile` for building their own C++ programs that interface with `MARLEY`.

7.3.4. Example ROOT macros

As a second example of the `MARLEY` analysis API, several *ROOT macros*, i.e., programs intended to be run using the interactive C++ interpreter distributed with ROOT, are provided in the `examples/macros/` folder. Although `RootEventFileReader` and the other `MARLEY` classes are fully compatible with version 6 of ROOT, version 5 lacks support for C++11 features, which are used extensively by `MARLEY`. To work around this problem, `MARLEY` provides a class called `MacroEventFileReader` which is usable within macros written for both versions 5 and 6 of ROOT. This class provides the same interface to `MARLEY` events as `RootEventFileReader`, but the use of C++ syntax that is incompatible with ROOT 5 has been hidden from the interpreter.

In order to correctly interact with `MARLEY` classes, ROOT requires precompiled dictionaries to be loaded at runtime. These are stored in the `MARLEY_ROOT` shared library. To avoid the need for users to manually load the `MARLEY` class dictionaries at the start of each ROOT interpreter session, a startup script called `mroot` is placed in the `build/` folder whenever `MARLEY` is successfully built against ROOT. After sourcing the `setup_marley.sh` setup script (see section 5.3), issuing the command

```
mroot
```

will start the interactive ROOT interpreter and automatically load the `MARLEY` class dictionaries. For version 5 of ROOT, only the `Event`, `Particle`, `Parity`, and `MacroEventFileReader` classes (all defined within the `marley` namespace) may be used within an `mroot` session. For ROOT 6, all `MARLEY` classes will be available if the appropriate header files are loaded via `#include` statements.

The `README.md` file in the `examples/macros/` folder gives brief descriptions of each of the example ROOT macros together with usage instructions. Users are encouraged to adopt these macros as a starting point for implementing their own calculations.

7.3.5. "Flat" ROOT files: the `marsum` utility

Although they are expected to be suitable for many applications, the standard ROOT-format output files generated by `MARLEY` (see section 7.2.4) have two important limitations: (1) They are only fully readable in environments in which both ROOT and `MARLEY` are installed, and (2) Each `Event` object to be analyzed must be loaded from disk in its entirety.

To address these limitations, `MARLEY` provides a command-line utility called `marsum`, which takes as input one or more files containing `MARLEY` events stored in any of the standard output formats. The `marsum` program creates a new file in which many quantities of interest from the input events have been saved as individual branches of a ROOT `TTree`. Following a successful build of `MARLEY` with ROOT support (see section 5), the `marsum` executable will be present in the `build/` folder. Assuming that the `setup_marley.sh` script has already been sourced, one may execute the command

```
marsum myout.root /home/events1.root /home/events2.ascii
```

to create a new file called `myout.root` in the working directory. This file will contain a single ROOT `TTree` named `mst` (for "`MARLEY` summary tree"[31]) with one entry for each event present in the two input files (`/home/events1.root` and `/home/events2.ascii`). The `mst` `TTree` will contain the following branches:

pdgv (*int*) Projectile PDG code
Ev (*double*) Projectile total energy (MeV)
KEv (*double*) Projectile kinetic energy (MeV)
pxv (*double*) Projectile 3-momentum *x*-component (MeV)

[31] This output format was inspired by the similar "gst" `TTree` produced by the `gntpc` utility distributed with GENIE [169, sec. 7.6.2, pp. 112–115].

pyv (*double*) Projectile 3-momentum y-component (MeV)
pzv (*double*) Projectile 3-momentum z-component (MeV)
pdgt (*int*) Target PDG code
Mt (*double*) Target mass (MeV)
pdgl (*int*) Ejectile PDG code
El (*double*) Ejectile total energy (MeV)
KEl (*double*) Ejectile kinetic energy (MeV)
pxl (*double*) Ejectile 3-momentum x-component (MeV)
pyl (*double*) Ejectile 3-momentum y-component (MeV)
pzl (*double*) Ejectile 3-momentum z-component (MeV)
pdgr (*int*) Residue PDG code
Er (*double*) Residue total energy (MeV)
KEr (*double*) Residue kinetic energy (MeV)
pxr (*double*) Residue 3-momentum x-component (MeV)
pyr (*double*) Residue 3-momentum y-component (MeV)
pzr (*double*) Residue 3-momentum z-component (MeV)
Ex (*double*) Initial residue excitation energy (MeV)
twoJ (*int*) Two times the initial residue spin
parity (*int*) Initial residue parity
np (*int*) Number of de-excitation products
pdgp (*int[np]*) De-excitation product PDG codes
Ep (*double[np]*) De-excitation product total energies (MeV)
KEp (*double[np]*) De-excitation product kinetic energies (MeV)
pxp (*double[np]*) De-excitation product 3-momentum x-components (MeV)
pyp (*double[np]*) De-excitation product 3-momentum y-components (MeV)
pzp (*double[np]*) De-excitation product 3-momentum z-components (MeV)
xsec (*double*) Flux-averaged total cross section (10^{-42} cm^2/atom)

The "flat" file created in this way will be readable by ROOT without the need to load the `MARLEY` class dictionaries. Individual branches may also be loaded and plotted (e.g., via the `TTree::Draw` member function) without the need to manipulate the event as a whole.

7.4. Converting event distributions to physics quantities

The theoretical predictions made by an event generator like `MARLEY` may most usefully be compared to competing calculations and experimental data in the form of cross sections and event rates. To see how these quantities may be obtained from a set of simulated events, first note that the expression given in eq. (73) for the probability density $P(E_\nu)$ of the energy E_ν of a reacting neutrino may be rewritten in the form

$$P(E_\nu) = \frac{\phi(E_\nu)\,\sigma(E_\nu)}{\Phi\,\langle\sigma\rangle} \tag{92}$$

where

$$\Phi \equiv \int_{E_\nu^{\min}}^{E_\nu^{\max}} \phi(E_\nu)\,dE_\nu \tag{93}$$

is the total neutrino flux and

$$\langle\sigma\rangle \equiv \frac{1}{\Phi}\int_{E_\nu^{\min}}^{E_\nu^{\max}} \phi(E_\nu)\,\sigma(E_\nu)\,dE_\nu \tag{94}$$

is the (abundance-weighted) flux-averaged total cross section. Let x denote an arbitrary, continuously-distributed observable that is computable from a `MARLEY` event record. Then, for a single event, the probability P_j that x falls within the jth bin $x \in [x_j, x_{j+1})$ is given by

$$P_j = \int_{E_\nu^{\min}}^{E_\nu^{\max}} P(E_\nu) \int_{x_j}^{x_{j+1}} P(x|E_\nu)\,dx\,dE_\nu = \frac{1}{\langle\sigma\rangle}\int_{x_j}^{x_{j+1}} \left\langle\frac{d\sigma}{dx}\right\rangle dx \tag{95}$$

where

$$P(x|E_\nu) = \frac{1}{\sigma(E_\nu)}\frac{d\sigma(E_\nu)}{dx} \tag{96}$$

is the conditional probability density of x at fixed neutrino energy, $d\sigma(E_\nu)/dx$ is the total differential cross section with respect to x, and

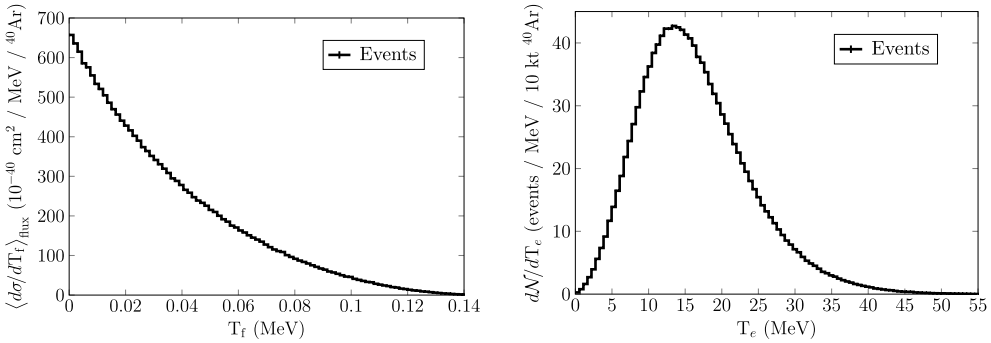

Fig. 3. Example `MARLEY` calculations of physics observables. LEFT: The flux-averaged differential cross section for muon decay-at-rest $\bar{\nu}_\mu$ undergoing coherent elastic neutrino-nucleus scattering on ^{40}Ar. RIGHT: The differential event rate for charged-current absorption of supernova ν_e on 10 kt of pure ^{40}Ar. The Fermi-Dirac neutrino source from Listing 5 was used as a toy model of the time-integrated supernova ν_e spectrum.

$$\left\langle \frac{d\sigma}{dx} \right\rangle \equiv \frac{1}{\Phi} \int_{E_\nu^{\min}}^{E_\nu^{\max}} \phi(E_\nu) \frac{d\sigma(E_\nu)}{dx} dE_\nu \tag{97}$$

is the flux-averaged total differential cross section. A Monte Carlo estimator for the average value of this quantity in the jth bin may be obtained via

$$\left\langle \frac{d\sigma}{dx} \right\rangle_j \equiv \frac{1}{\Delta x_j} \int_{x_j}^{x_{j+1}} \left\langle \frac{d\sigma}{dx} \right\rangle dx \approx \frac{\langle \sigma \rangle f_j}{\Delta x_j}, \tag{98}$$

where $\Delta x_j \equiv x_{j+1} - x_j$ is the bin width and $f_j = n_j/N$ is the ratio of the n_j events that fall within the jth bin to the total number N of simulated events. Recognizing that n_j follows a binomial distribution allows for an estimate of the associated Monte Carlo statistical uncertainty (standard deviation) via

$$\mathrm{SD}\left(\left\langle \frac{d\sigma}{dx} \right\rangle_j \right) \approx \frac{\langle \sigma \rangle}{\Delta x_j N} \sqrt{\frac{(N - n_j) n_j}{N}}. \tag{99}$$

The results from eqs. (98) and (99) may readily be extended to multiple dimensions. For a Monte Carlo calculation of an n-dimensional differential cross section, simply let Δx_j denote the product of the n widths of the jth bin in the n-dimensional phase space.

