

# LUMINA-SN

Monte Carlo Radiative Transfer for Type Ia Supernovae

A Complete Technical Manual

Professor Juhan Kim

February 2026

Copyright © 2026 Juhan Kim

LUMINA-SN: LUMINOSITY-DRIVEN MONTE CARLO SUPERNOVA SPECTRAL SYNTHESIZER

This code implements 1D Monte Carlo radiative transfer in homologously expanding supernova ejecta, following the formalism of Lucy [3–5] and the TARDIS code [2]. Written in C99 with CUDA GPU acceleration.

*Version 2.0 — February 2026*

# Contents

---

I	Physics & Theory	7
1	Introduction to Type Ia Supernovae	9
1.1	What is a Type Ia Supernova?	9
1.2	The Expanding Ejecta	9
1.3	Spectral Features	9
1.4	The Inverse Problem	10
2	Radiative Transfer in Expanding Atmospheres	11
2.1	The Transfer Equation	11
2.2	The Sobolev Approximation	11
2.3	The Dilute Radiation Field	12
2.4	Frame Transformations	12
2.5	Bound-Free and Free-Free Opacity	12
3	Monte Carlo Photon Transport	13
3.1	The Indivisible Energy Packet Formalism	13
3.2	Packet Initialization	13
3.2.1	Frequency Sampling from the Planck Function	13
3.2.2	Angular Distribution	14
3.3	The Packet Propagation Loop	14
3.4	Distance Calculations	14
3.4.1	Distance to Shell Boundary	14
3.4.2	Distance to Electron Scattering	14
3.4.3	Distance to Sobolev Resonance	14
3.5	Line Interaction Types	15
3.5.1	Resonant Scattering (Mode 0)	15
3.5.2	Downbranching (Mode 1)	15
3.5.3	Macro-Atom (Mode 2)	15
3.6	Monte Carlo Estimators	15
3.7	Convergence: The Iteration Loop	16

<b>4</b>	<b>Spectrum Synthesis Methods</b>	<b>17</b>
4.1	Overview of the Three Methods	17
4.2	Real Packets	17
4.2.1	Principle	17
4.2.2	Algorithm	18
4.2.3	Strengths and Limitations	18
4.3	Virtual Packets	18
4.3.1	Physical Motivation	18
4.3.2	The $p$ - $z$ Coordinate System	18
4.3.3	Algorithm	19
4.3.4	Optical Depth Accumulation	19
4.3.5	Emission Points	20
4.3.6	Strengths and Limitations	20
4.4	Rotation Packets	20
4.4.1	Physical Motivation	20
4.4.2	Doppler Weighting	21
4.4.3	Face-On Observer	21
4.4.4	Algorithm	21
4.4.5	Strengths and Limitations	21
4.5	Performance Comparison	22
4.6	When to Use Each Method	22
4.6.1	Command-Line Usage	22
II	Code Architecture	25
<b>5</b>	<b>Overview &amp; Build System</b>	<b>27</b>
5.1	File Structure	27
5.2	Build System	27
5.3	Dependencies	28
5.4	Execution	28
<b>6</b>	<b>Data Structures</b>	<b>29</b>
6.1	The RPacket Structure	29
6.2	Geometry	29
6.3	Opacity State	30
6.4	Plasma State	30
6.5	Atomic Data	30
6.6	Monte Carlo Estimators	31
<b>7</b>	<b>Transport Engine</b>	<b>33</b>
7.1	Overview	33
7.2	The trace_packet Function	33
7.3	Sobolev Line Sweep	33
7.4	Thomson Scattering	34

7.5	Boundary Crossing .....	34
<b>8</b>	<b>Plasma Physics Solver .....</b>	<b>35</b>
8.1	Overview .....	35
8.2	Step 1: Partition Functions .....	35
8.3	Step 2: Electron Density .....	35
8.4	Step 3: Nebular Saha Ionization .....	36
8.5	Step 4: $\tau_{\text{Sobolev}}$ Update .....	36
8.6	Step 5: Radiation Field Update .....	36
<b>9</b>	<b>Atomic Data System .....</b>	<b>37</b>
9.1	Data Sources .....	37
9.2	The NPY Format Reader .....	37
9.3	The CSV Parser .....	37
<b>10</b>	<b>CUDA GPU Implementation .....</b>	<b>39</b>
10.1	Design Philosophy .....	39
10.2	Memory Layout .....	39
10.3	Kernel Launch Configuration .....	39
10.4	Random Number Generation .....	40
10.5	Atomic Estimator Updates .....	40
10.6	Performance .....	40
10.7	Resolved GPU Bugs .....	41
III	Usage & Applications .....	43
<b>11</b>	<b>Installation &amp; Quick Start .....</b>	<b>45</b>
11.1	Prerequisites .....	45
11.2	Step-by-Step Setup .....	45
11.3	Input Directory Structure .....	46
11.4	Output Files .....	46
<b>12</b>	<b>The Ejecta Model .....</b>	<b>47</b>
12.1	Three-Zone Composition .....	47
12.2	Broken Power-Law Density .....	47
12.3	Physical Parameter Space .....	47
<b>13</b>	<b>Parameter Fitting .....</b>	<b>49</b>
13.1	Fitting Strategy .....	49
13.1.1	Phase 1: Coarse Exploration .....	49
13.1.2	Phase 2: Refinement .....	49
13.1.3	Phase 3: Production .....	49
13.2	Objective Function .....	49

<b>14</b>	<b>Physical Constants &amp; Reference Values .....</b>	<b>51</b>
	<b>Bibliography .....</b>	<b>53</b>
	<b>Index .....</b>	<b>55</b>



## Physics & Theory





# Introduction to Type Ia Supernovae

## 1.1 What is a Type Ia Supernova?

A Type Ia supernova (SN Ia) is the thermonuclear explosion of a carbon-oxygen white dwarf star that has reached a critical mass near the Chandrasekhar limit ( $M_{\text{Ch}} \approx 1.4 M_{\odot}$ ). The explosion completely unbinds the star, leaving no compact remnant, and synthesizes approximately  $0.6 M_{\odot}$  of radioactive  $^{56}\text{Ni}$  [7].

**Important:** SN Ia are cosmological standard candles: their peak luminosity correlates with their light-curve decline rate (the Phillips relation, [8]), enabling precise distance measurements to galaxies. This led to the discovery of the accelerating expansion of the Universe.

## 1.2 The Expanding Ejecta

After the explosion, the ejecta expand freely into a vacuum. Within hours, the expansion reaches a state of **homologous expansion**:

$$v(r, t) = \frac{r}{t} \quad (1.1)$$

where  $r$  is the radial distance from the center and  $t$  is the time since explosion. This means velocity maps directly to radius: faster material is farther out.

**Definition 1.1 — Homologous Expansion.** In homologous expansion, each fluid element moves at constant velocity. The density at any velocity coordinate  $v$  evolves as:

$$\rho(v, t) = \rho_0(v) \left( \frac{t_0}{t} \right)^3 \quad (1.2)$$

where  $\rho_0(v)$  is the density at reference epoch  $t_0$ , and the  $t^{-3}$  factor comes from the 3D volumetric dilution.

## 1.3 Spectral Features

SN Ia spectra are dominated by **P Cygni profiles**: blueshifted absorption troughs paired with redshifted emission peaks. These arise because:

1. Material approaching the observer (along the line of sight) absorbs photons at a blueshifted wavelength.
2. The surrounding envelope re-emits photons isotropically, producing a net emission component redward of the rest wavelength.

Table 1.1: Key spectral features in SN Ia near maximum light.

Ion	Rest $\lambda$ (Å)	Observed Range (Å)	Diagnostic Value
Si II	6355	5800–6500	Expansion velocity, temperature
Si II	5972	5600–6000	Temperature indicator
S II	5454, 5640	5200–5700	“W” feature, burning completeness
Ca II	3934, 3968	3600–4000	H&K lines, high-velocity features
Ca II	8498, 8542, 8662	8000–8800	IR triplet
Fe II	4500–5200	4300–5200	Iron-group blanketing
O I	7774	7400–7900	Unburned oxygen indicator

## 1.4 The Inverse Problem

Given an observed spectrum, we want to determine the physical parameters of the explosion:

- Luminosity  $L$  and photospheric temperature  $T_{\text{inner}}$
- Density profile  $\rho(v)$  and its power-law exponents
- Chemical composition as a function of velocity (abundance tomography)
- Time since explosion  $t_{\text{exp}}$

LUMINA-SN solves the *forward problem*: given these parameters, compute the emergent spectrum. Combined with Bayesian inference (Part III), this enables solving the inverse problem.

# Radiative Transfer in Expanding Atmospheres

## 2.1 The Transfer Equation

The specific intensity  $I_\nu$  along a ray satisfies:

$$\frac{dI_\nu}{ds} = -\kappa_\nu I_\nu + j_\nu \quad (2.1)$$

where  $s$  is the path length,  $\kappa_\nu$  is the absorption coefficient (opacity), and  $j_\nu$  is the emissivity.

