LUMINA-SN

Monte Carlo Radiative Transfer for Type Ia Supernovae

A Complete Technical Manual

Professor Juhan Kim

February 2026

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 [8–10] and the TARDIS code [5]. Written in C99 with CUDA GPU acceleration.

Version 2.0 — February 2026

Contents

I

Physics & Theory

1 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 ($M_{\mathrm{Ch}} \approx 1.4\,M_\odot$). The explosion completely unbinds the star, leaving no compact remnant, and synthesizes approximately $0.6\,M_\odot$ of radioactive ^{56}Ni [12].

Important: SN Ia are cosmological standard candles: their peak luminosity correlates with their light-curve decline rate (the Phillips relation, [13]), 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**:

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

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:

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

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

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.

2

Radiative Transfer in Expanding Atmospheres

2.1 The Transfer Equation

The specific intensity I_ν along a ray satisfies:

$$\frac{dI_\nu}{ds} = -\kappa_\nu I_\nu + j_\nu \tag{2.1}$$

where s is the path length, κ_ν is the absorption coefficient (opacity), and j_ν is the emissivity.

In a supernova atmosphere, three opacity sources contribute:

1. **Electron scattering** (Thomson): frequency-independent, $\sigma_T = 6.652 \times 10^{-25}$ 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 $dv/dr = 1/t_{\exp}$ 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 $l \to u$ in the Sobolev approximation is:

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

where f_{lu} is the oscillator strength, λ_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:

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

The *escape probability* from a Sobolev line is:

$$\beta_{\text{Sob}} = \frac{1 - e^{-\tau_{\text{Sob}}}}{\tau_{\text{Sob}}} \tag{2.4}$$

2.3 The Dilute Radiation Field

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

$$J_\nu = W\, B_\nu(T_{\rm rad}) \tag{2.5}$$

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

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

Remark 2.1 — Physical Interpretation of W. At $r = R_{\rm phot}$: $W = 0.5$ (hemisphere illuminated). At $r \gg R_{\rm phot}$: $W \approx R_{\rm phot}^2/(4r^2) \to 0$ (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:

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

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

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

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

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 Å) could ionize these ions.
- **Free–free**: Electron densities are very low ($n_e \sim 10^9$ cm^{-3} inner, $\sim 10^6$ cm^{-3} outer), giving $\kappa_{\rm ff} \sim 10^{-20}$ cm^{-1}.

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

3 Monte Carlo Photon Transport

3.1 The Indivisible Energy Packet Formalism

LUMINA follows the Lucy [8, 9] 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
μ	Direction cosine ($\cos\theta$)	–
ν	Lab-frame frequency	Hz
ε	Packet energy	erg
i_{shell}	Current shell index	–

All packets carry the same energy:

$$\varepsilon_{\text{pkt}} = \frac{L_{\text{inner}}\,\Delta t}{N_{\text{packets}}} \tag{3.1}$$

where L_{inner} is the luminosity at the inner boundary and Δt 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_{inner} using the Bjorkman and Wood [1] method:

1. Draw $\xi_0 \sim U(0,1)$
2. Find l_{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:

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

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
       • $d_{\text{e}}$: distance to electron scattering ($\tau = n_e \sigma_T d$)
  3. Select minimum: $d_{\text{min}} = \min(d_{\text{boundary}}, d_{\text{line}}, d_{\text{e}})$
  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, μ) in shell $[r_{\text{in}}, r_{\text{out}}]$:

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

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

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

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

3.4.2 Distance to Electron Scattering

$$d_e = \frac{\tau_{\text{event}}}{n_e\,\sigma_T} \tag{3.5}$$

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 ν_{line}:

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

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:

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

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:

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

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 [9, 10] 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.

$$\hat{J}_\nu = \frac{1}{4\pi\, V\, \Delta t} \sum_{\text{packets}} \varepsilon_{\text{cmf}}\, \Delta s \tag{3.9}$$

In practice, two scalar estimators are accumulated per shell:

$$j = \sum_{\text{packets}} \varepsilon_{\text{cmf}}\, \Delta s \tag{3.10}$$

$$\overline{\nu} = \frac{\sum_{\text{packets}} \varepsilon_{\text{cmf}}\, \nu_{\text{cmf}}\, \Delta s}{\sum_{\text{packets}} \varepsilon_{\text{cmf}}\, \Delta s} \tag{3.11}$$

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

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

$$W = \frac{j}{4\, \sigma_{\text{SB}}\, T_{\text{rad}}^4\, \Delta t\, V} \tag{3.13}$$

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

3.7 Convergence: The Iteration Loop

LUMINA iterates between transport and plasma calculations:

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

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

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

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

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_{inner} also changes the opacity (through ionization), leading to non-linear feedback. The empirical -0.5 accounts for this.