For a discrete observable, similar expressions may be used with the bin width Δx_j omitted. The flux-averaged partial cross section $\langle \sigma \rangle_j$ to produce events fulfilling a criterion j (e.g., involving emission of a single neutron) is estimated from the simulation results via $\langle \sigma \rangle_j \approx \langle \sigma \rangle f_j$, where $f_j = n_j/N$ is the fraction of simulated events satisfying the criterion. The statistical uncertainty of this estimator is approximated by the expression on the right-hand side of eq. (99) with the substitution $\Delta x_j \to 1$.

7.4.1. Examples

Fig. 3 shows the results of two example calculations of physics observables performed using `MARLEY` events and the procedure outlined in section 7.4. The histograms shown in the left and right panels of Fig. 3 were computed using independent simulations of $N = 2 \times 10^6$ events each. To produce the left-hand plot, coherent elastic neutrino-nucleus scattering (CEvNS) on ^{40}Ar was simulated for $\bar{\nu}_\mu$ produced by μ^+ decays at rest (see section 6.5.4). A histogram describing the distribution of the kinetic energy T_f of the recoiling final-state nucleus was prepared from the simulated events using a uniform bin width $\Delta T_f = 1.5$ keV. The event counts n_j from each bin were renormalized according to eq. (98) to yield a Monte Carlo estimator for the mean value of the flux-averaged differential cross section in the jth bin:

$$\left\langle \frac{d\sigma}{dT_f} \right\rangle_j \approx \frac{\langle \sigma \rangle n_j}{N \Delta T_f}. \tag{100}$$

Here $\langle \sigma \rangle = 26.69 \times 10^{-40} \mathrm{cm}^2 / {}^{40}\mathrm{Ar}$ is the `MARLEY` prediction for the CEvNS flux-averaged total cross section. Equation (100) gives the expression used to obtain the content of each histogram bin shown in the left-hand plot of Fig. 3. Based on eq. (99), an estimate of the statistical uncertainty

$$\mathrm{SD}\left(\left\langle \frac{d\sigma}{dT_f} \right\rangle_j \right) \approx \frac{\langle \sigma \rangle}{N \Delta T_f} \sqrt{\frac{(N - n_j) n_j}{N}} \tag{101}$$

associated with each bin was also calculated but is small on the scale of the plot.

A similar procedure was used to obtain the distribution shown in the right-hand plot of Fig. 3, but the quantity of interest is T_e, the kinetic energy of the electron produced in the charged-current reaction $^{40}\mathrm{Ar}(\nu_e, e^-)^{40}\mathrm{K}^*$. The simulation of this process was carried out using the reaction input file `ve40ArCC_Bhattacharya1998.react` (see section 6.4.1). The incident neutrino spectrum was defined using the Fermi-Dirac source shown in Listing 5, which was treated as a toy model of the time-integrated ν_e flux at Earth produced by a core-collapse supernova. The flux-averaged differential cross section was converted into a differential event rate via

$$\frac{d\mathcal{N}}{dT_e} = \Phi \left\langle \frac{d\sigma}{dT_e} \right\rangle n_{\text{targets}} \tag{102}$$

where a total time-integrated flux[32] of $\Phi = 1.0 \times 10^{11}$ ν_e / cm^2 was assumed. There are $n_{\text{targets}} = 1.5 \times 10^{32}$ atoms in 10 kt of pure ^{40}Ar.

Performance. The two example calculations shown in Fig. 3 were obtained using separate sets of two million events each. The CEvNS simulation needed for the left-hand plot requires only the incident antineutrino energy (see section 4.2.1) and the outgoing antineutrino direction (see section 4.2.4) to be sampled for each event. As a result, program execution is very fast. On the author's MacBook Pro, the full two-million-event `MARLEY` job was completed in 36.6 seconds. Generation of inelastic neutrino-nucleus scattering events is much slower due to the computational demands of the de-excitation loop (see section 4.3). On the same computer as the CEvNS simulation, the two million charged-current events used to create the right-hand plot in Fig. 3 took 125 minutes and 25.7 seconds to produce. Since only the electron kinetic energy is needed for the plot, however, the speed of the simulation can be greatly enhanced by disabling nuclear de-excitations (see section 6.7.2). When this is done, the time required to generate two million events is reduced to 3 minutes and 57.6 seconds. The remaining slowdown relative to the CEvNS case can be attributed to the time needed to sample a nuclear transition (see section 4.2.3) for each event.

7.4.2. Energy-dependent total cross sections: `mardumpxs`

One of the observables predicted by `MARLEY` that allows the most straightforward comparison to other theoretical models is the (abundance-weighted) total cross section as a function of neutrino energy $\sigma(E_\nu)$. While a Monte Carlo estimate of this quantity may be computed from generated events[33] using the approach described in section 7.4, $\sigma(E_\nu)$ is exactly calculable given only the neutrino target composition (see section 4.1.6) and the information stored in the reaction input file(s) (see section 4.1.4). For the convenience of users, an example C++ program called `mardumpxs` (`examples/executables/mardumpxs.cc`) is provided with `MARLEY` that produces tables of $\sigma(E_\nu)$. Assuming that the `setup_marley.sh` script has already been sourced (see section 5.3), one may build `mardumpxs` by executing the commands

```
cd ${MARLEY}/build
make mardumpxs
```

The `mardumpxs` executable takes the name of an output file followed by the name of a `MARLEY` job configuration file (see section 6) as command-line arguments. For example, the command

```
mardumpxs xsec_table.txt config.js
```

will write a table of $\sigma(E_\nu)$ values to the output file `xsec_table.txt` after configuring `MARLEY` according to the settings given in `config.js`. Each line of the output file will have the format

```
KE XSec
```

where `KE` is the projectile kinetic energy[34] in MeV and `XSec` is the abundance-weighted total reaction cross section $\sigma(E_\nu)$ (see section 4.1.6) in 10^{-42} cm^2/atom.

By default, `mardumpxs` assumes a ν_e projectile and produces a table of 10,000 cross section values using a regularly-spaced energy grid between 0 and 100 MeV. This behavior may be altered using `mardumpxs`-specific keys in the job configuration file, as described in section 6.7.8.

8. Interfacing MARLEY with external tools

While it is hoped that `MARLEY`'s capabilities as a standalone software package will be beneficial to the low-energy neutrino physics community, interfacing the generator with external codes has the potential to greatly extend its usefulness. This is particularly true for neutrino detection experiments, which typically rely on end-to-end simulations of neutrino production, neutrino interactions in and around the detector, final-state particle propagation, and the detector electronics response in order to interpret their measurements. For some applications, simply passing `MARLEY` output files as input to the next stage of a multi-step simulation may be satisfactory. In other contexts, direct calls to `MARLEY` functions by a client code may be more appropriate.

Section 8.1 presents the recommended approach to generating `MARLEY` events within an external C++ application, which involves use of the `marley::JSONConfig` and `marley::Generator` classes. An example application of this kind, `marg4`, is discussed in section 8.2. The `marg4` program produces `MARLEY` events and tracks the final-state particles through a simple geometry using the popular Geant4 [101,102] particle transport code. Section 8.3 then briefly discusses the `MARLEY` interface included in the LArSoft [170–173] toolkit.

8.1. C++ event generation API

The core functionality of `MARLEY` is encapsulated for use by external applications in the form of the `marley::Generator` class. While a `Generator` object may be instantiated and configured without reference to a file, doing so is not recommended in most situations. Instead, users are encouraged to construct `Generator` objects indirectly by means of the `create_generator` member function of the `marley::JSONConfig` class. The constructor of `JSONConfig` takes a single string argument containing the name of a

[32] The quoted value is a rough estimate for a core-collapse supernova at 10 kpc from Earth.

[33] Specifically, this involves generating events using a uniform neutrino energy spectrum $\phi(E_\nu)$ and then constructing a histogram of the neutrino energy distribution.

[34] Since `MARLEY` treats neutrinos as massless, this is the same as the total energy E_ν. Kinetic energy is used by `mardumpxs` in anticipation of extensions to `MARLEY` involving massive projectiles.

```
1  // Standard library includes
2  #include <iostream>
3
4  // MARLEY includes
5  #include "marley/Event.hh"
6  #include "marley/Generator.hh"
7  #include "marley/JSONConfig.hh"
8
9  constexpr int NUM_EVENTS = 10;
10
11 int main() {
12
13   marley::JSONConfig cfg( "/home/config.js" );
14   marley::Generator gen = cfg.create_generator();
15
16   for ( int j = 0; j < NUM_EVENTS; ++j ) {
17     marley::Event ev = gen.create_event();
18     std::cout << ev << '\n';
19   }
20
21 }
```

Listing 16: Minimal working example of a C++ program that generates MARLEY events.

MARLEY job configuration file. The contents of this file are parsed and used to initialize a Generator object during a subsequent call to JSONConfig::create_generator. With the exception of the parameters described in section 6.6 (which are unique to the marley command-line executable), all configuration options listed in section 6 will be recognized and respected by the JSONConfig class. Once a fully-initialized Generator object has been created, a single neutrino scattering event may be simulated and returned as a marley::Event object by calling the create_event member function. A new event will be generated with each call to this function.

Listing 16 shows the source code for examples/executables/minimal/evgen.cc, an example C++ program that uses the recommended event generation API. On line 13, a JSONConfig object called cfg is constructed using settings from the job configuration file /home/config.js. The settings parsed by cfg are then used to create a Generator object called gen on line 14. A simple event loop is defined on lines 16–19 and iterates until ten events have been generated (line 17) and printed in ASCII format to standard output (line 18). The evgen.cc example program may most easily be compiled by using the marley-config script as described in section 5.5.1.