In a supernova atmosphere, three opacity sources contribute:

1. **Electron scattering** (Thomson): frequency-independent,  $\sigma_T = 6.652 \times 10^{-25} \text{ cm}^2$
2. **Line opacity** (Sobolev): resonant absorption in atomic transitions
3. **Continuum opacity**: bound-free and free-free (negligible in SN Ia, see §2.5)

## 2.2 The Sobolev Approximation

In homologous expansion, the velocity gradient  $dv/dr = 1/t_{\text{exp}}$  is constant. A photon sweeps through a line's resonance frequency over a very short distance (the *Sobolev length*). This means line interactions are *local*: each line either absorbs the photon or lets it pass.

**Definition 2.1 — Sobolev Optical Depth.** The optical depth of a line transition  $l \rightarrow u$  in the Sobolev approximation is:

$$\tau_{\text{Sob}} = \frac{\pi e^2}{m_e c} f_{lu} \lambda_0 t_{\text{exp}} n_l \left( 1 - \frac{g_l n_u}{g_u n_l} \right) \quad (2.2)$$

where  $f_{lu}$  is the oscillator strength,  $\lambda_0$  is the rest wavelength,  $n_l$  and  $n_u$  are the lower and upper level populations, and  $g_l$ ,  $g_u$  are the statistical weights.

The numerical coefficient is:

$$\frac{\pi e^2}{m_e c} = 0.02654 \text{ cm}^2 \text{ s}^{-1} \quad (\text{CGS}) \quad (2.3)$$

The *escape probability* from a Sobolev line is:

$$\beta_{\text{Sob}} = \frac{1 - e^{-\tau_{\text{Sob}}}}{\tau_{\text{Sob}}} \quad (2.4)$$

## 2.3 The Dilute Radiation Field

Far from the photosphere, the radiation field is diluted. The mean intensity is:

$$J_\nu = W B_\nu(T_{\text{rad}}) \quad (2.5)$$

where  $W$  is the **dilution factor** and  $T_{\text{rad}}$  is the **radiation temperature**. For a geometrically thin photosphere at radius  $R_{\text{phot}}$ :

$$W(r) = \frac{1}{2} \left( 1 - \sqrt{1 - \left( \frac{R_{\text{phot}}}{r} \right)^2} \right) \quad (2.6)$$

**Remark 2.1 — Physical Interpretation of  $W$ .** At  $r = R_{\text{phot}}$ :  $W = 0.5$  (hemisphere illuminated). At  $r \gg R_{\text{phot}}$ :  $W \approx R_{\text{phot}}^2/(4r^2) \rightarrow 0$  (point source). In practice, Monte Carlo estimators yield  $W$  values that include the effects of line scattering and fluorescence.

## 2.4 Frame Transformations

In homologous expansion, the comoving-frame (CMF) frequency differs from the lab-frame frequency:

$$\nu_{\text{cmf}} = \nu_{\text{lab}} \left( 1 - \frac{\mu v}{c} \right) = \nu_{\text{lab}} \left( 1 - \frac{\mu r}{c t_{\text{exp}}} \right) \quad (2.7)$$

where  $\mu = \cos \theta$  is the direction cosine of the photon with respect to the radial direction, and  $v = r/t_{\text{exp}}$ .

**Important:** As a photon propagates through a shell, the comoving-frame frequency changes *linearly* with distance  $s$  along the ray:

$$\nu_{\text{cmf}}(s) = \nu_{\text{lab}} \left( 1 - \frac{r_0 \mu_0 + s}{c t_{\text{exp}}} \right) \quad (2.8)$$

This monotonic frequency sweep is the basis of the Sobolev sweep algorithm (§3.4.3).

## 2.5 Bound–Free and Free–Free Opacity

In SN Ia ejecta, continuum opacities are negligible because:

- **Bound–free:** Optical photons (1.5–2.5 eV) are far below the ionization thresholds of the dominant species (Si II: 16.35 eV, Fe II: 16.19 eV, Ca II: 11.87 eV). Only far-UV photons ( $< 1000 \text{ \AA}$ ) could ionize these ions.
- **Free–free:** Electron densities are very low ( $n_e \sim 10^9 \text{ cm}^{-3}$  inner,  $\sim 10^6 \text{ cm}^{-3}$  outer), giving  $\kappa_{\text{ff}} \sim 10^{-20} \text{ cm}^{-1}$ .

The combined continuum optical depth across a shell is  $\tau_{\text{cont}} \sim 10^{-6}$  to  $10^{-9}$ , confirming that **line opacity dominates** the spectrum formation in SN Ia.

# Monte Carlo Photon Transport

## 3.1 The Indivisible Energy Packet Formalism

LUMINA follows the Lucy [3, 4] formalism, where photons are represented as discrete **energy packets** ( $r$ -packets) with properties:

Table 3.1: Properties of an  $r$ -packet in LUMINA.

Symbol	Description	Unit
$r$	Radial position	cm
$\mu$	Direction cosine ( $\cos \theta$ )	–
$\nu$	Lab-frame frequency	Hz
$\varepsilon$	Packet energy	erg
$i_{\text{shell}}$	Current shell index	–

All packets carry the same energy:

$$\varepsilon_{\text{pkt}} = \frac{L_{\text{inner}} \Delta t}{N_{\text{packets}}} \quad (3.1)$$

where  $L_{\text{inner}}$  is the luminosity at the inner boundary and  $\Delta t$  is the simulation time interval.

## 3.2 Packet Initialization

Packets are emitted from the photosphere ( $r = R_{\text{inner}}$ ) with:

### 3.2.1 Frequency Sampling from the Planck Function

The emission frequency is sampled from a blackbody at temperature  $T_{\text{inner}}$  using the Bjorkman and Wood [1] method:

1. Draw  $\xi_0 \sim U(0, 1)$
2. Find  $l_{\text{min}}$  such that  $\sum_{i=1}^{l_{\text{min}}} i^{-4} \geq \frac{\pi^4}{90} \xi_0$
3. Draw  $\xi_1, \xi_2, \xi_3, \xi_4 \sim U(0, 1)$
4. Compute  $x = -\ln(\xi_1 \xi_2 \xi_3 \xi_4) / l_{\text{min}}$
5. Set  $\nu = x k_B T_{\text{inner}} / h$

This is exact (no rejection) and samples directly from  $B_\nu(T)$ .

### 3.2.2 Angular Distribution

The direction cosine is sampled from a limb-darkened distribution:

$$\mu = \sqrt{\xi}, \quad \xi \sim U(0, 1) \quad (3.2)$$

ensuring that more packets are emitted along the normal direction.

## 3.3 The Packet Propagation Loop

Each packet undergoes a loop until it escapes ( $r > R_{\text{outer}}$ ) or is reabsorbed ( $r < R_{\text{inner}}$  moving inward):

#### Algorithm: Single Packet Loop

1. **Draw random optical depth:**  $\tau_{\text{event}} = -\ln(\xi)$
2. **Compute distances** to possible events:
  - $d_{\text{boundary}}$ : distance to next shell wall
  - $d_{\text{line}}$ : distance to next Sobolev resonance
  - $d_e$ : distance to electron scattering ( $\tau = n_e \sigma_T d$ )
3. **Select minimum:**  $d_{\text{min}} = \min(d_{\text{boundary}}, d_{\text{line}}, d_e)$
4. **Move packet:** update  $r$ ,  $\mu$ , accumulate estimators
5. **Handle interaction:**
  - Boundary: change shell, check escape/reabsorption
  - Line: scatter, downbranch, or activate macro-atom
  - Electron: Thomson scatter (isotropic re-emission)
6. **Repeat** from step 1

## 3.4 Distance Calculations

### 3.4.1 Distance to Shell Boundary

For a packet at  $(r, \mu)$  in shell  $[r_{\text{in}}, r_{\text{out}}]$ :

**Outward** ( $\mu > 0$  or  $r > r_{\text{in}}/\mu$ ):

$$d_{\text{out}} = \sqrt{r_{\text{out}}^2 - r^2(1 - \mu^2)} - r\mu \quad (3.3)$$

**Inward** ( $\mu < 0$  and impact parameter  $< r_{\text{in}}$ ):

$$d_{\text{in}} = -r\mu - \sqrt{r_{\text{in}}^2 - r^2(1 - \mu^2)} \quad (3.4)$$

### 3.4.2 Distance to Electron Scattering

$$d_e = \frac{\tau_{\text{event}}}{n_e \sigma_T} \quad (3.5)$$

### 3.4.3 Distance to Sobolev Resonance

As the packet traverses the shell, the CMF frequency sweeps through the line list. For a line at rest frequency  $\nu_{\text{line}}$ :

$$d_{\text{line}} = \frac{\nu_{\text{cmf}}(r, \mu) - \nu_{\text{line}}}{\nu_{\text{lab}}} c t_{\text{exp}} \quad (3.6)$$

**Important:** The Sobolev sweep must scan from the *entry* frequency to the *exit* frequency of each shell. The original LUMINA code used a fixed  $\pm 1\%$  window, which missed lines during thick shell crossings. The corrected algorithm (Task #067) uses the full Doppler sweep, increasing line interactions from 0.6% to  $> 50\%$  of packet steps.