4 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:

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

where ε_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_{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 [5]. 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 ν_{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:

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

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:

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

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

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

$$1 - \frac{v_z}{c} = 1 - \frac{z}{c\,t_{\text{exp}}} \tag{4.5}$$

This means the comoving-frame frequency is:

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

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, μ_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 $\mu_v < 0 \to$ 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: $L_\lambda \mathrel{+}= \varepsilon \cdot L_{\text{inner}} \cdot P_{\text{esc}}/(\Delta\lambda \cdot N_v)$
```

4.3.4 Optical Depth Accumulation

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

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

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

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

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

The electron scattering contribution is:

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

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

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

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_{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 ~ 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 τ
+ 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 μ_{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 μ (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:

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

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

The specific luminosity transforms between frames as:

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

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:

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

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 $\beta = r/(c\,t_{\text{exp}})$
     • 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: $L_\lambda \mathrel{+}= \varepsilon \cdot L_{\text{inner}} \cdot w/\Delta\lambda$
```

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 μ_{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

```
# CPU: real only (default)
./lumina tardis_reference 200000 20

# CPU: real + rotation
./lumina tardis_reference 200000 20 rotation

# GPU: real + virtual (formal integral)
./lumina_cuda atomic/kurucz.h5 200000 output.csv virtual

# GPU: real + rotation
./lumina_cuda atomic/kurucz.h5 200000 output.csv rotation

```

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

5 Non-LTE Rate Equation Solver

5.1 Motivation: Beyond the Nebular Approximation

The nebular approximation (Chapter 8) treats ionization via a modified Saha equation with dilution factor W and radiation temperature T_{rad}, and level populations via Boltzmann at T_{rad}. This is adequate for most photospheric-phase diagnostics but has known limitations:

- **UV spectrum**: Boltzmann populations overestimate excited-level populations for Fe-group ions, producing too much UV line blanketing.
- **Ionization balance**: The ζ-corrected Saha approximation may not capture the correct Si II/III ratio in the silicon-burning zone, directly affecting the depth and velocity of the Si II 6355 Å feature.
- **Calcium H&K**: The emission-to-absorption ratio of Ca II depends sensitively on the population of the $4p$ levels, which depart significantly from Boltzmann.

LUMINA implements a **full NLTE solver** for the four most spectroscopically important elements: Si, Ca, Fe, and S (8 ion stages total, $\sim$2000 levels). For these species, the complete statistical equilibrium is solved including all radiative (bound–bound and bound–free) and collisional rates—the same physics as CMFGEN or PHOENIX. All other species (C, O, Co, Ni) remain on the nebular approximation, as they contribute negligibly to the optical spectrum of normal SN Ia at photospheric epochs.

5.2 Statistical Equilibrium

For each NLTE ion, level populations are determined by the system of statistical equilibrium equations:

$$\boxed{\sum_{j\neq i} n_j\, R_{j\to i} = n_i \sum_{j\neq i} R_{i\to j} \quad \text{for each level } i} \tag{5.1}$$

where $R_{i\to j}$ is the total (radiative + collisional) rate from level i to level j. This forms a linear system $\mathbf{A}\mathbf{n} = \mathbf{0}$, supplemented by the conservation equation:

$$\sum_i n_i = n_{\text{total}} \quad \text{(from nebular ionization balance)} \tag{5.2}$$

5.2.1 Rate Matrix Structure

The rate matrix $\mathbf{A}$ has dimensions $N \times N$ where N is the combined level count for an ion pair (e.g., Si II + Si III). Ion pairs are solved together because photoionization and recombination couple the two stages.

Table 5.1: NLTE ion pairs and matrix dimensions.