In cases where handling of ROOT-dependent configuration file parameters (see, e.g., section 6.5.7) is desired, the RootJSONConfig class should be used instead of JSONConfig . This amounts to making the replacement JSONConfig → RootJSONConfig on lines 7 and 13 of Listing 16. Assuming that MARLEY has been built with ROOT support, the compilation command given in section 5.5.1 will automatically link to the required ROOT libraries.

Because the main marley executable mentioned in section 5.4 makes use of the event generation API described above, the performance of MARLEY is comparable when it is linked to an external code and when running independently. Experimental collaborations which have included MARLEY in a full simulation chain (including DUNE [46] and MicroBooNE [174]) have typically found that simulation of the detector response contributes much more to the total runtime than generation of the neutrino interaction events themselves.

8.2. Interfacing MARLEY with Geant4

The C++-based Geant4 software framework provides a large suite of tools for simulating particle propagation through matter. In the main function of a typical Geant4 application, a G4RunManager object is constructed and used to drive the simulation. Before the simulation can begin, the G4RunManager must be initialized with pointers to three objects, each of which is derived from a distinct abstract base class defined by Geant4. The three required classes used to initialize the run manager are

G4VUserDetectorConstruction Defines the geometry and material composition of the *world volume* (and zero or more subvolumes) through which the simulated particles will be tracked

G4VUserPhysicsList Defines the particle species and physics processes to be included in the simulation

G4VUserPrimaryGeneratorAction Defines a member function, GeneratePrimaries, which will be used to populate each new event with the starting locations, momenta, etc. of all primary particles to be tracked

In addition to these required classes, pointers to objects that instantiate optional *user action* classes may also be registered with the run manager. These allow user-defined functions to be executed at various stages of the simulation. Further details about general Geant4 application development are available in Ref. [175].

8.2.1. The marg4 Geant4 application

The folder examples/marg4/ contains the source code for marg4, an example Geant4 application that uses MARLEY events as a source of primary particles. The world volume is defined by the DetectorConstruction class and consists of a single uniform sphere of liquid argon with a radius of 10 m and centered on the origin. A built-in Geant4 physics list suitable for MeV-scale particle transport, QGSP_BIC_HP, is constructed using a factory method in examples/marg4/src/marg4.cc. As a trivial example of a user action, an EventAction object is used to print the current event count to standard output at the beginning of every hundredth event.

```
1  void MarleyPrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
2  {
3    // Create a new primary vertex at the spacetime origin.
4    G4PrimaryVertex* vertex = new G4PrimaryVertex(0., 0., 0., 0.); // x,y,z,t0
5
6    // Generate a new MARLEY event using the owned marley::Generator object
7    marley::Event ev = marley_generator_.create_event();
8
9    // This line, if uncommented, will print the event in ASCII format
10   // to standard output
11   //std::cout << ev << '\n';
12
13   // Loop over each of the final particles in the MARLEY event
14   for ( const auto& fp : ev.get_final_particles() ) {
15
16     // Convert each one from a marley::Particle into a G4PrimaryParticle.
17     // Do this by first setting the PDG code and the 4-momentum components.
18     G4PrimaryParticle* particle = new G4PrimaryParticle( fp->pdg_code(),
19       fp->px(), fp->py(), fp->pz(), fp->total_energy() );
20
21
22     // Also set the charge of the G4PrimaryParticle appropriately
23     particle->SetCharge( fp->charge() );
24
25
26     // Add the fully-initialized G4PrimaryParticle to the primary vertex
27     vertex->SetPrimary( particle );
28   }
29
30   // The primary vertex has been fully populated with all final-state particles
31   // from the MARLEY event. Add it to the G4Event object so that Geant4 can
32   // begin tracking the particles through the simulated geometry.
33   anEvent->AddPrimaryVertex( vertex );
34 }
```

Listing 17: Definition of the `GeneratePrimaries` member function of the `MarleyPrimaryGeneratorAction` class. This listing is an excerpt from the file `examples/marg4/src/MarleyPrimaryGeneratorAction.cc` included with `MARLEY`.

An example of a direct interface between `MARLEY` and Geant4 is provided by the `MarleyPrimaryGeneratorAction` class, which is derived from `G4VUserPrimaryGeneratorAction`. The constructor of this class takes a single `std::string` argument containing the name of a `MARLEY` job configuration file. This file name is used to create either a `JSONConfig` or a `RootJSONConfig` object, with the latter being chosen if the `USE_ROOT` preprocessor macro is defined (see section 5.5). The member variable `marley_generator_`, which is a `Generator` object, is then initialized using the parameters from the configuration file.

Listing 17 shows the `GeneratePrimaries` member function defined by the `MarleyPrimaryGeneratorAction` class. This function is called once during initialization of each Geant4 event. On line 4, a new `G4PrimaryVertex` object is created at the spacetime origin. All primary particles which are associated with it will begin their Geant4 trajectories at the same 4-position. After a single `marley::Event` object is created on line 7, each of its final-state particles is converted into a new `G4PrimaryParticle` by the loop defined on lines 14–28. Line 27 associates each fully-initialized `G4PrimaryParticle` with the primary vertex defined previously, and line 33 adds the completed primary vertex to the current `G4Event` object. Propagation of the primary particles obtained from the `MARLEY` event is simulated by Geant4 after the `GeneratePrimaries` function returns.

8.2.2. *Building and running `marg4`*

Building the example `marg4` program requires the `geant4-config` utility (included as part of a standard Geant4 installation) to be present on the system's executable search path. Additionally, use of the `QGSP_BIC_HP` physics list mentioned above requires installation of the data files belonging to the Geant4 Neutron Data Library (G4NDL) [176,177]. These files, which are available for download from the Geant4 website (https://geant4.web.cern.ch/support/download), are required for high-precision (HP) tracking of low-energy neutrons by Geant4. The use of Geant4's HP treatment of neutron transport is strongly recommended for propagation of neutrino-induced neutrons from `MARLEY` events.

Assuming that the `setup_marley.sh` script has already been sourced (see section 5.3), the `marg4` executable may be built against Geant4 via the commands

```
cd ${MARLEY}/build
make marg4
```

The `marg4` executable takes the number of desired events followed by the name of a `MARLEY` job configuration file as its command-line arguments. For example, invoking the program via

```
marg4 500 /home/myconfig.js
```

will simulate 500 Geant4 events. Each of these events will contain a single primary vertex populated with the final-state particles from one `MARLEY` event. The `MARLEY` events will be generated using the job configuration given in the file `/home/myconfig.js`.

The marg4 program is provided as a simple usage example for the API described in section 8.1. As such, it does not produce any output other than logging messages from both MARLEY and Geant4. Interested users are encouraged to copy and modify the marg4 source code to meet their needs. Information from the marley::Event objects themselves may be accessed using the member functions described in section 7.3.2. The functions needed to extract quantities of interest from the Geant4 simulation are documented in Ref. [175].

8.3. LArSoft interface

The liquid argon software toolkit (LArSoft) provides a flexible simulation, reconstruction, and analysis framework designed for liquid argon time projection chamber (LArTPC) experiments. Based on the art event-processing framework [178], LArSoft has been adopted as a key part of the software infrastructure for various experimental collaborations, including ArgoNeuT [179], LArIAT [180], MicroBooNE [181], ICARUS [182], SBND [183], and DUNE [184].

The LArSoft source code is hosted on GitHub and split into multiple respositories, each of which provides a particular type of functionality. The larsim respository (https://github.com/LArSoft/larsim) includes, among other things, interfaces to external physics event generators for neutrino interactions (e.g., GENIE) and for other processes (e.g., the cosmic-ray generators CORSIKA [185] and CRY [186]).

Version 6.04.00 of LArSoft was the first[35] to include a direct larsim interface to MARLEY contributed by the present author.[36] Ever since the initial version of the interface was added, MARLEY has been included as part of the standard LArSoft distribution. A detailed description of the LArSoft interface to MARLEY, which is more sophisticated than the examples given in section 8.1 and section 8.2, is beyond the present scope. However, brief descriptions of the relevant C++ classes defined in the larsim source code (see the larsim/EventGenerator/MARLEY/ subfolder of the larsim repository) are given below. All four of these classes are defined within the evgen namespace used by LArSoft for event generation.

ActiveVolumeVertexSampler Used to sample neutrino vertex positions uniformly over the active volume(s) of a detector. The approach used by this class is only suitable for simulations in which (1) the detector is uniformly illuminated by the incident neutrino flux, and (2) neutrino interactions occurring outside the detector active volume(s) are not of interest.

MarleyGen Creates neutrino scattering events in the native LArSoft format using the ActiveVolumeVertexSampler and MARLEY Helper classes

MARLEYHelper Implements the low-level interface between MARLEY and LArSoft. At the beginning of a simulation job, this class initializes a marley::Generator object by converting a LArSoft configuration given in the Fermilab Hierarchical Configuration Language (FHiCL) [188] to the JSON-like format (see section 6) used by MARLEY. With the exception of the "executable settings" described in section 6.6, all MARLEY job configuration file parameters are available for use via FHiCL.

This class also provides a member function (create_MCTruth) which generates a MARLEY event and stores a representation of it in an instance of the simb::MCTruth class. The simb::MCTruth class is used by LArSoft as a generator-agnostic event record.

MarleyTimeGen Similar to MarleyGen, but provides experimental support for generating events using a time-dependent neutrino spectrum

Further technical details about these and other LArSoft classes are available in the LArSoft Doxygen documentation (https://nusoft.fnal.gov/larsoft/doxsvn/html/index.html).

9. Prospects for future development

This work describes a new Monte Carlo event generator, MARLEY, suitable for simulating tens-of-MeV neutrino scattering on complex nuclei. The release of MARLEY 1.2.0, the first version of the code to be documented with the present level of detail, represents a significant milestone. However, active, open-source development of MARLEY is ongoing, and contributions from the community are encouraged. To coordinate development efforts, those interested in contributing improvements to MARLEY are asked to contact the author at the earliest opportunity.