## 3.5 Line Interaction Types

When a packet encounters a line, three interaction modes are available:

### 3.5.1 Resonant Scattering (Mode 0)

The packet is absorbed and re-emitted at the same line frequency with a new random direction:

$$\mu_{\text{new}} \sim U(-1, +1), \quad \nu_{\text{new}} = \nu_{\text{line}} \quad (3.7)$$

### 3.5.2 Downbranching (Mode 1)

The packet is absorbed into the upper level and re-emitted in a *different* line, selected from the downbranching probability distribution:

$$P(\text{emit in line } j) = \frac{A_{u \rightarrow l_j}}{\sum_k A_{u \rightarrow l_k}} \quad (3.8)$$

This enables fluorescence: a UV photon absorbed in one line can be re-emitted in the optical.

### 3.5.3 Macro-Atom (Mode 2)

The full macro-atom formalism [4, 5] activates the atom at the upper energy level and follows a Markov chain of internal transitions:

#### Algorithm: Macro-Atom Transition Walk

1. Start at activation level  $l$
  2. Look up transition block for level  $l$  (list of possible transitions)
  3. Draw  $\xi \sim U(0, 1)$ , select transition  $k$  from cumulative probabilities
  4. If transition type  $\geq 0$  (internal): move to destination level, go to step 2
  5. If transition type  $< 0$  (emission): emit in the associated line, **exit**
- Maximum 500 iterations (safety limit).

Emission types include:

- -1: Bound-bound emission (emit photon in a specific line)
- -2: Bound-free emission (photoionization, thermalization)
- -3: Free-free emission (thermal)
- -4: Adiabatic cooling

## 3.6 Monte Carlo Estimators

As packets propagate, they contribute to estimators that probe the radiation field.

**Definition 3.1 — Mean Intensity Estimator.**

$$\hat{j}_\nu = \frac{1}{4\pi V \Delta t} \sum_{\text{packets}} \varepsilon_{\text{cmf}} \Delta s \quad (3.9)$$

In practice, two scalar estimators are accumulated per shell:

$$j = \sum_{\text{packets}} \varepsilon_{\text{cmf}} \Delta s \quad (3.10)$$

$$\bar{\nu} = \frac{\sum_{\text{packets}} \varepsilon_{\text{cmf}} \nu_{\text{cmf}} \Delta s}{\sum_{\text{packets}} \varepsilon_{\text{cmf}} \Delta s} \quad (3.11)$$

From these, the radiation temperature and dilution factor are recovered:

$$T_{\text{rad}} = T_{\text{rad,const}} \times \frac{\bar{\nu}}{j} \quad (3.12)$$

$$W = \frac{j}{4 \sigma_{\text{SB}} T_{\text{rad}}^4 \Delta t V} \quad (3.13)$$

where  $T_{\text{rad,const}} = \frac{\pi^4}{15 \cdot 24 \cdot \zeta(5)} \frac{h}{k_B} = 1.2523 \times 10^{-11} \text{ K}\cdot\text{s}$ .

### 3.7 Convergence: The Iteration Loop

LUMINA iterates between transport and plasma calculations:

1. **Transport:** Run  $N_{\text{packets}}$  through the ejecta, accumulating estimators.
2. **Radiation field update:** Compute  $T_{\text{rad}}$ ,  $W$  from estimators, with **damping**:

$$X_{\text{new}} = X_{\text{old}} + d \cdot (X_{\text{est}} - X_{\text{old}}), \quad d = 0.5 \quad (3.14)$$

3. **Plasma update:** Recompute ionization, level populations,  $\tau_{\text{Sob}}$ .
4.  $T_{\text{inner}}$  **update** (after hold iterations):

$$T_{\text{inner,new}} = T_{\text{inner,old}} + d \cdot \left[ T_{\text{inner}} \left( \frac{L_{\text{emitted}}}{L_{\text{requested}}} \right)^{-0.5} - T_{\text{inner,old}} \right] \quad (3.15)$$

**Remark 3.1 — Exponent  $-0.5$  vs  $+0.25$ .** Naïvely, Stefan–Boltzmann gives  $T \propto L^{0.25}$ , suggesting the correction factor should be  $(L_{\text{em}}/L_{\text{req}})^{0.25}$ . However, TARDIS uses the exponent  $-0.5$  because changing  $T_{\text{inner}}$  also changes the opacity (through ionization), leading to non-linear feedback. The empirical  $-0.5$  accounts for this.



## Spectrum Synthesis Methods

LUMINA implements three distinct methods for constructing the emergent spectrum from the Monte Carlo simulation. Each method makes different trade-offs between physical fidelity, noise characteristics, and computational cost.

### 4.1 Overview of the Three Methods

Table 4.1: Comparison of the three spectrum synthesis methods.

Property	Real Packet	Virtual Packet	Rotation Packet
Transport type	Full Monte Carlo	Formal integral (ray tracing)	Post-processing
Cost per real packet	$1\times$ (baseline)	$+10\times$ (per interaction)	Negligible
Noise	$\propto N^{-1/2}$	Lower (many rays)	Same as real
Observer direction	Angle-averaged	Angle-averaged	Direction-dependent
Hardware	CPU + GPU	GPU only	CPU + GPU
Output file	spectrum.csv	spectrum_virtual.csv	spectrum_rotation.csv

### 4.2 Real Packets

The **real packet** method is the standard Monte Carlo approach. It is the simplest, most robust, and provides the estimators  $(j, \bar{\nu})$  needed for convergence.

#### 4.2.1 Principle

Each energy packet is launched from the photosphere, undergoes a random walk through the ejecta (line scattering, electron scattering, boundary crossings), and eventually either escapes through the outer boundary or is reabsorbed at the inner boundary. The emergent spectrum is constructed by *binning* the escaped packets:

$$L_\lambda(\lambda_k) = \frac{1}{\Delta\lambda} \sum_{\substack{p \in \text{escaped} \\ \lambda_p \in [\lambda_k, \lambda_k + \Delta\lambda)}} \varepsilon_p \quad (4.1)$$

where  $\varepsilon_p$  is the packet energy and  $\lambda_p = c/\nu_p$  is its escape wavelength.

### 4.2.2 Algorithm

#### Algorithm: Real Packet Spectrum

1. Launch  $N$  packets from  $r = R_{\text{inner}}$  with blackbody frequency,  $\mu = \sqrt{\xi}$
2. For each packet: trace through ejecta until escape or reabsorption (see §3.3)
3. Accumulate  $j$  and  $\bar{\nu}$  estimators at each step (for convergence)
4. For escaped packets: store  $(\nu_{\text{lab}}, \varepsilon)$  and bin into spectrum
5. Compute  $L_{\text{emitted}} = \sum_{\text{escaped}} \varepsilon_p$  for  $T_{\text{inner}}$  update

### 4.2.3 Strengths and Limitations

- + Self-consistent: the same packets drive both the spectrum *and* the convergence loop
- + Captures all non-linear transport effects (multiple scatterings, fluorescence)
- + Available on both CPU and GPU
- High Monte Carlo noise in wavelength bins with few packets (deep absorption troughs)
- Angle-averaged: no directional information preserved

## 4.3 Virtual Packets

The **virtual packet** method implements a formal-integral ray-tracing approach inspired by TARDIS [2]. At every interaction point of a real packet,  $N_v$  virtual packets are emitted in random directions and passively traced through the remaining ejecta to compute their escape probability.

### 4.3.1 Physical Motivation

Consider a real packet that has just undergone a line scattering event at position  $(r, \hat{n})$  in shell  $s$ , emitting a photon at comoving frequency  $\nu_{\text{cmf}}$ . The question is: what fraction of this radiation escapes the ejecta along a given direction? Rather than relying on the single random walk of the real packet, virtual packets evaluate this explicitly by integrating the optical depth along rays:

$$P_{\text{esc}}(\hat{n}) = \exp\left(-\int_0^\infty \kappa(s) ds\right) = \exp(-\tau_{\text{total}}) \quad (4.2)$$

The key insight is that this integral can be evaluated *deterministically* (no random interactions), making it much cheaper per ray than full Monte Carlo transport.