Ion pair	**Levels**	**Matrix size**	**NLTE lines**
Si II + Si III	$100 + 169 = 269$	269×269	~2,400
Ca II + Ca III	$93 + 150 = 243$	243×243	~1,700
Fe II + Fe III	$796 + 566 = 1{,}362$	1362×1362	~28,000
S II + S III	$85 + 58 = 143$	143×143	~4,500
Total	**2,017**	—	~36,600

The matrix element A_{ij} $(i \neq j)$ represents the rate of transitions *into* level i from level j. The diagonal $A_{ii} = -\sum_{j \neq i} R_{i \to j}$ ensures row sums are zero.

5.3 Transition Rates

5.3.1 Radiative Bound–Bound

Using Einstein coefficients loaded from the atomic database:

$$R_{\text{abs}}(l \to u) = B_{lu}\,\bar{J}(\nu_{lu}) \qquad \text{(absorption)} \tag{5.3}$$

$$R_{\text{stim}}(u \to l) = B_{ul}\,\bar{J}(\nu_{lu}) \qquad \text{(stimulated emission)} \tag{5.4}$$

$$R_{\text{spont}}(u \to l) = A_{ul} \qquad \text{(spontaneous emission)} \tag{5.5}$$

where $\bar{J}(\nu)$ is the angle-averaged mean intensity at the line frequency, obtained from the Monte Carlo frequency histogram (Section 5.4).

5.3.2 Collisional Bound–Bound

Since no collision data exists for Si/Ca/Fe/S in the atomic database, we use standard approximation formulas:

Permitted transitions (van Regemorter 1962):

$$C_{l \to u} = 2.16 \times 10^{-6}\, n_e\, f_{lu}\, \frac{\bar{g}\, e^{-\Delta E / k_B T_e}}{g_l\, \sqrt{T_e}} \tag{5.6}$$

with effective Gaunt factor $\bar{g} \approx 0.2$ for allowed transitions.

Forbidden transitions (Axelrod 1980):

$$C_{l \to u} = 8.63 \times 10^{-6}\, n_e\, \frac{\Omega}{g_l\, \sqrt{T_e}}\, e^{-\Delta E / k_B T_e} \tag{5.7}$$

with effective collision strength $\Omega \approx 1.0$.

Downward collisional rates follow detailed balance:

$$C_{u\to l} = C_{l\to u}\,\frac{g_l}{g_u}\,e^{\Delta E/k_B T_e} \tag{5.8}$$

5.3.3 Photoionization

Photoionization couples the lower ion to the ground state of the higher ion via the Kramers hydrogenic cross-section:

$$\sigma_{\rm bf}(\nu) = \sigma_0 \left(\frac{\nu_{\rm thresh}}{\nu}\right)^3 \quad \text{for } \nu \geq \nu_{\rm thresh} \tag{5.9}$$

where $\sigma_0 = 7.91\times 10^{-18}/Z_{\rm eff}^2\,{\rm cm}^2$ and $\nu_{\rm thresh} = (\chi - E_l)/h$ is the level-dependent ionization threshold. The photoionization rate is:

$$R_{\rm bf}(l) = \int_{\nu_{\rm thresh}}^{\infty} \frac{4\pi\,\bar{J}_\nu\,\sigma_{\rm bf}(\nu)}{h\nu}\,d\nu \tag{5.10}$$

evaluated numerically over the $\bar{J}_\nu$ histogram.

5.3.4 Recombination

Recombination rates follow from the Milne detailed-balance relation:

$$R_{\rm rec}(l) = R_{\rm bf}(l) \times n_e \left(\frac{h^2}{2\pi m_e k_B T_e}\right)^{3/2} \frac{g_l}{2\,g_{\rm ion}}\,e^{(\chi - E_l)/k_B T_e} \tag{5.11}$$

5.4 The $\bar{J}_\nu$ Frequency Histogram

To evaluate radiative rates, the solver needs the frequency-resolved mean intensity $\bar{J}_\nu$ in each shell. This is accumulated during Monte Carlo transport as a logarithmically-binned frequency histogram:

$$j_\nu^{\rm raw}(s,b) = \sum_{\rm steps} \epsilon_{\rm cmf} \times d_{\rm step} \quad \text{for } \nu_{\rm cmf} \in \text{bin } b \tag{5.12}$$

normalized after each iteration to physical units:

$$\bar{J}_\nu(s,b) = \frac{j_\nu^{\rm raw}(s,b)}{4\pi\,V_{\rm shell}\,\Delta t_{\rm sim}\,\Delta\nu_b} \tag{5.13}$$

Definition 5.1 — Frequency Grid. 1000 logarithmically spaced bins covering $\nu_{\rm min} = 1.5\times 10^{14}\,{\rm Hz}$ ($\lambda = 20{,}000\,$Å) to $\nu_{\rm max} = 3\times 10^{16}\,{\rm Hz}$ ($\lambda = 100\,$Å), with $\Delta\log\nu = 0.00529$ per bin ($\sim$1.2% resolution).