Potential avenues for future development of MARLEY include the following:

- Preparation of additional reaction input files. In addition to new scattering modes (inelastic NC and $\bar{\nu}_e$ CC) for ^{40}Ar, the addition of input files for several other nuclides will likely be of immediate interest. These include but are not limited to
  1. ^{12}C and ^{16}O, both of which make small but non-negligible contributions to the total event rate in hydrocarbon (e.g., NOvA [189]) and water Cherenkov (e.g., Super-Kamiokande [190]) supernova neutrino detectors
  2. The stable lead isotopes (especially ^{208}Pb), which serve as the target material for the HALO [48,49] detector
  3. The stable iron and copper isotopes (especially ^{56}Fe and ^{63}Cu), which, together with lead, are under study by the COHERENT [36] experiment in an effort to understand low-energy neutrino-induced neutron production.
- Inclusion of forbidden transitions in the MARLEY neutrino-nucleus cross section model. These are currently neglected via the allowed approximation. Recent theoretical calculations using a Continuum Random Phase Approximation (CRPA) approach [191,192] indicate that the forbidden contributions can have a significant impact on both the total cross section and the kinematic distributions of the outgoing lepton.
- Addition of new keys to the job configuration file format to allow variations of model parameters, e.g., those used by the nuclear optical model. With some related enhancements to other parts of the code, e.g., storage of the full de-excitation history in the MARLEY event record, these variations could be used to assess theoretical uncertainties on MARLEY predictions via event reweighting.

[35] Ref. [187] incorrectly identifies the initial LArSoft version as 6.03.00. This number refers instead to the corresponding version of the larsim subpackage.

[36] The first MARLEY version to be included in LArSoft was 0.9.5, an August 2016 public beta release.

An approach of this kind is commonly used by accelerator neutrino experiments. Documentation of the GENIE event reweighting framework, for example, is given in Ref. [169, ch. 9, pp. 129–154].

- Improvements to the MARLEY treatment of nuclear de-excitations. These could include
  - Realistic angular distributions for the de-excitation products (instead of the isotropic emission that is currently assumed)
  - Storage of particle emission times in the event record. Related updates could also be made to the nuclear structure data format to allow measured discrete level lifetimes to be listed.
  - Competition of internal conversion with γ-ray emission
  - Pre-equilibrium emission of nuclear fragments
  - Modeling of neutrino-induced fission
- Creation of new subclasses of `marley::Reaction` to simulate events for projectiles other than neutrinos. Processes likely to be of interest include electron-nucleus scattering and beyond the Standard Model reactions like nuclear absorption of MeV-scale dark matter [193].
- Implementation of a tool to facilitate comparisons of MARLEY model predictions with cross section measurements from low-energy neutrino experiments. This could potentially be accomplished by adding low-energy datasets and a MARLEY interface to the NUISANCE [194] framework used by the accelerator neutrino community.
- Simulation of events within a detector geometry in which both the target material and the incident neutrino flux are spatially non-uniform. This may most easily be achieved by interfacing MARLEY with flux and geometry navigation drivers present in an existing generator (see Ref. [169, sec. 6] for a description of GENIE's). A "community based" version of these tools, independent of any particular generator, has also been proposed [195].

Acknowledgements

I am grateful to Myung-Ki Cheoun for providing the QRPA Gamow-Teller matrix elements which are used to compute $^{40}\mathrm{Ar}(\nu_e, e^-)^{40}\mathrm{K}^*$ cross sections at high excitation energies (see section 6.4.1 and Ref. [107]). I also thank the authors of the TALYS code for sharing their nuclear level data files under the GNU General Public License.

Vishvas Pandey provided helpful feedback on a draft of this paper, and I thank Robert Svoboda, Ramona Vogt, and Michael Mulhearn for their comments on the PhD thesis [187] which served as the first formal description of MARLEY. I am also indebted to Sam Hedges and Erin Conley for their tests of the MARLEY 1.2.0 release candidate.

My work to develop MARLEY while at the University of California, Davis was supported in part by the John Jungerman-Charles Soderquist Graduate Fellowship and by the DOE National Nuclear Security Administration through the Nuclear Science and Security Consortium under award number DE-NA0003180.

This manuscript has been authored by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Nuclear structure data file format

The MARLEY nuclear structure data files mentioned in section 4.1.5 contain tables of nuclear energy levels and γ-ray branching ratios for one or more nuclides. Each table follows a whitespace-delimited format that begins with the header

```
Z A num_levels
```

in which `Z` is the proton number, `A` is the mass number, and `num_levels` is the number of tabulated nuclear levels. The header is followed by data blocks for each level in order of increasing excitation energy. Each block begins with the line

```
Ex twoJ Pi num_gammas
```

where `Ex` is the level excitation energy (MeV), `twoJ` is two times the level spin (the factor of two allows half-integer spins to be represented by an integer), and `Pi` is the parity (denoted by the character `+` or `-`). The data block for a level terminates with a set of lines describing each available γ-ray transition (for a total of `num_gammas` transitions). Each of these lines has the format

```
E_gamma RI Lf_index
```

where `E_gamma` is the energy of the emitted γ-ray (MeV) and `RI` is the relative intensity of the transition. Although the relative intensities in the official MARLEY structure data files are normalized to sum to unity, this is not required. The parameter `Lf_index` gives the position of the final nuclear level accessed by the transition, with `Lf_index` $= 0$ corresponding to the first listed level (presumably the ground state).

Listing 18 shows the table for ^{43}Cl which appears in the structure data file `data/structure/Cl.dat` included with MARLEY. Five nuclear levels appear in the table, each (apart from the ground state) having one listed γ-ray transition. Although the lines giving the nuclide header, level descriptions, and γ-ray descriptions are shown here with different indentations to improve readability, this is not required by the file format.

```
17 43 5
  0 1 + 0
  0.329 3 + 1
    0.329 1 0
  0.944 5 + 1
    0.615 1 1
  1.34 5 + 1
    1.34 1 0
  1.829 7 + 1
    0.885 1 2
```

Listing 18: Example MARLEY nuclear structure data file.

Table 6
Reaction mode labeling scheme used in MARLEY 1.2.0. The integer labels used in reaction input files are represented in the source code by the marley::Reaction::ProcessType enumerated type.

Reaction mode	ProcessType integer	ProcessType enum	Isospin operator
$A(\nu, \ell^-)$	0	NeutrinoCC	t_-
$A(\bar{\nu}, \ell^+)$	1	AntiNeutrinoCC	t_+
$A(\nu, \nu)$ or $A(\bar{\nu}, \bar{\nu})$	2	NC	t_3 or 1
$e^-(\nu, \nu)$ or $e^-(\bar{\nu}, \bar{\nu})$	3	NuElectronElastic	none

```
3 1000180360
1000180380
1000180400
```

Listing 19: Example MARLEY reaction input file for neutrino-electron elastic scattering.

Appendix B. Reaction input file format

The reaction input files mentioned in section 6.4 provide information needed for MARLEY to configure cross section calculations according to eq. (5) (for neutrino-nucleus scattering) or eq. (55) (for neutrino-electron elastic scattering). Any line of a reaction input file that begins with a # character will be treated as a comment and ignored by the parser. The first line of the file that is not a comment contains a header of the form

```
ProcessType NucPDG
```

where ProcessType is an integer code representing the reaction mode and NucPDG is a nuclear PDG code (see section 7.1) identifying the target nuclide involved in the initial state. Table 6 lists the allowed values of ProcessType in the second column together with their corresponding reaction modes and isospin operators. In the first column, the symbol A is used as a stand-in for any target nucleus. The third column lists the elements of the enumerated type used in the source code to represent each reaction mode.

For reaction input files describing neutrino-electron elastic scattering (ProcessType = 3), only the header shown above is required to be present. If the user wishes to enable simulation of this process for multiple atomic targets, additional values of NucPDG may be included on subsequent lines. Listing 19 shows an example reaction input file which instructs MARLEY to simulate neutrino-electron elastic scattering for the three stable isotopes of argon: ^{36}Ar, ^{38}Ar, and ^{40}Ar.

For neutrino-nucleus reaction modes, each line of the input file following the header is used to specify the value of a reduced matrix element describing a transition to a particular final nuclear level. These lines are whitespace-delimited and have the format

```
Ex B type
```

where Ex is the excitation energy of the final level (MeV, measured with respect to the ground state of the residual nucleus), B is the corresponding value of either B(F) or B(GT), and type is an integer code that represents whether B should be interpreted as a value of B(F) (type = 0) or of B(GT) (type = 1). The matrix elements must be listed in order of increasing Ex in the reaction input file. If this is not the case, then MARLEY will halt with the error message

```
[ERROR]: Invalid reaction dataset. Level energies must be unique and must be given in ascending order.
```

when the file is used.

In most cases, the matrix element values given in the reaction input file should be computed using the full expressions shown in eq. (6) and eq. (7). This includes, e.g., any assumed value of the axial-vector coupling constant g_A. The sole exception is the Fermi matrix element for neutral-current scattering, which should be listed with a factor of $Q_W^2/4$ removed. For an NC transition to the ground state of a spin-zero nucleus, $B(F) = g_V^2\, Q_W^2/4$, but $\mathrm{B} = g_V^2 = 1$.

Listing 20 shows an excerpt from the reaction input file data/react/ve40ArCC_Bhattacharya1998.react, which tabulates nuclear matrix elements for CC neutrino-argon scattering. Two B(GT) values (0.90 and 1.50) are given, both for final states between 2 and 3 MeV above the ^{40}K ground state.

```
1  0 1000180400
2  2.289868 0.90 1
3  2.730357 1.50 1
```

Listing 20: Example `MARLEY` reaction matrix element data file.