### 4.3.2 The $p$ - $z$ Coordinate System

Virtual packets are traced in the  $(p, z)$  coordinate system, where  $p$  is the **impact parameter** (perpendicular distance from the ray to the center) and  $z$  is the coordinate along the ray:

$$p = r \sqrt{1 - \mu_v^2} \quad (\text{constant along the ray}) \quad (4.3)$$

$$z = r \mu_v \quad (\text{varies as the packet moves}) \quad (4.4)$$

The radius at any point along the ray is  $r(z) = \sqrt{p^2 + z^2}$ . The Doppler factor depends only on  $z$ :

$$1 - \frac{v_z}{c} = 1 - \frac{z}{c t_{\text{exp}}} \quad (4.5)$$

This means the comoving-frame frequency is:

$$\nu_{\text{cmf}}(z) = \nu_{\text{lab}} \left( 1 - \frac{z}{c t_{\text{exp}}} \right) \quad (4.6)$$

which is linear in  $z$  — exactly as in the Sobolev sweep for real packets.

**Definition 4.1 —  $p$ - $z$  Geometry.** For a ray emitted at  $(r, \mu_v)$ :

- $p^2 = r^2(1 - \mu_v^2)$  — the impact parameter squared
- Shell  $s$  boundary at  $z = \pm \sqrt{r_s^2 - p^2}$  (inner/outer boundaries)
- Turning point (closest approach):  $z = 0, r_{\text{min}} = p$
- If  $p < R_{\text{inner}}$  and  $\mu_v < 0$ : ray hits the photosphere (discarded)

### 4.3.3 Algorithm

#### Algorithm: Virtual Packet Tracing

At each interaction point  $(r, \text{shell\_id}, \nu_{\text{cmf,emit}}, \varepsilon)$ :

**For**  $i = 1$  to  $N_v$  (default:  $N_v = 10$ ):

1. **Draw direction:**  $\mu_v \sim U(-1, +1)$
2. **Compute:**  $p^2 = r^2(1 - \mu_v^2)$ ,  $z_0 = r \mu_v$
3. **Lab frequency:**  $\nu_{\text{lab}} = \nu_{\text{cmf,emit}} / (1 - z_0 / (c t_{\text{exp}}))$
4. **Check:** if  $p < R_{\text{inner}}$  and  $\mu_v < 0 \rightarrow$  skip (hits photosphere)
5. **Phase 1** (inward,  $\mu_v < 0$ ): trace from  $z_0$  toward  $z = 0$ 
  - For each shell crossed: accumulate  $\tau_{\text{line}}$  (Sobolev sweep) +  $\tau_e = n_e \sigma_T \Delta z$
  - If turning point reached ( $p \geq r_{\text{inner,shell}}$ ): reverse to Phase 2
6. **Phase 2** (outward): trace from turning point to outer boundary
  - For each shell crossed: accumulate  $\tau_{\text{line}} + \tau_e$
7. **Escape:**  $P_{\text{esc}} = e^{-\tau_{\text{total}}}$  (if  $\tau > 50$ , skip)
8. **Bin:**  $L_\lambda += \varepsilon \cdot L_{\text{inner}} \cdot P_{\text{esc}} / (\Delta \lambda \cdot N_v)$

### 4.3.4 Optical Depth Accumulation

Within each shell  $s$ , the virtual packet sweeps from  $z_{\text{entry}}$  to  $z_{\text{exit}}$ . The corresponding CMF frequency range is:

$$\nu_{\text{high}} = \nu_{\text{lab}} \left( 1 - \frac{z_{\text{entry}}}{c t_{\text{exp}}} \right) \quad (4.7)$$

$$\nu_{\text{low}} = \nu_{\text{lab}} \left( 1 - \frac{z_{\text{exit}}}{c t_{\text{exp}}} \right) \quad (4.8)$$

All lines with  $\nu_{\text{low}} \leq \nu_{\text{line}} \leq \nu_{\text{high}}$  contribute their Sobolev optical depth:

$$\tau_{\text{lines}} = \sum_{\nu_{\text{low}} \leq \nu_j \leq \nu_{\text{high}}} \tau_{\text{Sob},j}(s) \quad (4.9)$$

The electron scattering contribution is:

$$\tau_e = n_e(s) \sigma_T |z_{\text{exit}} - z_{\text{entry}}| \quad (4.10)$$

The total optical depth is the sum over all shells traversed:

$$\tau_{\text{total}} = \sum_{\text{shells}} (\tau_{\text{lines}} + \tau_e) \quad (4.11)$$

### 4.3.5 Emission Points

Virtual packets are emitted at three types of interaction events:

1. **Photosphere**: when the real packet is first launched from  $R_{\text{inner}}$
2. **Line scattering**: after the real packet scatters in a line (resonant, downbranch, or macro-atom)
3. **Electron scattering**: after a Thomson scatter event

At each event,  $N_v = 10$  virtual packets are emitted, sampling 10 random directions. This means a real packet that undergoes 50 interactions generates  $\sim 500$  virtual rays.

### 4.3.6 Strengths and Limitations

- + Dramatically lower noise than real packets (many more rays sample each spectral bin)
- + No Monte Carlo branching: deterministic ray tracing with cumulative  $\tau$
- + Each virtual packet is independent — ideal for GPU parallelism
  - Computationally expensive:  $N_v = 10$  virtual rays per interaction
  - Assumes Sobolev approximation for the formal integral (no partial redistribution)
  - GPU only in LUMINA (requires `atomicAdd` for spectrum accumulation)
  - Does *not* contribute to  $j$  or  $\bar{\nu}$  estimators (transport convergence uses real packets only)

**Remark 4.1 — TARDIS Virtual Packets.** TARDIS uses  $N_v \approx 10$ –40 virtual packets per interaction. LUMINA defaults to  $N_v = 10$ , balancing noise reduction against GPU compute cost. The optimal value depends on the number of real packets: for  $N_{\text{real}} \gtrsim 10^6$ , real packet noise is already low enough that  $N_v = 10$  suffices.

## 4.4 Rotation Packets

The **rotation packet** method applies Doppler weighting to escaped real packets to compute the spectrum as seen by an observer at a specific viewing angle. This enables modeling of asymmetric or direction-dependent spectral features.

### 4.4.1 Physical Motivation

In the real-packet method, escaped packets leave the ejecta in random directions. The resulting spectrum is *angle-averaged*: it represents the mean emission over all solid angles. A real observer, however, sees the supernova from a single direction. The Doppler effect means that material moving toward the observer is blueshifted, while receding material is redshifted, producing a direction-dependent spectrum.

In spherical symmetry, the observed spectrum depends on a single parameter: the observer’s direction cosine  $\mu_{\text{obs}}$  relative to each escaping packet’s velocity vector. The rotation method corrects for this by weighting each escaped packet by a Doppler factor.

### 4.4.2 Doppler Weighting

For a packet escaping at radius  $r$  with direction  $\mu$  (relative to the radial direction), the velocity at the escape point is  $v = r/t_{\text{exp}}$  and  $\beta = v/c$ . Two Doppler factors are relevant:

$$D_{\text{pkt}} = 1 - \beta \mu_{\text{pkt}} \quad (\text{packet's actual escape frame}) \quad (4.12)$$

$$D_{\text{obs}} = 1 - \beta \mu_{\text{obs}} \quad (\text{observer's viewing direction}) \quad (4.13)$$

The specific luminosity transforms between frames as:

$$L_{\lambda, \text{obs}} = \left( \frac{D_{\text{obs}}}{D_{\text{pkt}}} \right)^2 L_{\lambda, \text{pkt}} \quad (4.14)$$

The quadratic power arises because the spectral flux density transforms as  $F_\nu \propto D^2$  under Lorentz boosting (one power for the frequency shift, one for the solid angle transformation).

### 4.4.3 Face-On Observer

LUMINA defaults to a **face-on observer** ( $\mu_{\text{obs}} = 1$ ), looking directly along the radial direction. In this case:

$$D_{\text{obs}} = 1 - \beta, \quad w = \left( \frac{1 - \beta}{1 - \beta \mu_{\text{pkt}}} \right)^2 \quad (4.15)$$

The weighting factor  $w$  has the following behavior:

- $w > 1$ : packets that escaped *sideways* ( $\mu_{\text{pkt}} \ll 1$ ) are *boosted* — the face-on observer would see them more strongly
- $w < 1$ : packets that escaped nearly radially ( $\mu_{\text{pkt}} \approx 1$ ) are *suppressed* — they were already heading toward the observer
- $w = 1$ : when  $\mu_{\text{pkt}} = 1$  (escape direction equals observer direction)

In practice, the mean weight  $\langle w \rangle \approx 1.0$ , confirming energy conservation.