On the GPU, each packet step adds one `atomicAdd` to the appropriate frequency bin, using the comoving-frame frequency already computed for the standard j-estimator. The memory cost is 240 KB (30 shells $\times$ 1000 bins $\times$ 8 bytes).

5.5 Matrix Solve: CPU and GPU Paths

5.5.1 CPU Path: Column-Oriented Gaussian Elimination

For the CPU binary (`lumina`), the rate matrix is solved via Gaussian elimination with partial pivoting. The matrix is stored in column-major format (for compatibility with the GPU path), and the elimination uses a cache-friendly column-oriented algorithm where the inner loop iterates over rows within a column (stride-1 access):

Listing 5.1: Cache-friendly column-oriented elimination

```
// Inner loop: column j (outer), row i (inner = contiguous)
for (int j = k + 1; j < N; j++) {
    double A_kj = A[j * N + k];  // pivot row element
    for (int i = k + 1; i < N; i++)
        A[j * N + i] -= A[k * N + i] * A_kj;
}
```

With OpenMP parallelization across shells (`schedule(dynamic,1)`), the CPU NLTE solve takes ∼20 s for all 4 ion pairs × 30 shells (dominated by Fe at 1362 × 1362).

5.5.2 GPU Path: cuBLAS Batched LU Factorization

For the GPU binary (`lumina_cuda`), the matrix solve uses cuBLAS batched operations to solve all 30 shells simultaneously on the GPU:

1. **Assembly (CPU, OpenMP)**: For each ion pair, assemble $N \times N$ rate matrices for all 30 shells in parallel (∼100 ms).
2. **Upload**: Copy matrices and RHS vectors to GPU.
3. **LU factorization**: `cublasDgetrfBatched()` — batched LU decomposition of 30 matrices.
4. **Triangular solve**: `cublasDgetrsBatched()` — batched forward/back substitution.
5. **Download**: Copy solution vectors back to CPU.

Table 5.2: NLTE solver performance comparison (200K packets, 3 iterations).

Configuration	Total time	NLTE overhead	Speedup
GPU transport, no NLTE	6.1 s	—	—
GPU transport + cuBLAS NLTE	9.1 s	3.0 s	1.0×
CPU+OMP transport + Gauss NLTE	22.8 s	15.4 s	—

The cuBLAS batched solve reduces the NLTE matrix solve from ∼20 s (CPU Gaussian elimination) to ∼3 s, a **7× speedup**. At production packet counts (2M+), the NLTE overhead becomes negligible compared to the transport kernel.

Important: GPU memory for the cuBLAS solver: the Fe 1362 × 1362 matrices for 30 shells require $30 \times 1362^2 \times 8 = 425$ MB of GPU memory, pre-allocated at initialization. This fits comfortably within the 32 GB VRAM of the RTX 5000 Ada.

5.6 $\tau_{\rm Sobolev}$ Update from NLTE Populations

After solving the rate equations, the Sobolev optical depth for each NLTE line is recomputed using the NLTE level populations instead of Boltzmann:

$$\tau_{\rm Sob} = \frac{\pi e^2}{m_e c}\, f_{lu}\, \lambda_{\rm cm}\, t_{\rm exp}\, n_l \left(1 - \frac{g_l\, n_u}{g_u\, n_l}\right) \tag{5.14}$$

where n_l and n_u are the NLTE lower and upper level populations respectively. Lines not belonging to NLTE ions retain their nebular τ values.