References

[1] M. Galassi, B. Gough, GNU Scientific Library: Reference Manual, Network Theory, 2009.
[2] GSL - GNU Scientific Library, https://www.gnu.org/software/gsl/, 2019. (Accessed 27 January 2021).
[3] R. Brun, F. Rademakers, Nucl. Instrum. Methods Phys. Res., Sect. A 389 (1997) 81–86, https://doi.org/10.1016/S0168-9002(97)00048-X.
[4] ROOT Data Analysis Framework, https://root.cern.ch, 2021. (Accessed 27 January 2021).
[5] T. Sjöstrand, S. Ask, J.R. Christiansen, R. Corke, N. Desai, P. Ilten, S. Mrenna, S. Prestel, C.O. Rasmussen, P.Z. Skands, Comput. Phys. Commun. 191 (2015) 159–177, https://doi.org/10.1016/j.cpc.2015.01.024, arXiv:1410.3012.
[6] J. Bellm, S. Gieseke, D. Grellscheid, S. Plätzer, M. Rauch, C. Reuschle, P. Richardson, P. Schichtel, M.H. Seymour, A. Siódmok, A. Wilcock, et al., Eur. Phys. J. C 76 (2016) 196, https://doi.org/10.1140/epjc/s10052-016-4018-8, arXiv:1512.01178.
[7] I. Lokhtin, L. Malinina, S. Petrushanko, A. Snigirev, I. Arsene, K. Tywoniuk, Comput. Phys. Commun. 180 (2009) 779–799, https://doi.org/10.1016/j.cpc.2008.11.015, arXiv:0809.2708.
[8] J. Verbeke, J. Randrup, R. Vogt, Comput. Phys. Commun. 191 (2015) 178–202, https://doi.org/10.1016/j.cpc.2015.02.002.
[9] O.A. Ponkratenko, V.I. Tretyak, Y.G. Zdesenko, Phys. At. Nucl. 63 (2000) 1282–1287, https://doi.org/10.1134/1.855784, arXiv:nucl-ex/0104018.
[10] S. Nagamiya, Prog. Theor. Exp. Phys. 2012 (2012) 02B001, https://doi.org/10.1093/ptep/pts025.
[11] A. Fava, J.L. Raaf, P. Shanahan, L. Suter, Z. Pavlovic, J. Zennamo, R. Zwaska, Status of Fermilab's Neutrino Facilities, https://indico.cern.ch/event/765096/contributions/3296027/, 2018, input submitted to the European Particle Physics Strategy Update 2018–2020.
[12] C. Andreopoulos, et al., Nucl. Instrum. Methods A 614 (2010) 87–104, https://doi.org/10.1016/j.nima.2009.12.009, arXiv:0905.2517.
[13] O. Buss, T. Gaitanos, K. Gallmeister, et al., Phys. Rep. 512 (2012) 1–124, https://doi.org/10.1016/j.physrep.2011.12.001.
[14] Y. Hayato, Acta Phys. Pol. B 40 (2009) 2477–2489, http://www.actaphys.uj.edu.pl/fulltext?series=Reg&vol=40&page=2477.
[15] T. Golan, J. Sobczyk, J. Żmuda, Nucl. Phys. B, Proc. Suppl. 229–232 (2012) 499, https://doi.org/10.1016/j.nuclphysbps.2012.09.136.
[16] T. Katori, AIP Conf. Proc. 1663 (2015) 030001, https://doi.org/10.1063/1.4919465, arXiv:1304.6014.
[17] A. Bodek, J.L. Ritchie, Phys. Rev. D 23 (1981) 1070–1091, https://doi.org/10.1103/PhysRevD.23.1070.
[18] O. Benhar, A. Fabrocini, S. Fantoni, Nucl. Phys. A 505 (1989) 267–299, https://doi.org/10.1016/0375-9474(89)90374-6.
[19] O. Benhar, A. Fabrocini, S. Fantoni, I. Sick, Nucl. Phys. A 579 (1994) 493–517, https://doi.org/10.1016/0375-9474(94)90920-2.
[20] G.B. King, K. Mahn, L. Pickering, N. Rocco, Phys. Rev. C 101 (2020) 065502, https://doi.org/10.1103/PhysRevC.101.065502, arXiv:2002.02626.
[21] S. Dytman, Acta Phys. Pol. B 40 (2009) 2445–2460, https://www.actaphys.uj.edu.pl/R/40/9/2445/pdf.
[22] W.Y. Ma, E.S.P. Guerra, M. Yu, A. Fiorentini, T. Feusels, J. Phys. Conf. Ser. 888 (2017) 012171, https://doi.org/10.1088/1742-6596/888/1/012171.
[23] T. Golan, C. Juszczak, J.T. Sobczyk, Phys. Rev. C 86 (2012) 015505, https://doi.org/10.1103/PhysRevC.86.015505, arXiv:1202.4197.
[24] S. Dytman, Y. Hayato, R. Raboanary, J. Sobczyk, J.T. Vidal, N. Vololoniaina, Comparison of validation methods of simulations for final state interactions in hadron production experiments, arXiv:2103.07535, 2021.
[25] https://gibuu.hepforge.org/trac/wiki/Paper, 2020. (Accessed 27 January 2021).
[26] U. Mosel, J. Phys. G, Nucl. Part. Phys. 46 (2019) 113001, https://doi.org/10.1088/1361-6471/ab3830, arXiv:1904.11506.
[27] A. Aurisano, C. Backhouse, R. Hatcher, N. Mayer, J. Musser, R. Patterson, R. Schroeter, A. Sousa, J. Phys. Conf. Ser. 664 (2015) 072002, https://doi.org/10.1088/1742-6596/664/7/072002.
[28] G.G. Raffelt, Nucl. Phys. B, Proc. Suppl. 221 (2011) 218–229, https://doi.org/10.1016/j.nuclphysbps.2011.09.006, arXiv:astro-ph/0701677.
[29] K. Scholberg, Annu. Rev. Nucl. Part. Sci. 62 (2012) 81–103, https://doi.org/10.1146/annurev-nucl-102711-095006, arXiv:1205.6003.
[30] S. Horiuchi, J.P. Kneller, J. Phys. G, Nucl. Part. Phys. 45 (2018) 043002, https://doi.org/10.1088/1361-6471/aaa90a, arXiv:1709.01515.
[31] F. Capozzi, S.W. Li, G. Zhu, J.F. Beacom, Phys. Rev. Lett. 123 (2019) 131803, https://doi.org/10.1103/PhysRevLett.123.131803, arXiv:1808.08232.
[32] P. Bakhti, M. Rajaee, Phys. Rev. D 102 (2020) 035024, https://doi.org/10.1103/PhysRevD.102.035024, arXiv:2003.12984.
[33] R. Harnik, K.J. Kelly, P.A.N. Machado, Phys. Rev. D 101 (2020) 033008, https://doi.org/10.1103/PhysRevD.101.033008, arXiv:1911.05088.
[34] F. Suekane, Hunt for sterile neutrinos: decay at rest experiments, arXiv:1604.06190, 2016.
[35] C. Rott, J. Phys. Conf. Ser. 1468 (2020) 012176, https://doi.org/10.1088/1742-6596/1468/1/012176.
[36] D. Akimov, J.B. Albert, et al., COHERENT Collaboration, COHERENT 2018 at the spallation neutron source, arXiv:1803.09183, 2018.
[37] C. Grant, B. Littlejohn, Opportunities with decay-at-rest neutrinos from decay-in-flight neutrino beams, arXiv:1510.08431, 2015.
[38] J.F. Beacom, W.M. Farr, P. Vogel, Phys. Rev. D 66 (2002) 033001, https://doi.org/10.1103/PhysRevD.66.033001, arXiv:hep-ph/0205220.
[39] D.Z. Freedman, Phys. Rev. D 9 (1974) 1389–1392, https://doi.org/10.1103/PhysRevD.9.1389.
[40] D. Akimov, J.B. Albert, P. An, et al., COHERENT Collaboration, Science 357 (2017) 1123–1126, https://doi.org/10.1126/science.aao0990, arXiv:1708.01294.
[41] D. Akimov, J.B. Albert, P. An, C. Awe, P.S. Barbeau, B. Becker, V. Belov, I. Bernardi, M.A. Blackston, L. Blokland, et al., COHERENT Collaboration, Phys. Rev. Lett. 126 (2021) 012002, https://doi.org/10.1103/PhysRevLett.126.012002, arXiv:2003.10630.
[42] A.M. Ankowski, Improved estimate of the cross section for inverse beta decay, arXiv:1601.06169, 2016.
[43] A. Strumia, F. Vissani, Phys. Lett. B 564 (2003) 42–54, https://doi.org/10.1016/S0370-2693(03)00616-6, arXiv:astro-ph/0302055.
[44] L. Alvarez-Ruso, C. Andreopoulos, et al., GENIE Collaboration, Recent Highlights from GENIE v3, arXiv:2106.09381, 2021.
[45] B. Abi, R. Acciarri, et al., DUNE Collaboration, Deep Underground Neutrino Experiment (DUNE), Far Detector Technical Design Report, Volume II: DUNE Physics, arXiv:2002.03005, 2020.
[46] B. Abi, et al., DUNE Collaboration, Eur. Phys. J. C 81 (2021) 423, https://doi.org/10.1140/epjc/s10052-021-09166-w, arXiv:2008.06647.
[47] A. Ankowski, J. Beacom, O. Benhar, S. Chen, et al., Supernova physics at DUNE, arXiv:1608.07853, 2016.
[48] C.A. Duba, F. Duncan, J. Farine, et al., J. Phys. Conf. Ser. 136 (2008) 042077, https://doi.org/10.1088/1742-6596/136/4/042077.
[49] K. Zuber, Nucl. Part. Phys. Proc. 265–266 (2015) 233–235, https://doi.org/10.1016/j.nuclphysbps.2015.06.059.
[50] Helium and Lead Observatory: Astronomically patient, https://www.snolab.ca/halo, 2012. (Accessed 27 January 2021).
[51] R. Davis, D.S. Harmer, K.C. Hoffman, Phys. Rev. Lett. 20 (1968) 1205–1209, https://doi.org/10.1103/PhysRevLett.20.1205.
[52] T. Suzuki, A.B. Balantekin, T. Kajino, S. Chiba, J. Phys. G, Nucl. Part. Phys. 46 (2019) 075103, https://doi.org/10.1088/1361-6471/ab1c11, arXiv:1904.11291.
[53] A.R. Samana, F. Krmpotić, N. Paar, C.A. Bertulani, Phys. Rev. C 83 (2011) 024303, https://doi.org/10.1103/PhysRevC.83.024303, arXiv:1005.2134.
[54] T. Suzuki, S. Chiba, T. Yoshida, T. Kajino, T. Otsuka, Phys. Rev. C 74 (2006) 034307, https://doi.org/10.1103/PhysRevC.74.034307, arXiv:nucl-th/0608056.
[55] A.C. Hayes, I.S. Towner, Phys. Rev. C 61 (2000) 044603, https://doi.org/10.1103/PhysRevC.61.044603, arXiv:nucl-th/9907049.
[56] C. Volpe, N. Auerbach, G. Colò, T. Suzuki, N. Van Giai, Phys. Rev. C 62 (2000) 015501, https://doi.org/10.1103/PhysRevC.62.015501, arXiv:nucl-th/0001050.
[57] E. Kolbe, K. Langanke, P. Vogel, Nucl. Phys. A 652 (1999) 91–100, https://doi.org/10.1016/S0375-9474(99)00152-9, arXiv:nucl-th/9903022.
[58] M.S. Reen, M. Sakuda, T. Sudo, A. Tamii, K. Nakazato, T. Suzuki, H. Suzuki, JPS Conf. Proc. 31 (2020) 011014, https://doi.org/10.7566/JPSCP.31.011014.
[59] K. Nakazato, T. Suzuki, M. Sakuda, Prog. Theor. Exp. Phys. 2018 (2018) 123E02, https://doi.org/10.1093/ptep/pty134, arXiv:1809.08398.
[60] T. Suzuki, S. Chiba, T. Yoshida, K. Takahashi, H. Umeda, Phys. Rev. C 98 (2018) 034613, https://doi.org/10.1103/PhysRevC.98.034613, arXiv:1807.02367.
[61] K. Langanke, P. Vogel, E. Kolbe, Phys. Rev. Lett. 76 (1996) 2629, https://doi.org/10.1103/physrevlett.76.2629, arXiv:nucl-th/9511032.
[62] E. Kolbe, K. Langanke, S. Krewald, F.-K. Thielemann, Phys. Rep. 227 (1993) 37–46, https://doi.org/10.1016/0370-1573(93)90055-I.
[63] E. Ydrefors, J. Suhonen, Phys. Rev. C 87 (2013) 034314, https://doi.org/10.1103/PhysRevC.87.034314.