### 4.4.4 Algorithm

#### Algorithm: Rotation Packet Spectrum

1. During real-packet transport: for each escaped packet, store  $(r, \mu)$  at escape
2. After transport completes, post-process all escaped packets:
  - Compute  $\beta = r/(ct_{\text{exp}})$
  - Compute  $D_{\text{pkt}} = 1 - \beta \mu_{\text{pkt}}$  and  $D_{\text{obs}} = 1 - \beta$
  - Weight:  $w = (D_{\text{obs}}/D_{\text{pkt}})^2$
3. Bin into spectrum:  $L_\lambda += \varepsilon \cdot L_{\text{inner}} \cdot w / \Delta\lambda$

### 4.4.5 Strengths and Limitations

- + Essentially free: post-processing with no additional transport
- + Provides observer-direction-dependent spectrum from a single simulation
- + Available on both CPU and GPU
- + Can be computed for arbitrary  $\mu_{\text{obs}}$  without re-running
- Same Monte Carlo noise as real packets (no extra sampling)

- In 1D spherical symmetry, the spectrum is isotropic; rotation weighting primarily affects non-radial escape patterns
- Does not capture true viewing-angle effects from multi-dimensional structure

## 4.5 Performance Comparison

Table 4.2: Runtime comparison for 200K packets, 20 iterations (NVIDIA RTX 5000 Ada).

Mode	Time per iter	Overhead	Total (20 iter)
Real only	0.73 s	—	14.6 s
Real + Rotation	0.73 s	< 0.01 s	14.7 s
Real + Virtual ( $N_v = 10$ )	7.3 s	+6.6 s (+900%)	146 s
Real + Virtual + Rotation	7.3 s	+6.6 s (+900%)	146 s

**Important:** Virtual packets are the dominant performance cost when enabled. The  $\sim 10\times$  slowdown comes from tracing  $N_v = 10$  virtual rays per interaction, each performing a binary search + sweep through the full line list. For production runs where spectrum quality is paramount, virtual packets are recommended. For parameter fitting where many models must be evaluated quickly, real-only or real+rotation mode is preferred.

## 4.6 When to Use Each Method

Table 4.3: Recommended spectrum modes for different use cases.

Use Case	Mode	Rationale
Convergence diagnostics	Real only	Fastest; estimators unaffected
Parameter search / SBI	Real + Rotation	Fast; observer-frame spectrum
Publication spectra	Real + Virtual	Lowest noise; best features
Full analysis	All three	Compare all methods

### 4.6.1 Command-Line Usage

Listing 4.1: Spectrum mode selection

```

1 # CPU: real only (default)
2 ./lumina tardis_reference 200000 20
3
4 # CPU: real + rotation
5 ./lumina tardis_reference 200000 20 rotation
6
7 # GPU: real + virtual (formal integral)
8 ./lumina_cuda atomic/kurucz.h5 200000 output.csv virtual
9
10 # GPU: real + rotation
11 ./lumina_cuda atomic/kurucz.h5 200000 output.csv rotation
12

```

```
13 # GPU: all three methods
14 ./lumina_cuda atomic/kurucz.h5 200000 output.csv all
```







## Code Architecture



## 5.1 File Structure

LUMINA-SN consists of the following source files:

Table 5.1: Source files and their roles.

File	Lines	Purpose
lumina.h	378	Master header: all structures, constants, prototypes
lumina_transport.c	515	CPU transport kernel (real + rotation)
lumina_plasma.c	524	Plasma solver & convergence
lumina_atomic.c	700	Atomic data loading (HDF5, CSV, NPY)
lumina_main.c	466	Main driver & iteration loop
lumina_cuda.cu	1353	CUDA GPU kernel (real + virtual + rotation)
<b>Total</b>	<b>3936</b>	

## 5.2 Build System

Listing 5.1: Building LUMINA-SN

```
1 # CPU build (serial)
2 make
3
4 # CPU build with OpenMP parallelism
5 make OMP=1
6
7 # CUDA GPU build
8 make cuda
```

**Important:** `make clean` deletes CSV output files. Always save important spectral outputs before rebuilding.

Table 5.2: Compiler flags.

Target	Compiler	Flags
CPU	gcc	-O2 -Wall -Wextra -std=c11 -lm
CPU+OMP	gcc	adds -fopenmp
GPU	nvcc	-O2 -arch=sm_89 -std=c++14

### 5.3 Dependencies

- **HDF5** (optional): For loading atomic data from `kurucz_cd23_chianti_H_He.h5`
- **CUDA Toolkit  $\geq 12.0$** : For GPU builds (tested with CUDA 13.0, sm\_89)
- **OpenMP**: For CPU parallelism (optional)
- **Standard C library**: `math.h`, `stdio.h`, `stdlib.h`, `string.h`

### 5.4 Execution

Listing 5.2: Running LUMINA-SN

```

1  # CPU: reference model with 200K packets, 20 iterations
2  ./lumina tardis_reference 200000 20
3
4  # CUDA: same with GPU acceleration
5  ./lumina_cuda atomic/kurucz_cd23_chianti_H_He.h5 200000 output.
    csv

```

# 6

## Data Structures

### 6.1 The RPacket Structure

The fundamental unit of the Monte Carlo simulation:

Listing 6.1: RPacket — photon energy packet

```
1  typedef struct {
2      double r;                // radial position [cm]
3      double mu;               // cos(theta) direction
4      double nu;               // lab-frame frequency [Hz]
5      double energy;           // packet energy [erg]
6      int    current_shell_id; // shell index [0..n_shells-1]
7      int    next_line_id;     // Sobolev sweep bookmark
8      PacketStatus status;     // IN_PROCESS / EMITTED /
                                // REABSORBED
9      int    index;            // packet ID (for RNG seeding)
10 } RPacket;
```

### 6.2 Geometry

The 1D spherically symmetric ejecta model:

Listing 6.2: Geometry — shell structure

```
1  typedef struct {
2      int    n_shells;          // number of shells (default: 30)
3      double *r_inner;          // [n_shells] inner radii [cm]
4      double *r_outer;          // [n_shells] outer radii [cm]
5      double *v_inner;          // [n_shells] inner velocities [cm/
                                // s]
6      double *v_outer;          // [n_shells] outer velocities [cm/
                                // s]
7      double time_explosion;     // t_exp [seconds]
8  } Geometry;
```

The radii are derived from velocities via homologous expansion:  $r = v \times t_{\text{exp}}$ .

## 6.3 Opacity State

Pre-computed opacity data used during transport:

Listing 6.3: OpacityState — line and continuum opacities

```

1  typedef struct {
2      int n_lines, n_shells;
3      double *line_list_nu;          // [n_lines] rest frequencies,
        descending
4      double *tau_sobolev;           // [n_lines * n_shells]
5      double *electron_density;      // [n_shells] n_e [cm^-3]
6      double *t_electrons;           // [n_shells] T_e [K]
7
8      // Macro-atom transition data
9      int n_macro_levels, n_macro_transitions;
10     int *macro_block_references;    // [n_levels+1]
11     int *transition_type;           // [n_transitions]
12     int *destination_level_id;      // [n_transitions]
13     int *transition_line_id;        // [n_transitions]
14     double *transition_probabilities; // [n_transitions *
        n_shells]
15     int *line2macro_level_upper;    // [n_lines]
16 } OpacityState;
```

**Remark 6.1 — Line List Ordering.** Lines are sorted in *descending* frequency order. As a packet’s CMF frequency decreases while traversing a shell, it encounters lines from high to low frequency. This ordering enables efficient forward-only scanning.

## 6.4 Plasma State

Thermodynamic state updated each iteration:

Listing 6.4: PlasmaState — radiation field quantities

```

1  typedef struct {
2      int n_shells;
3      double *W;                    // [n_shells] dilution factor
4      double *T_rad;                // [n_shells] radiation temperature
        [K]
5      double *rho;                  // [n_shells] mass density [g/cm^3]
6      double *n_electron;           // [n_shells] electron density [cm
        ^-3]
7      double T_e_T_rad_ratio;       // default: 0.9
8  } PlasmaState;
```

## 6.5 Atomic Data

Comprehensive atomic physics database for the plasma solver:

Listing 6.5: AtomicData — atomic physics for Saha–Boltzmann

```

1  typedef struct {
2      // Per-line data
3      int      *line_atomic_number;    // [n_lines] Z
4      int      *line_ion_number;      // [n_lines] ionization stage
5      double   *line_f_lu;            // [n_lines] oscillator
        strength
6      double   *line_wavelength_cm;    // [n_lines] rest wavelength
7
8      // Energy levels
9      int      n_levels;
10     double   *level_energy_eV;       // [n_levels]
11     int      *level_g;               // [n_levels] statistical
        weight
12     int      *level_metastable;      // [n_levels] 0 or 1
13
14     // Ionization energies
15     int      n_ionization;
16     double   *ioniz_energy_eV;       // [n_ionization]
17
18     // Zeta correction factors (dilute non-LTE)
19     double   *zeta_data;             // [n_zeta_ions *
        n_zeta_temps]
20
21     // Abundances
22     double   *abundances;            // [n_elements * n_shells]
23
24     // Computed quantities (updated each iteration)
25     double   *ion_number_density;    // [n_ion_pops * n_shells]
26     double   *partition_functions;   // [n_ion_pops * n_shells]
27 } AtomicData;

```

## 6.6 Monte Carlo Estimators

Accumulated during transport, used to update the radiation field:

Listing 6.6: Estimators — radiation field accumulators

```

1  typedef struct {
2      double   *j_estimator;          // [n_shells] integral of E*ds
3      double   *nu_bar_estimator;     // [n_shells] integral of E*nu*
        ds
4      double   *j_blue_estimator;     // [n_lines * n_shells] (CPU
        only)
5      double   *Edotlu_estimator;     // [n_lines * n_shells] (CPU
        only)
6  } Estimators;

```

**Remark 6.2 — GPU Limitation.** The `j_blue` and `Edotlu` estimators are *not computed* on the GPU because they require  $n_{\text{lines}} \times n_{\text{shells}} \approx 4$  million atomic additions per iteration — prohibitively expensive for `atomicAdd`.