5.7 Integration into the Iteration Loop

The NLTE solver is called after each Monte Carlo iteration (for iterations > 1), following the standard plasma state update:

Iteration workflow with NLTE enabled

1. Monte Carlo transport kernel $\longrightarrow$ j_ν histogram + standard estimators
2. Solve radiation field: update W, $T_{\rm rad}$, $T_{\rm inner}$
3. Nebular plasma state: partition functions, n_e, Saha ionization, $\tau_{\rm Sob}$
4. **NLTE solve**: normalize $\bar{J}_\nu$ $\rightarrow$ assemble rate matrices $\rightarrow$ solve (GPU/CPU) $\rightarrow$ update τ for NLTE lines
5. Re-upload $\tau_{\rm Sob}$ to GPU for next iteration

Enable NLTE via the command line or environment variable:

```
# GPU
./lumina_cuda data/tardis_reference 200000 20 real nlte

# CPU
./lumina data/tardis_reference 200000 20 real nlte

# Or via environment variable
LUMINA_NLTE=1 ./lumina_cuda data/tardis_reference 200000 20
```

5.8 Roadmap: All-Species NLTE

While the current NLTE solver covers the four most important optical diagnostic elements, a complete treatment requires additional species and inter-species coupling.

5.8.1 Species Coverage

Table 5.3 shows the prioritised expansion roadmap.
After Tier 1, the NLTE solver will cover $\sim$3500 levels across 6 ion pairs, accounting for >99% of the optical depth in normal SN Ia. The runtime overhead is modest: cuBLAS batched LU scales well, and the added Ni II/III (1061×1061 matrix) is smaller than the existing Fe II/III (1362×1362).

Table 5.3: NLTE species expansion tiers for SN Ia.

Tier	Ion pair	Levels	Lines	Motivation
4*Current	Si II/III	269	1,815	Si II 6355 Å (primary SN Ia diagnostic)
	Ca II/III	243	2,518	Ca II H&K, NIR triplet
	Fe II/III	1,362	31,613	UV/optical line blanketing (dominant)
	S II/III	143	670	S "W" feature at 5640 Å
2*1	Co II/III	469	5,100	^{56}Ni decay product; optical blanketing
	Ni II/III	1,061	22,606	^{56}Ni parent; Fe-group blanketing
2*2	C II	$\sim$40	$\sim$200	C II 6580 Å (unburned carbon diagnostic)
	Mg II	$\sim$50	$\sim$100	Mg II 4481 Å (high-v SN Ia)

Table 5.4: Estimated runtime impact of NLTE expansion.

Component	Current (4 pairs)	After Tier 1 (6 pairs)	Change
Rate matrix construction	$\sim$1.5 s	$\sim$2.7 s	+80%
cuBLAS LU solve	$\sim$1.5 s	$\sim$2.0 s	+33%
NLTE subtotal	**$\sim$3.0 s**	**$\sim$4.7 s**	+57%
MC Transport	$\sim$10 s	$\sim$10 s	0%
Total per model	**$\sim$14 s**	**$\sim$15.7 s**	+12%

5.8.2 Charge Exchange

A physically important process omitted from most Monte Carlo NLTE implementations is **charge exchange** (CE):

$$A^+ + B^{2+} \rightleftharpoons A^{2+} + B^+ \tag{5.15}$$

Unlike photoionisation (which depends on J_ν) or collisional ionisation (which depends on n_e), CE rates depend on the *product of two ion densities*:

$$R_{\rm CE} = n(A^+) \cdot n(B^{2+}) \cdot \langle \sigma v \rangle_{\rm CE} \tag{5.16}$$

with typical $\langle \sigma v \rangle_{\rm CE} \sim 10^{-9}\,\mathrm{cm^3/s}$ [6]. In the dense inner shells of SN Ia ejecta ($n_{\rm ion} \sim 10^8$–$10^9\,\mathrm{cm^{-3}}$), CE rates can rival or exceed radiative recombination rates.

Important: The most important CE reactions for SN Ia are:

- $\mathrm{Fe^+ + Co^{2+} \rightleftharpoons Fe^{2+} + Co^+}$ (Fe/Co co-located in ^{56}Ni decay zone)
- $\mathrm{Fe^+ + Ni^{2+} \rightleftharpoons Fe^{2+} + Ni^+}$ (^{56}Ni parent material)
- $\mathrm{Si^+ + Ca^{2+} \rightleftharpoons Si^{2+} + Ca^+}$ (near-resonant ionisation potentials)

5.8.3 Implementation Strategy

LUMINA currently solves each ion pair independently. Adding CE coupling does *not* require solving all species simultaneously—even CMFGEN uses iterative convergence rather than a monolithic matrix:

Iterative CE coupling scheme

1. Solve each ion pair's NLTE rate equations independently (current approach).
2. After all pairs are solved, compute CE rates using the updated ionisation fractions.
3. Add CE rates as additional source/sink terms to each ion pair's rate matrix.
4. Re-solve all ion pairs with the updated rates.
5. Repeat steps 2–4 until ionisation fractions converge (typically 3–5 inner iterations).