[64] E. Ydrefors, J. Suhonen, Adv. High Energy Phys. 2012 (2012) 373946, https://doi.org/10.1155/2012/373946.
[65] E. Ydrefors, K. Balasi, T. Kosmas, J. Suhonen, Nucl. Phys. A 896 (2012) 1–23, https://doi.org/10.1016/j.nuclphysa.2012.10.001.
[66] K. Balasi, E. Ydrefors, T. Kosmas, Nucl. Phys. A 868–869 (2011) 82–98, https://doi.org/10.1016/j.nuclphysa.2011.08.003.
[67] H. Ejiri, J. Engel, N. Kudomi, Phys. Lett. B 530 (2002) 27–32, https://doi.org/10.1016/S0370-2693(02)01349-7, arXiv:astro-ph/0112379.
[68] H. Ejiri, J. Engel, R. Hazama, P. Krastev, N. Kudomi, R.G.H. Robertson, Phys. Rev. Lett. 85 (2000) 2917–2920, https://doi.org/10.1103/PhysRevLett.85.2917, arXiv:nucl-ex/9911008.
[69] S. Haselschwardt, B. Lenardo, P. Pirinen, J. Suhonen, Phys. Rev. D 102 (2020) 072009, https://doi.org/10.1103/PhysRevD.102.072009, arXiv:2009.00535.
[70] P. Pirinen, J. Suhonen, E. Ydrefors, Phys. Rev. C 99 (2019) 014320, https://doi.org/10.1103/PhysRevC.99.014320.
[71] P.C. Divari, Adv. High Energy Phys. 2013 (2013) 143184, https://doi.org/10.1155/2013/143184.
[72] N. Shul'gina, B. Danilin, Nucl. Phys. A 554 (1993) 137–157, https://doi.org/10.1016/0375-9474(93)90362-2.
[73] R.S. Raghavan, Phys. Rev. Lett. 78 (1997) 3618–3621, https://doi.org/10.1103/PhysRevLett.78.3618.
[74] W. Almosly, B.G. Carlsson, J. Dobaczewski, J. Suhonen, J. Toivanen, P. Vesely, E. Ydrefors, Phys. Rev. C 89 (2014) 024308, https://doi.org/10.1103/PhysRevC.89.024308.
[75] W. Almosly, E. Ydrefors, J. Suhonen, J. Phys. G, Nucl. Part. Phys. 42 (2014) 025106, https://doi.org/10.1088/0954-3899/42/2/025106.
[76] H. Ejiri, S.R. Elliott, Phys. Rev. C 89 (2014) 055501, https://doi.org/10.1103/PhysRevC.89.055501, arXiv:1309.7957.
[77] P.C. Divari, J. Cosmol. Astropart. Phys. 2018 (2018) 029, https://doi.org/10.1088/1475-7516/2018/09/029, arXiv:1808.01677.
[78] P.C. Divari, J. Cosmol. Astropart. Phys. 2020 (2020) 008, https://doi.org/10.1088/1475-7516/2020/07/008, arXiv:2004.12189.
[79] A. Vyborov, L. Inzhechik, G. Koroteev, Y.S. Lutostansky, V. Tikhonov, A. Fazliakhmetov, Phys. At. Nucl. 82 (2019) 477–482, https://doi.org/10.1134/S1063778819050132.
[80] M. Harakeh, A. Woude, Giant Resonances: Fundamental High-Frequency Modes of Nuclear Excitation, Oxford Science Publications, Oxford University Press, 2001, https://books.google.com/books?id=ux0JhIdbGT8C.
[81] K. Goeke, J. Speth, Annu. Rev. Nucl. Part. Sci. 32 (1982) 65–115, https://doi.org/10.1146/annurev.ns.32.120182.000433.
[82] V. Pandey, N. Jachowicz, M. Martini, R. González-Jiménez, J. Ryckebusch, T. Van Cuyck, N. Van Dessel, Phys. Rev. C 94 (2016) 054609, https://doi.org/10.1103/PhysRevC.94.054609, arXiv:1607.01216.
[83] V. Pandey, N. Jachowicz, T. Van Cuyck, J. Ryckebusch, M. Martini, Phys. Rev. C 92 (2015) 024606, https://doi.org/10.1103/PhysRevC.92.024606, arXiv:1412.4624.
[84] K. Balasi, K. Langanke, G. Martínez-Pinedo, Prog. Part. Nucl. Phys. 85 (2015) 33–81, https://doi.org/10.1016/j.ppnp.2015.08.001, arXiv:1503.08095v1.
[85] E. Kolbe, K. Langanke, S. Krewald, F.-K. Thielemann, Nucl. Phys. A 540 (1992) 599–620, https://doi.org/10.1016/0375-9474(92)90175-j.
[86] E. Kolbe, T.S. Kosmas, in: Symmetries in Intermediate and High Energy Physics, Springer, Berlin, Heidelberg, 2000, pp. 199–225.
[87] E. Kolbe, K. Langanke, Phys. Rev. C 63 (2001) 025802, https://doi.org/10.1103/PhysRevC.63.025802, arXiv:nucl-th/0003060.
[88] E. Kolbe, Nucl. Phys. A 719 (2003) C135–C143, https://doi.org/10.1016/s0375-9474(03)00983-7.
[89] M.-K. Cheoun, E. Ha, T. Hayakawa, S. Chiba, K. Nakamura, T. Kajino, G.J. Mathews, Phys. Rev. C 85 (2012) 065807, https://doi.org/10.1103/physrevc.85.065807, arXiv:1108.4229.
[90] A. Bandyopadhyay, P. Bhattacharjee, S. Chakraborty, K. Kar, S. Saha, Phys. Rev. D 95 (2017) 065022, https://doi.org/10.1103/PhysRevD.95.065022, arXiv:1607.05591.
[91] D. Vale, T. Rauscher, N. Paar, J. Cosmol. Astropart. Phys. 2016 (2016) 007, https://doi.org/10.1088/1475-7516/2016/02/007, arXiv:1509.07342.
[92] J. Bondorf, A. Botvina, A. Iljinov, I. Mishustin, K. Sneppen, Phys. Rep. 257 (1995) 133–221, https://doi.org/10.1016/0370-1573(94)00097-M.
[93] T. Gaitanos, H. Lenske, U. Mosel, Phys. Lett. B 663 (2008) 197–201, https://doi.org/10.1016/j.physletb.2008.04.011, arXiv:0712.3292.
[94] T. Gaitanos, H. Lenske, U. Mosel, Phys. Lett. B 675 (2009) 297–304, https://doi.org/10.1016/j.physletb.2009.04.038, arXiv:0904.2106.
[95] T. Gaitanos, H. Lenske, U. Mosel, Prog. Part. Nucl. Phys. 62 (2009) 439–444, https://doi.org/10.1016/j.ppnp.2008.12.036, arXiv:0811.3506.
[96] A.B. Larionov, M. Strikman, Phys. Rev. C 101 (2020) 014617, https://doi.org/10.1103/PhysRevC.101.014617, arXiv:1812.08231.
[97] S. Dytman, AIP Conf. Proc. 1189 (2009) 51–59, https://doi.org/10.1063/1.3274190.
[98] S. Leray, D. Mancusi, P. Kaitaniemi, J.C. David, A. Boudard, B. Braunn, J. Cugnon, J. Phys. Conf. Ser. 420 (2013) 012065, https://doi.org/10.1088/1742-6596/420/1/012065.