# 7

## Transport Engine

### 7.1 Overview

The transport engine (`lumina_transport.c`, 515 lines) propagates  $r$ -packets through the ejecta. It is the most performance-critical component.

### 7.2 The `trace_packet` Function

This function computes the next interaction event for a packet:

1. Compute distance to shell boundaries ( $d_{\text{boundary}}$ )
2. Scan the Sobolev line list for resonances ( $d_{\text{line}}$ , accumulated  $\tau$ )
3. Compute distance to electron scattering ( $d_e = \tau_{\text{event}}/(n_e \sigma_T)$ )
4. Return the minimum distance and interaction type

### 7.3 Sobolev Line Sweep

The sweep algorithm processes lines in descending frequency order:

Listing 7.1: Sobolev sweep (simplified)

```
1  double tau_trace_combined = 0.0;
2  for (int j = pkt->next_line_id; j < n_lines; j++) {
3      double nu_line = line_list_nu[j];
4      double d_line = compute_d_line(pkt, nu_line);
5
6      if (d_line < 0 || d_line > d_boundary) break;
7
8      double tau_line = tau_sobolev[j * n_shells + shell_id];
9      tau_trace_combined += tau_line;
10
11     if (tau_trace_combined > tau_event) {
12         // Line interaction!
13         *interaction_type = INTERACTION_LINE;
14         *d_min = d_line;
15         pkt->next_line_id = j;
16         return;
17     }
18 }
```

## 7.4 Thomson Scattering

Elastic scattering with free electrons:

1. Transform packet energy/frequency to comoving frame at current angle
2. Sample new direction:  $\mu_{\text{new}} \sim U(-1, +1)$  (isotropic in CMF)
3. Transform back to lab frame with new angle
4. Energy is conserved in the comoving frame

$$\varepsilon_{\text{lab,new}} = \varepsilon_{\text{cmf}} \times \frac{1}{1 - \mu_{\text{new}} \cdot v/c} \quad (7.1)$$

## 7.5 Boundary Crossing

When a packet crosses a shell boundary:

1. Update shell index:  $i_{\text{shell}} \leftarrow i_{\text{shell}} + \delta$  where  $\delta = +1$  (outward) or  $-1$  (inward)
2. Nudge position by  $\epsilon = 10^{-10} \times \Delta r_{\text{shell}}$  into the new shell
3. Check for escape ( $i > n_{\text{shells}} - 1$ ) or reabsorption ( $i < 0$ , inward-moving)

**Important:** The position nudge is critical. Without it, the packet lands exactly on the boundary, and the next distance calculation returns  $d = 0$ , causing an infinite loop (Task #024).

## 8.1 Overview

The plasma solver (`lumina_plasma.c`, 524 lines) computes the thermodynamic state of the ejecta for each iteration. It implements the TARDIS-compatible nebular approximation [6].

## 8.2 Step 1: Partition Functions

The partition function for ion ( $Z$ , stage) in shell  $s$  is:

$$\mathcal{Z}(Z, \text{stage}, s) = \underbrace{\sum_{i \in \text{meta}} g_i e^{-E_i/k_B T_{\text{rad}}(s)}}_{\mathcal{Z}_{\text{meta}}} + W(s) \cdot \underbrace{\sum_{i \in \text{non-meta}} g_i e^{-E_i/k_B T_{\text{rad}}(s)}}_{\mathcal{Z}_{\text{non}}} \quad (8.1)$$

**Important:** The Boltzmann factors use  $T_{\text{rad}}$  for *all* levels (both metastable and non-metastable). Earlier code incorrectly used  $T_e$  for metastable levels. The dilution factor  $W$  suppresses the non-metastable contribution at large distances from the photosphere.

## 8.3 Step 2: Electron Density

Computed iteratively with TARDIS-style damping:

### Algorithm: Electron Density Iteration

1. Start with initial guess  $n_e^{(0)}$
2. For each element: compute ionization ratios using nebular Saha (Eq. 8.2)
3. Normalize ion populations to element abundance
4. Compute  $n_e^{(\text{calc})} = \sum_{\text{ions}} \text{stage} \times n_{\text{ion}}$
5. Damped update:  $n_e^{(k+1)} = 0.5 \times n_e^{(\text{calc})} + 0.5 \times n_e^{(k)}$
6. Convergence:  $|n_e^{(k+1)} - n_e^{(k)}|/n_e^{(k)} < 0.05$

### 8.4 Step 3: Nebular Saha Ionization

The ionization ratio between consecutive stages is:

$$\boxed{\frac{n_{i+1}}{n_i} = \frac{\Phi_{\text{neb}}}{n_e}} \quad (8.2)$$

where the nebular ionization coefficient  $\Phi_{\text{neb}}$  is:

$$\Phi_{\text{neb}} = \Phi_{\text{LTE}} \times W \times [\zeta \cdot \delta + W \cdot (1 - \zeta)] \times \sqrt{\frac{T_e}{T_{\text{rad}}}} \quad (8.3)$$

$$\Phi_{\text{LTE}} = \frac{Z_{i+1}}{Z_i} \times 2 \times g_e \times e^{-\chi/k_B T_{\text{rad}}} \quad (8.4)$$

$$g_e = \left( \frac{2\pi m_e k_B T_{\text{rad}}}{h^2} \right)^{3/2} \quad (8.5)$$

$$\delta = \frac{T_e}{T_{\text{rad}}} \exp \left[ \chi \left( \frac{1}{k_B T_{\text{rad}}} - \frac{1}{k_B T_e} \right) \right] \quad (8.6)$$

Here  $\chi$  is the ionization energy, and  $\zeta$  is a non-LTE correction factor interpolated from tabulated values.

### 8.5 Step 4: $\tau_{\text{Sobolev}}$ Update

With ion populations known,  $\tau_{\text{Sob}}$  is recomputed for each line and shell using Eq. (2.2). The level populations follow the Boltzmann distribution within each ion:

$$n_{l,\text{meta}} = \frac{g_l}{Z} n_{\text{ion}} e^{-E_l/k_B T_{\text{rad}}}, \quad n_{l,\text{non}} = W \times \frac{g_l}{Z} n_{\text{ion}} e^{-E_l/k_B T_{\text{rad}}} \quad (8.7)$$

### 8.6 Step 5: Radiation Field Update

From the MC estimators  $j$  and  $\bar{\nu}$  (Eqs. 3.10–3.11):

$$T_{\text{rad,est}}(s) = 1.2523 \times 10^{-11} \times \frac{\bar{\nu}(s)}{j(s)} \quad [\text{K}] \quad (8.8)$$

$$W_{\text{est}}(s) = \frac{j(s)}{4 \sigma_{\text{SB}} T_{\text{rad}}^4(s) \Delta t V(s)} \quad (8.9)$$

All quantities are damped:

$$X_{\text{new}} = X_{\text{old}} + 0.5 \times (X_{\text{est}} - X_{\text{old}}) \quad (8.10)$$

## 9.1 Data Sources

LUMINA uses the TARDIS reference atomic dataset, originally from Kurucz CD23 and CHIANTI:

Table 9.1: Atomic data files.