This approach couples the species through n_e, T, and the CE rates themselves, without increasing the matrix dimensions. The additional cost is $\sim$3–5 re-solves of the existing batched LU system, adding at most $\sim$10 s per Monte Carlo iteration.

II

Code Architecture

6 Overview & Build System

6.1 File Structure

LUMINA-SN consists of the following source files:

Table 6.1: 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)
Total	**3936**	

6.2 Build System

Listing 6.1: Building LUMINA-SN

```
# CPU build (serial)
make

# CPU build with OpenMP parallelism
make OMP=1

# CUDA GPU build
make cuda
```

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

Table 6.2: 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`

6.3 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`

6.4 Execution

Listing 6.2: Running LUMINA-SN

```
# CPU: reference model with 200K packets, 20 iterations
./lumina tardis_reference 200000 20

# CUDA: same with GPU acceleration
./lumina_cuda atomic/kurucz_cd23_chianti_H_He.h5 200000 output.csv
```

7 Data Structures

7.1 The `RPacket` Structure

The fundamental unit of the Monte Carlo simulation:

Listing 7.1: `RPacket` — photon energy packet

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

7.2 Geometry

The 1D spherically symmetric ejecta model:

Listing 7.2: `Geometry` — shell structure

```c
typedef struct {
    int    n_shells;           // number of shells (default: 30)
    double *r_inner;           // [n_shells] inner radii [cm]
    double *r_outer;           // [n_shells] outer radii [cm]
    double *v_inner;           // [n_shells] inner velocities [cm/
        s]
    double *v_outer;           // [n_shells] outer velocities [cm/
        s]
    double time_explosion;     // t_exp [seconds]
} Geometry;
```

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

7.3 Opacity State

Pre-computed opacity data used during transport:

Listing 7.3: `OpacityState` — line and continuum opacities

```c
typedef struct {
    int n_lines, n_shells;
    double *line_list_nu;          // [n_lines] rest frequencies,
        descending
    double *tau_sobolev;           // [n_lines * n_shells]
    double *electron_density;      // [n_shells] n_e [cm^-3]
    double *t_electrons;           // [n_shells] T_e [K]

    // Macro-atom transition data
    int n_macro_levels, n_macro_transitions;
    int    *macro_block_references;        // [n_levels+1]
    int    *transition_type;               // [n_transitions]
    int    *destination_level_id;          // [n_transitions]
    int    *transition_line_id;            // [n_transitions]
    double *transition_probabilities;      // [n_transitions *
       n_shells]
    int    *line2macro_level_upper;        // [n_lines]
} OpacityState;
```

Remark 7.1 — 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.

7.4 Plasma State

Thermodynamic state updated each iteration:

Listing 7.4: `PlasmaState` — radiation field quantities

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

7.5 Atomic Data

Comprehensive atomic physics database for the plasma solver:

Listing 7.5: `AtomicData` — atomic physics for Saha–Boltzmann

```c
typedef struct {
    // Per-line data
    int    *line_atomic_number;    // [n_lines] Z
    int    *line_ion_number;       // [n_lines] ionization stage
    double *line_f_lu;             // [n_lines] oscillator
       strength
    double *line_wavelength_cm;    // [n_lines] rest wavelength

    // Energy levels
    int     n_levels;
    double *level_energy_eV;       // [n_levels]
    int    *level_g;               // [n_levels] statistical
       weight
    int    *level_metastable;      // [n_levels] 0 or 1

    // Ionization energies
    int     n_ionization;
    double *ioniz_energy_eV;       // [n_ionization]

    // Zeta correction factors (dilute non-LTE)
    double *zeta_data;             // [n_zeta_ions *
       n_zeta_temps]

    // Abundances
    double *abundances;            // [n_elements * n_shells]

    // Computed quantities (updated each iteration)
    double *ion_number_density;    // [n_ion_pops * n_shells]
    double *partition_functions;   // [n_ion_pops * n_shells]
} AtomicData;
```

7.6 Monte Carlo Estimators

Accumulated during transport, used to update the radiation field:

Listing 7.6: `Estimators` — radiation field accumulators

```c
typedef struct {
    double *j_estimator;         // [n_shells] integral of E*ds
    double *nu_bar_estimator;    // [n_shells] integral of E*nu*
       ds
    double *j_blue_estimator;    // [n_lines * n_shells] (CPU
       only)
    double *Edotlu_estimator;    // [n_lines * n_shells] (CPU
       only)
} Estimators;
```

Remark 7.2 — 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`.