[99] INCL the Liège Intranuclear Cascade model, http://irfu.cea.fr/dphn/Spallation/incl.html, 2014. (Accessed 27 January 2021).
[100] D. Wright, M. Kelsey, Nucl. Instrum. Methods Phys. Res., Sect. A 804 (2015) 175–188, https://doi.org/10.1016/j.nima.2015.09.058.
[101] S. Agostinelli, J. Allison, K. Amako, et al., Nucl. Instrum. Methods Phys. Res., Sect. A 506 (2003) 250–303, https://doi.org/10.1016/S0168-9002(03)01368-8.
[102] Geant4: a simulation toolkit, https://geant4.web.cern.ch, 2021. (Accessed 27 January 2021).
[103] J. Cheng, Y.-F. Li, L.-J. Wen, S. Zhou, Phys. Rev. D 103 (2021) 053001, https://doi.org/10.1103/PhysRevD.103.053001, arXiv:2008.04633.
[104] A.J. Koning, S. Hilaire, M.C. Duijvestijn, in: O. Bersillon, F. Gunsing, E. Bauge, R. Jacqmin, S. Leray (Eds.), Proceedings of the International Conference on Nuclear Data for Science and Technology, EDP Sciences, 2007, pp. 211–214.
[105] A.J. Koning, D. Rochman, Nucl. Data Sheets 113 (2012) 2841–2934, https://doi.org/10.1016/j.nds.2012.11.002.
[106] J.M. Quesada, V. Ivanchenko, A. Ivanchenko, Prog. Nucl. Sci. Technol. 2 (2011) 936–941, https://doi.org/10.15669/pnst.2.936.
[107] S. Gardiner, Phys. Rev. C 103 (2021) 044604, https://doi.org/10.1103/PhysRevC.103.044604, arXiv:2010.02393.
[108] S. Olver, A. Townsend, Fast inverse transform sampling in one and two dimensions, arXiv:1307.1223, 2013.
[109] W. Hauser, H. Feshbach, Phys. Rev. 87 (1952) 366, https://doi.org/10.1103/PhysRev.87.366.
[110] M. Lindner, W. Rodejohann, X.-J. Xu, J. High Energy Phys. 2017 (2017), https://doi.org/10.1007/JHEP03(2017)097, arXiv:1612.04150.
[111] S. Kerman, V. Sharma, M. Deniz, H.T. Wong, J.-W. Chen, H.B. Li, S.T. Lin, C.-P. Liu, Q. Yue, TEXONO Collaboration, Phys. Rev. D 93 (2016) 113006, https://doi.org/10.1103/PhysRevD.93.113006, arXiv:1603.08786.
[112] E. Fermi, Z. Phys. 88 (1934) 161–177, https://doi.org/10.1007/BF01351864, in German.
[113] F.L. Wilson, Am. J. Phys. 36 (1968) 1150, https://doi.org/10.1119/1.1974382, English translation of Ref. [112].
[114] F. Hoyle, R.H. Fowler, Proc. R. Soc. A 166 (1938) 249–269, https://doi.org/10.1098/rspa.1938.0091.
[115] M. Cannoni, Int. J. Mod. Phys. A 32 (2017) 1730002, https://doi.org/10.1142/S0217751X17300022, arXiv:1605.00569.
[116] J. Engel, Phys. Rev. C 57 (1998) 2004–2009, https://doi.org/10.1103/PhysRevC.57.2004, arXiv:nucl-th/9711045.
[117] C. Volpe, N. Auerbach, G. Colò, N. Van Giai, Phys. Rev. C 65 (2002) 044603, https://doi.org/10.1103/PhysRevC.65.044603, arXiv:nucl-th/0103039.
[118] R. Capote, M. Herman, P. Obložinský, P. Young, S. Goriely, T. Belgya, A. Ignatyuk, A.J. Koning, S. Hilaire, V.A. Plujko, et al., Nucl. Data Sheets 110 (2009) 3107–3214, https://doi.org/10.1016/j.nds.2009.10.004.
[119] A.J. Koning, S. Hilaire, S. Goriely, Nucl. Phys. A 810 (2008) 13–76, https://doi.org/10.1016/j.nuclphysa.2008.06.005.
[120] A. Koning, J. Delaroche, Nucl. Phys. A 713 (2003) 231–310, https://doi.org/10.1016/S0375-9474(02)01321-0.
[121] D. Madland, in: Proceedings of a Specialists' Meeting on Preequilibrium Nuclear Reactions, 1988, pp. 103–110, https://www.oecd-nea.org/science/docs/1988/neandc1988-245-u.pdf.
[122] NIST Digital Library of Mathematical Functions, Release 1.0.26 of 2020-03-15, http://dlmf.nist.gov/, 2020, F.W.J. Olver, A.B. Olde Daalhuis, D.W. Lozier, B.I. Schneider, R.F. Boisvert, C.W. Clark, B.R. Miller, B.V. Saunders, H.S. Cohl, M.A. McClain (Eds.).
[123] B.V. Noumerov, Mon. Not. R. Astron. Soc. 84 (1924) 592–602, https://doi.org/10.1093/mnras/84.8.592.
[124] B. Numerov, Astron. Nachr. 230 (1927) 359–364, https://doi.org/10.1002/asna.19272301903.
[125] J. Thijssen, Computational Physics, 2 edn., Cambridge University Press, 2007, http://books.google.com/books?vid=ISBN9780521833462.
[126] W.E. Brown, Random number generation in C++11, http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3551.pdf, 2013. (Accessed 27 January 2021).
[127] M. Matsumoto, T. Nishimura, ACM Trans. Model. Comput. Simul. 8 (1998) 3–30, https://doi.org/10.1145/272991.272995.
[128] T. Nishimura, ACM Trans. Model. Comput. Simul. 10 (2000) 348–357, https://doi.org/10.1145/369534.369540.
[129] T. Becker, C/C++ Users J. 19 (2001) 51–57, https://web.archive.org/web/20140811092434/https://collaboration.cmc.ec.gc.ca/science/rpn/biblio/ddj/Website/articles/CUJ/2001/0108/becker/becker.htm.
[130] R.P. Brent, in: Algorithms for Minimization Without Derivatives, Prentice-Hall, 1973, pp. 61–80, https://books.google.com/books?id=Ee5QAAAAMAAJ.
[131] S. Olver, A. Townsend, Inverse transform sampling MATLAB implementation, https://github.com/dlfivefifty/InverseTransformSampling, 2013.
[132] T.A. Driscoll, N. Hale, L.N. Trefethen, Chebfun Guide, Pafnuty Publications, 2014, http://www.chebfun.org/docs/guide/.
[133] Z. Battles, L. Trefethen, SIAM J. Sci. Comput. 25 (2004) 1743–1770, https://doi.org/10.1137/S1064827503430126, http://www.chebfun.org/publications/chebfun_paper.pdf.