File	Format	Contents
line_list.csv	CSV	$\nu$ , $Z$ , ion, $f_{lu}$ , $\lambda$ per line
levels.csv	CSV	$E$ (eV), $g$ , metastable flag per level
ionization_energies.csv	CSV	$\chi$ per ion
tau_sobolev.npy	NPY	$\tau_{\text{Sob}}$ reference values
transition_probabilities.npy	NPY	Macro-atom transition probs
macro_atom_data.csv	CSV	Transition types, destinations
zeta_data.npy	NPY	Non-LTE correction factors
abundances.csv	CSV	Mass fractions per shell

## 9.2 The NPY Format Reader

LUMINA includes a custom NPY reader (no NumPy dependency in C):

1. Read 6-byte magic: `\x93NUMPY`
2. Read version (1 or 2) and header length
3. Parse Python dict header for `shape`, `dtype`, `fortran_order`
4. Read raw binary data
5. Transpose if Fortran-ordered

## 9.3 The CSV Parser

**Important:** The `macro_atom_data.csv` header starts with an unnamed index column: `“,atomic_number,...”`. The standard `strtok()` function skips leading delim-

iters, causing a column offset of  $-1$ . LUMINA uses a manual field-by-field parser that handles empty fields correctly.

## 10.1 Design Philosophy

The GPU implementation maps **one CUDA thread per packet**. Each thread independently propagates its packet through the ejecta, requiring no inter-thread communication except for atomic estimator updates.

## 10.2 Memory Layout

Table 10.1: GPU memory allocation.

Data	Access	Size (200K pkts)
Line frequencies	Read-only	$n_{\text{lines}} \times 8 \text{ B}$
$\tau_{\text{Sob}}$	Read-only	$n_{\text{lines}} \times n_{\text{shells}} \times 8 \text{ B}$
Transition probs	Read-only	$n_{\text{trans}} \times n_{\text{shells}} \times 8 \text{ B}$
Shell geometry	Read-only	$n_{\text{shells}} \times 4 \times 8 \text{ B}$
RNG states	Read/write	$N_{\text{pkt}} \times 4 \times 8 \text{ B}$
$j, \bar{\nu}$ estimators	Atomic write	$n_{\text{shells}} \times 2 \times 8 \text{ B}$
Output arrays	Write-only	$N_{\text{pkt}} \times 3 \times 8 \text{ B}$

Total GPU memory: approximately 2 GB for a typical run.

## 10.3 Kernel Launch Configuration

Listing 10.1: Kernel launch

```

1  int threads_per_block = 256;
2  int blocks = (n_packets + threads_per_block - 1)
3              / threads_per_block;
4  // Max blocks: 131072 (was 1024 -- critical bug #13)
5  transport_kernel<<<blocks, threads_per_block>>>(...);

```

**Important:** The original code had `CUDA_MAX_BLOCKS = 1024`, limiting execution to 262K packets regardless of  $N_{\text{packets}}$ . For 2M packets, only 13% executed, causing  $j_{\text{estimator}}$  to be  $76\times$  too low. This was the single largest GPU bug (Task #13).

## 10.4 Random Number Generation

Each thread uses an independent **xoshiro256\*\*** generator with 256 bits of state ( $4 \times \text{uint64}$ ). Seeds are derived from the packet index via SplitMix64:

Listing 10.2: Per-thread RNG initialization

```
1  __device__ void init_rng(uint64_t *state, uint64_t seed) {
2      // SplitMix64 to expand seed into 4 state words
3      state[0] = splitmix64(&seed);
4      state[1] = splitmix64(&seed);
5      state[2] = splitmix64(&seed);
6      state[3] = splitmix64(&seed);
7  }
```

## 10.5 Atomic Estimator Updates

The  $j$  and  $\bar{\nu}$  estimators are updated using CUDA `atomicAdd`:

Listing 10.3: Estimator accumulation on GPU

```
1  __device__ void update_estimators(
2      double *d_j_est, double *d_nu_bar_est,
3      int shell_id, double comov_energy,
4      double comov_nu, double distance)
5  {
6      atomicAdd(&d_j_est[shell_id],
7                comov_energy * distance);
8      atomicAdd(&d_nu_bar_est[shell_id],
9                comov_energy * distance * comov_nu);
10 }
```

Since there are only  $n_{\text{shells}} = 30$  accumulation targets, contention is manageable.

## 10.6 Performance

Table 10.2: CPU vs GPU performance (NVIDIA RTX 5000 Ada, sm\_89).

$N_{\text{packets}}$	CPU (1 core)	CPU (OMP64)	GPU	Speedup
20,000	0.7 s	0.1 s	0.08 s	9×
200,000	7.2 s	1.1 s	0.73 s	10×
2,000,000	72 s	11 s	7.3 s	10×
20,000,000	720 s	110 s	73 s	10×



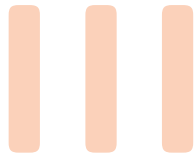
**Remark 10.1 — Statistical Accuracy.** GPU and CPU produce statistically identical results. At 200K packets:  $W$  error 1.06%,  $T_{\text{rad}}$  error 0.58% (relative to TARDIS reference). The scaling follows  $\sigma \propto N^{-0.35}$  to  $N^{-0.40}$ , close to Poisson ( $N^{-0.5}$ ).

## 10.7 Resolved GPU Bugs

Four critical bugs were identified and fixed during development:

1. **MAX\_BLOCKS = 1024:** Only 262K threads could launch. Fixed to 131072.
2. **Shared memory race:** `__shared__ ShellCache` shared by all 256 threads, but only thread 0 loaded data. Fixed: use L1-cached global memory.
3. **Counter accumulation:** Escape/reabsorb counters not reset between iterations. Fixed: explicit reset in `cuda_reset_estimators()`.
4. **Boundary sticking:** Packets land exactly on shell boundaries due to floating-point precision. Fixed: explicit nudge by  $10^{-10} \times \Delta r$ .





## Usage & Applications



# 11

## Installation & Quick Start

### 11.1 Prerequisites

Table 11.1: System requirements.

Component	Requirement
C Compiler	$\text{GCC} \geq 9.0$ (C11 support)
CUDA (optional)	Toolkit $\geq 12.0$ , compute capability $\geq 7.0$
HDF5 (optional)	<code>libhdf5-dev</code> for atomic data loading
Memory	$\geq 4$ GB RAM (CPU), $\geq 4$ GB VRAM (GPU)

### 11.2 Step-by-Step Setup

Listing 11.1: Complete setup procedure

```
1  # Clone the repository
2  git clone git@github.com:kjhan0606/lumina-sn.git
3  cd lumina-sn
4
5  # Build CPU version
6  make
7
8  # (Optional) Build GPU version
9  make cuda
10
11 # Verify with a quick test (1000 packets, 5 iterations)
12 ./lumina tardis_reference 1000 5
13
14 # Production run (200K packets, 20 iterations)
15 ./lumina tardis_reference 200000 20
16
17 # Check output
18 head lumina_spectrum.csv
```

### 11.3 Input Directory Structure

LUMINA expects a reference data directory (default: `data/tardis_reference/`) containing:

Listing 11.2: Required input files

```

1 data/tardis_reference/
2   config.json           # Simulation parameters
3   geometry.csv          # Shell radii and velocities
4   density.csv           # Mass density per shell
5   abundances.csv        # Element mass fractions
6   electron_densities.csv # Initial electron densities
7   plasma_state.csv      # Initial W, T_rad per shell
8   line_list.csv         # Atomic line data
9   tau_sobolev.npy       # Reference optical depths
10  transition_probabilities.npy
11  macro_atom_data.csv
12  macro_atom_references.csv
13  line2macro_level_upper.npy
14  levels.csv            # Energy levels
15  ionization_energies.csv
16  zeta_ions.csv
17  zeta_temps.csv
18  zeta_data.npy
19  atom_masses.csv

```

### 11.4 Output Files

Table 11.2: Output files produced by LUMINA.

File	Contents
<code>lumina_spectrum.csv</code>	$\lambda$ (Å) vs $L_\lambda$ (erg/s/cm)
<code>lumina_plasma_state.csv</code>	Final $W$ , $T_{\text{rad}}$ per shell
<code>lumina_tau_validation.csv</code>	$\tau_{\text{Sob}}$ at shell 0 (debug)

# 12

## The Ejecta Model

### 12.1 Three-Zone Composition

LUMINA uses a three-zone abundance structure motivated by SN Ia nucleosynthesis:

Table 12.1: Default three-zone composition model.

Zone	Fe	Si	S	Ca	Co	Ni	C	O
Core ( $v < v_{\text{core}}$ )	<i>free</i>	0.05	0.05	0.03	0.05	<i>free</i>	0.02	filler
Wall ( $v_{\text{core}} - v_{\text{wall}}$ )	<i>free</i>	<i>free</i>	0.05	0.03	0.05	<i>free</i>	0.02	filler
Outer ( $v > v_{\text{wall}}$ )	<i>free</i>	0.02	0.02	0.01	0.05	<i>free</i>	0.02	filler

“Filler” means oxygen fills the remaining mass fraction to ensure  $\sum X_i = 1$ .

**Important:** Oxygen is the correct filler element for the outer zone because it has very few optical absorption lines and is essentially transparent. Using Fe/Ni/S as fillers creates massive line blanketing ( $> 10^6$  active lines) that produces an artificial pseudo-photosphere (Task #063).

### 12.2 Broken Power-Law Density

The density profile is a broken power law:

$$\rho(v) = \begin{cases} \rho_0 \left( \frac{v}{v_{\text{inner}}} \right)^{n_{\text{inner}}} & v < v_{\text{break}} \\ \rho_{\text{break}} \left( \frac{v}{v_{\text{break}}} \right)^{n_{\text{outer}}} & v \geq v_{\text{break}} \end{cases} \quad (12.1)$$

where  $\rho_{\text{break}} = \rho_0 (v_{\text{break}}/v_{\text{inner}})^{n_{\text{inner}}}$  ensures continuity at  $v_{\text{break}}$ .

Typical values:  $n_{\text{inner}} \approx -7$ ,  $n_{\text{outer}} \approx -10$ .

### 12.3 Physical Parameter Space

LUMINA-ML (the machine learning emulator companion) uses a 15-dimensional parameter space:

Table 12.2: Full 15D parameter space with ranges.

#	Parameter	Min	Max	Description
1	$\log L$	42.50	43.50	Luminosity [erg/s]
2	$v_{\text{inner}}$	7000	15000	Photosphere velocity [km/s]
3	$\log \rho_0$	-14.0	-12.3	Reference density [g/cm <sup>3</sup> ]
4	$n_{\text{inner}}$	-10	-4	Inner density exponent
5	$T_e/T_{\text{rad}}$	0.7	1.0	Temperature ratio
6	$v_{\text{core}}$	9000	17000	Core/wall boundary [km/s]
7	$v_{\text{wall}}$	12000	24000	Wall/outer boundary [km/s]
8	$X_{\text{Fe,core}}$	0.05	0.85	Core iron abundance
9	$X_{\text{Si,wall}}$	0.05	0.75	Wall silicon abundance
10	$v_{\text{break}}$	10000	22000	Density break velocity [km/s]
11	$n_{\text{outer}}$	-14	-4	Outer density exponent
12	$t_{\text{exp}}$	10	28	Time since explosion [days]
13	$X_{\text{Fe,wall}}$	0.001	0.50	Wall iron contamination
14	$X_{\text{Ni}}$	0.005	0.25	Nickel abundance (all zones)
15	$X_{\text{Fe,outer}}$	0.001	0.15	Outer iron abundance



# 13

## Parameter Fitting

---

### 13.1 Fitting Strategy

LUMINA includes a multi-phase parameter search framework (`scripts/fit_parameter_search.py`) that combines Latin Hypercube Sampling with progressive refinement.

#### 13.1.1 Phase 1: Coarse Exploration

- 200 Latin Hypercube samples across full parameter space
- 20K packets  $\times$  5 iterations (fast,  $\sim 5$  s per model)
- Score by feature-weighted RMS
- Select top-20 candidates

#### 13.1.2 Phase 2: Refinement

- Top-20 candidates re-simulated with 100K packets  $\times$  10 iterations
- Better statistics reduce Monte Carlo noise
- Select top-3

#### 13.1.3 Phase 3: Production

- Top-3 candidates with 500K packets  $\times$  20 iterations
- Highest-fidelity spectra
- Final selection based on composite score

### 13.2 Objective Function

The composite scoring function includes:

$$\text{Score} = \text{RMS}_{\text{spec}} + 0.5 |\Delta d_{\text{Si II}}| + 0.2 |\Delta \log v_{\text{Si II}}| + 0.1 |\Delta \lambda_{\text{min}}| \quad (13.1)$$

where:

- $\text{RMS}_{\text{spec}}$ : Spectral RMS over 5000–8000 Å
- $\Delta d_{\text{Si II}}$ : Si II 6355 absorption depth error
- $\Delta \log v_{\text{Si II}}$ : Si II velocity error (log scale)
- $\Delta \lambda_{\text{min}}$ : Si II trough wavelength error



## Physical Constants & Reference Values

Table 14.1: Physical constants used in LUMINA (CGS).

Symbol	Description	Value
$c$	Speed of light	$2.99792458 \times 10^{10}$ cm/s
$h$	Planck constant	$6.62607015 \times 10^{-27}$ erg·s
$k_B$	Boltzmann constant	$1.380649 \times 10^{-16}$ erg/K
$\sigma_{\text{SB}}$	Stefan–Boltzmann	$5.670374 \times 10^{-5}$ erg/cm <sup>2</sup> /s/K <sup>4</sup>
$\sigma_T$	Thomson cross-section	$6.6525 \times 10^{-25}$ cm <sup>2</sup>
$m_e$	Electron mass	$9.10938 \times 10^{-28}$ g
$e$	Electron charge	$4.80321 \times 10^{-10}$ esu
$\pi e^2/(m_e c)$	Sobolev coefficient	$2.6540 \times 10^{-2}$ cm <sup>2</sup> /s
$T_{\text{rad,const}}$	$T_{\text{rad}}$ estimator constant	$1.2523 \times 10^{-11}$ K·s



## Bibliography

---

- [1] J. E. Bjorkman and K. Wood. “Radiative Equilibrium and Temperature Correction in Monte Carlo Radiation Transfer”. In: *The Astrophysical Journal* 554 (2001), pp. 615–623.
- [2] W. E. Kerzendorf and S. A. Sim. “A spectral synthesis code for rapid modelling of supernovae”. In: *Monthly Notices of the Royal Astronomical Society* 440 (2014), pp. 387–404. DOI: 10.1093/mnras/stu055.
- [3] L. B. Lucy. “Improved Monte Carlo techniques for the spectral synthesis of supernovae”. In: *Astronomy & Astrophysics* 345 (1999), pp. 211–220.
- [4] L. B. Lucy. “Monte Carlo techniques for time-dependent radiative transfer in 3-D supernovae”. In: *Astronomy & Astrophysics* 384 (2002), pp. 725–735.
- [5] L. B. Lucy. “Monte Carlo transition probabilities”. In: *Astronomy & Astrophysics* 403 (2003), pp. 261–275.
- [6] P. A. Mazzali and L. B. Lucy. “The application of Monte Carlo methods to the synthesis of early-time supernovae spectra”. In: *Astronomy & Astrophysics* 279 (1993), pp. 447–456.
- [7] K. Nomoto, F.-K. Thielemann, and K. Yokoi. “Accreting white dwarf models of Type I supernovae. III. Carbon deflagration supernovae”. In: *The Astrophysical Journal* 286 (1984), pp. 644–658.
- [8] M. M. Phillips. “The absolute magnitudes of Type IA supernovae”. In: *The Astrophysical Journal Letters* 413 (1993), pp. L105–L108.



- Atomic Data, 37
  - CSV Parser, 37
  - NPY Format, 37
- Build System, 27
  - Makefile, 27
- Continuum Opacity, 12
- Convergence, 16
- CUDA, 39
  - Atomic Operations, 40
  - Bug Fixes, 41
  - Kernel Launch, 39
  - Memory Layout, 39
  - Performance, 40
  - RNG, 40
- Data Structures, 29
  - AtomicData, 30
  - Estimators, 31
  - Geometry, 29
  - OpacityState, 30
  - PlasmaState, 30
  - RPacket, 29
- Dilution Factor, 12
- Distance Calculations, 14
- Doppler
  - Frame Transformations, 12
- Ejecta
  - Homologous Expansion, 9
- Ejecta Model, 47
  - Density Profile, 47
  - Parameter Space, 47
  - Three-Zone, 47
- Energy Packets, 13
- Estimators, 15
  - Mean Intensity, 16
- Execution, 28
- Input Files, 46
- Installation, 45
- Line Interactions, 15
  - Downbranching, 15
  - Macro-Atom, 15
  - Resonant Scattering, 15
- Macro-Atom, 15
- Monte Carlo Methods, 13
- Output Files, 46
- Packet Initialization, 13
- Parameter Fitting, 49
  - Objective Function, 49
- Physical Constants, 51
- Plasma Solver, 35
  - Electron Density, 35
  - Partition Functions, 35
  - Radiation Field, 36
  - Saha Equation, 36
  - Tau Sobolev, 36
- Propagation Loop, 14
- Radiative Transfer, 11
- Rotation Packets, 20
  - Doppler Weighting, 21
- Sobolev Approximation, 11
- Sobolev Optical Depth, 11
- Sobolev Sweep, 14
- Spectral Features, 9
- Spectrum Synthesis, 17
  - Performance, 22
  - Real Packets, 17
  - Recommendations, 22
  - Rotation Packets, 20
  - Virtual Packets, 18
- Transfer Equation, 11
- Transport Engine, 33
  - Boundary Crossing, 34
  - Sobolev Sweep, 33
  - Thomson Scattering, 34

- trace\_packet, 33
- Type Ia Supernovae, 9
- Virtual Packets, 18
  - Algorithm, 19
  - Optical Depth, 19
  - p-z Coordinates, 18