[134] J. Berrut, L. Trefethen, SIAM Rev. 46 (2004) 501–517, https://doi.org/10.1137/S0036144502417715, https://people.maths.ox.ac.uk/trefethen/barycentric.pdf.
[135] A. Fernandes, P.N. Swarztrauber, R. Valent, FFTPACK4: fast Fourier transform, https://people.sc.fsu.edu/~jburkardt/c_src/fftpack4/fftpack4.html, 2019. (Accessed 27 January 2021).
[136] P.N. Swarztrauber, in: G. Rodrigue (Ed.), Parallel Computations, Computational Techniques, Academic Press, 1982, pp. 51–83, https://books.google.com/books?id=ur7SBQAAQBAJ.
[137] P.N. Swarztrauber, http://www.netlib.org/fftpack, 2021. (Accessed 27 January 2021).
[138] P.J. Mohr, B.N. Taylor, D.B. Newell, Rev. Mod. Phys. 84 (2012) 1527–1605, https://doi.org/10.1103/RevModPhys.84.1527, https://physics.nist.gov/cuu/Constants/Preprints/lsa2010.pdf.
[139] G. Audi, M. Wang, A. Wapstra, F. Kondev, M. MacCormick, X. Xu, B. Pfeiffer, Chin. Phys. C 36 (2012) 1287–1602, https://doi.org/10.1088/1674-1137/36/12/002.
[140] W.D. Myers, W.J. Swiatecki, Nucl. Phys. 81 (1966) 1–60, https://doi.org/10.1016/0029-5582(66)90639-0.
[141] C.W. Clenshaw, A.R. Curtis, Numer. Math. 2 (1960) 197–205, https://doi.org/10.1007/BF01386223, http://www.digizeitschriften.de/dms/resolveppn/?PID=GDZPPN001163442.
[142] GNU Bash, http://gnu.org/software/bash/, 2020. (Accessed 27 January 2021).
[143] GCC, the GNU Compiler Collection, https://gcc.gnu.org/, 2021. (Accessed 27 January 2021).
[144] Clang: a C language family frontend for LLVM, https://clang.llvm.org/, 2021. (Accessed 27 January 2021).
[145] GNU Make, https://www.gnu.org/software/make/, 2020. (Accessed 27 January 2021).
[146] Homebrew: the Missing Package Manager for macOS (or Linux), https://brew.sh/, 2021. (Accessed 27 January 2021).
[147] IEEE Standard for Information Technology–Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7, 2018. https://doi.org/10.1109/IEEESTD.2018.8277153, IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008).
[148] S. Gardiner, MARLEY user guide, https://www.marleygen.org, 2021. (Accessed 27 January 2021).
[149] pkg-config, https://www.freedesktop.org/wiki/Software/pkg-config/, 2018. (Accessed 27 January 2021).
[150] T. Bray, The JavaScript Object Notation (JSON) data interchange format, https://doi.org/10.17487/RFC8259, 2017.
[151] N. Semmel, SimpleJSON, https://github.com/nbsdx/SimpleJSON, 2016.
[152] J.F. Beacom, Annu. Rev. Nucl. Part. Sci. 60 (2010) 439–462, https://doi.org/10.1146/annurev.nucl.010909.083331, arXiv:1004.3311.
[153] M.-K. Cheoun, E. Ha, T. Kajino, Eur. Phys. J. A 48 (2012) 137, https://doi.org/10.1140/epja/i2012-12137-y.
[154] M. Bhattacharya, A. García, N.I. Kaloskamis, E.G. Adelberger, H.E. Swanson, R. Anne, M. Lewitowicz, M.G. Saint-Laurent, W. Trinder, C. Donzaud, et al., Phys. Rev. C 58 (1998) 3677–3687, https://doi.org/10.1103/PhysRevC.58.3677.
[155] W. Liu, M. Hellström, R. Collatz, J. Benlliure, L. Chulkov, D.C. Gil, F. Farget, H. Grawe, Z. Hu, N. Iwasa, et al., Phys. Rev. C 58 (1998) 2677–2688, https://doi.org/10.1103/PhysRevC.58.2677.
[156] M. Bhattacharya, C.D. Goodman, A. García, Phys. Rev. C 80 (2009) 055501, https://doi.org/10.1103/PhysRevC.80.055501.
[157] T. Totani, K. Sato, H.E. Dalhed, J.R. Wilson, Astrophys. J. 496 (1998) 216, https://doi.org/10.1086/305364, arXiv:astro-ph/9710203.
[158] M.-Y. Huang, X.-H. Guo, B.-L. Young, Chin. Phys. C 40 (2016) 073102, https://doi.org/10.1088/1674-1137/40/7/073102, arXiv:1511.00806.
[159] M.T. Keil, G.G. Raffelt, H.-T. Janka, Astrophys. J. 590 (2003) 971–991, https://doi.org/10.1086/375130, arXiv:astro-ph/0208035.
[160] T.S. Kosmas, D.K. Papoulias, M. Tórtola, J.W.F. Valle, Phys. Rev. D 96 (2017) 063013, https://doi.org/10.1103/PhysRevD.96.063013, arXiv:1703.00054.
[161] A. Bolozdynya, F. Cavanna, Y. Efremenko, G. Garvey, et al., Opportunities for neutrino physics at the Spallation Neutron Source: a white paper, arXiv:1211.5199, 2012.
[162] P.A. Zyla, R.M. Barnett, J. Beringer, O. Dahl, D.A. Dwyer, D.E. Groom, C.-J. Lin, K.S. Lugovsky, E. Pianori, D.J. Robinson, et al., Particle Data Group, Prog. Theor. Exp. Phys. 2020 (2020) 083C01, https://doi.org/10.1093/ptep/ptaa104.
[163] T. Sjöstrand, in: G. Altarelli, R. Kleiss, C. Verzegnassi (Eds.), Z Physics at LEP 1, volume 3, Event Generators and Software, 1989, pp. 143–340, http://inspirehep.net/record/288141/files/.
[164] A. Buckley, P. Ilten, D. Konstantinov, L. Lönnblad, J. Monk, W. Pokorski, T. Przedzinski, A. Verbytskyi, Comput. Phys. Commun. 260 (2021) 107310, https://doi.org/10.1016/j.cpc.2020.107310, arXiv:1912.08005.
[165] Trees in five steps, https://ph-root-2.cern.ch/d/trees-five-steps.html, 2018. (Accessed 27 January 2021).
[166] M. Goto, CINT: C++ Interpreter, http://www.hanno.jp/gotom/Cint.html, 2015. (Accessed 27 January 2021).
[167] V. Vassilev, P. Canal, A. Naumann, L. Moneta, P. Russo, J. Phys. Conf. Ser. 396 (2012) 052071, https://doi.org/10.1088/1742-6596/396/5/052071.
[168] D. van Heesch, Doxygen, https://www.doxygen.nl/, 2021. (Accessed 27 January 2021).
[169] C. Andreopoulos, C. Barry, S. Dytman, H. Gallagher, T. Golan, R. Hatcher, G. Perdue, J. Yarba, The GENIE neutrino Monte Carlo generator: physics and user manual, arXiv:1510.05494, 2015.
[170] E.D. Church, LArSoft: a software package for liquid argon time projection drift chambers, arXiv:1311.6774, 2013.
[171] E. Snider, G. Petrillo, J. Phys. Conf. Ser. 898 (2017) 042057, https://doi.org/10.1088/1742-6596/898/4/042057.
[172] R. Pordes, E. Snider, in: 38th International Conference on High Energy Physics, ICHEP2016, 2017, PoS(ICHEP2016)182.
[173] LArSoft Collaboration, Software for LArTPCs, https://larsoft.org, 2021. (Accessed 27 January 2021).
[174] MicroBooNE Collaboration, MeV scale physics in MicroBooNE, https://microboone.fnal.gov/wp-content/uploads/MICROBOONE-NOTE-1076-PUB.pdf, 2020. (Accessed 23 June 2021).
[175] Geant4 book for application developers, http://geant4-userdoc.web.cern.ch/geant4-userdoc/UsersGuides/ForApplicationDeveloper/html/index.html, 2021. (Accessed 27 January 2021).
[176] E. Mendoza, D. Cano-Ott, Update of the evaluated neutron cross section libraries for the Geant4 code, Technical Report INDC(NDS)-0758, International Atomic Energy Agency (IAEA), 2018, https://www-nds.iaea.org/publications/indc/indc-nds-0758/.
[177] E. Mendoza, D. Cano-Ott, C. Guerrero, R. Capote, New evaluated neutron cross section libraries for the GEANT4 code, Technical Report INDC(NDS)-0612, International Atomic Energy Agency (IAEA), 2012, https://www-nds.iaea.org/publications/indc/indc-nds-0612/.
[178] C. Green, J. Kowalkowski, M. Paterno, M. Fischler, L. Garren, Q. Lu, J. Phys. Conf. Ser. 396 (2012) 022020, https://doi.org/10.1088/1742-6596/396/2/022020.
[179] C. Anderson, M. Antonello, B. Baller, et al., ArgoNeuT Collaboration, J. Instrum. 7 (2012) P10019, https://doi.org/10.1088/1748-0221/7/10/p10019, arXiv:1205.6747.
[180] R. Acciarri, C. Adams, J. Asaadi, et al., LArIAT Collaboration, J. Instrum. 15 (2020) P04026, https://doi.org/10.1088/1748-0221/15/04/p04026, arXiv:1911.10379.
[181] R. Acciarri, C. Adams, R. An, A. Aparicio, et al., MicroBooNE Collaboration, J. Instrum. 12 (2017) P02017, https://doi.org/10.1088/1748-0221/12/02/p02017, arXiv:1612.05824.
[182] C. Rubbia, M. Antonello, P. Aprili, et al., ICARUS Collaboration, J. Instrum. 6 (2011) P07011, https://doi.org/10.1088/1748-0221/6/07/p07011.
[183] N. McConkey, J. Phys. Conf. Ser. 888 (2017) 012148, https://doi.org/10.1088/1742-6596/888/1/012148.
[184] B. Abi, R. Acciarri, M. Acero, G. Adamov, D. Adams, M. Adinolfi, Z. Ahmad, J. Ahmed, T. Alion, S.A. Monsalve, et al., DUNE Collaboration, J. Instrum. 15 (2020) T08008, https://doi.org/10.1088/1748-0221/15/08/t08008, arXiv:2002.02967.
[185] D. Heck, J. Knapp, J. Capdevielle, G. Schatz, T. Thouw, CORSIKA: a Monte Carlo code to simulate extensive air showers, Technical Report FZKA-6019, Institut für Kernphysik, 1998, https://doi.org/10.5445/IR/270043064.
[186] C. Hagmann, D. Lange, D. Wright, in: 2007 IEEE Nuclear Science Symposium Conference Record, vol. 2, 2007, pp. 1143–1146.
[187] S. Gardiner, Nuclear effects in neutrino detection, Ph.D. thesis, University of California, Davis, 2018.
[188] R. Putz, Specification of the Fermilab Hierarchical Configuration Language, https://cdcvs.fnal.gov/redmine/attachments/download/6639/grammar.pdf, 2012. (Accessed 27 January 2021).
[189] M. Acero, P. Adamson, G. Agam, L. Aliaga, T. Alion, V. Allakhverdian, N. Anfimov, A. Antoshkin, E. Arrieta-Diaz, L. Asquith, et al., NOvA Collaboration, J. Cosmol. Astropart. Phys. 2020 (2020) 014, https://doi.org/10.1088/1475-7516/2020/10/014, arXiv:2005.07155.
[190] M. Ikeda, A. Takeda, Y. Fukuda, et al., Super-Kamiokande Collaboration, Astrophys. J. 669 (2007) 519–524, https://doi.org/10.1086/521547, arXiv:0706.2283.
[191] N. Van Dessel, N. Jachowicz, A. Nikolakopoulos, Phys. Rev. C 100 (2019) 055503, https://doi.org/10.1103/PhysRevC.100.055503, arXiv:1903.07726.
[192] N. Van Dessel, A. Nikolakopoulos, N. Jachowicz, Phys. Rev. C 101 (2020) 045502, https://doi.org/10.1103/PhysRevC.101.045502, arXiv:1912.10714.
[193] J.A. Dror, G. Elor, R. McGehee, Phys. Rev. Lett. 124 (2020) 181301, https://doi.org/10.1103/PhysRevLett.124.181301, arXiv:1905.12635.
[194] P. Stowell, C. Wret, C. Wilkinson, L. Pickering, et al., J. Instrum. 12 (2017) P01016, https://doi.org/10.1088/1748-0221/12/01/p01016, arXiv:1612.07393.
[195] J. Barrow, M. Betancourt, L. Cremonesi, S. Dytman, et al., Summary of workshop on common neutrino event generator tools, arXiv:2008.06566, 2020.