8 Transport Engine

8.1 Overview

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

8.2 The `trace_packet` Function

This function computes the next interaction event for a packet:

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

8.3 Sobolev Line Sweep

The sweep algorithm processes lines in descending frequency order:

Listing 8.1: Sobolev sweep (simplified)

```c
double tau_trace_combined = 0.0;
for (int j = pkt->next_line_id; j < n_lines; j++) {
    double nu_line = line_list_nu[j];
    double d_line = compute_d_line(pkt, nu_line);

    if (d_line < 0 || d_line > d_boundary) break;

    double tau_line = tau_sobolev[j * n_shells + shell_id];
    tau_trace_combined += tau_line;

    if (tau_trace_combined > tau_event) {
        // Line interaction!
        *interaction_type = INTERACTION_LINE;
        *d_min = d_line;
        pkt->next_line_id = j;
        return;
    }
}
```

8.4 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

$$\varepsilon_{\text{lab,new}} = \varepsilon_{\text{cmf}} \times \frac{1}{1 - \mu_{\text{new}} \cdot v/c} \tag{8.1}$$

8.5 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).

9

Plasma Physics Solver

9.1 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 [11].

9.2 Step 1: Partition Functions

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

$$\mathcal{Z}(Z, \text{stage}, s) = \underbrace{\sum_{i \in \text{meta}} g_i \, e^{-E_i/k_B T_{\text{rad}}(s)}}_{\mathcal{Z}_{\text{meta}}} + W(s) \cdot \underbrace{\sum_{i \in \text{non-meta}} g_i \, e^{-E_i/k_B T_{\text{rad}}(s)}}_{\mathcal{Z}_{\text{non}}} \tag{9.1}$$

Important: The Boltzmann factors use T_{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.

9.3 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. 9.2)
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: $|n_e^{(k+1)} - n_e^{(k)}|/n_e^{(k)} < 0.05$
```

9.4 Step 3: Nebular Saha Ionization

The ionization ratio between consecutive stages is:

$$\boxed{\frac{n_{i+1}}{n_i} = \frac{\Phi_{\text{neb}}}{n_e}} \tag{9.2}$$

where the nebular ionization coefficient Φ_{neb} is:

$$\Phi_{\text{neb}} = \Phi_{\text{LTE}} \times W \times [\zeta \cdot \delta + W \cdot (1 - \zeta)] \times \sqrt{\frac{T_e}{T_{\text{rad}}}} \tag{9.3}$$

$$\Phi_{\text{LTE}} = \frac{\mathcal{Z}_{i+1}}{\mathcal{Z}_i} \times 2 \times g_e \times e^{-\chi/k_B T_{\text{rad}}} \tag{9.4}$$

$$g_e = \left(\frac{2\pi m_e k_B T_{\text{rad}}}{h^2}\right)^{3/2} \tag{9.5}$$

$$\delta = \frac{T_e}{T_{\text{rad}}} \exp\left[\chi\left(\frac{1}{k_B T_{\text{rad}}} - \frac{1}{k_B T_e}\right)\right] \tag{9.6}$$

Here χ is the ionization energy, and ζ is a non-LTE correction factor interpolated from tabulated values.

9.5 Step 4: τ_{Sobolev} Update

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

$$n_{l,\text{meta}} = \frac{g_l}{\mathcal{Z}} n_{\text{ion}}\, e^{-E_l/k_B T_{\text{rad}}}, \qquad n_{l,\text{non}} = W \times \frac{g_l}{\mathcal{Z}} n_{\text{ion}}\, e^{-E_l/k_B T_{\text{rad}}} \tag{9.7}$$

9.6 Step 5: Radiation Field Update

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

$$T_{\text{rad,est}}(s) = 1.2523 \times 10^{-11} \times \frac{\overline{\nu}(s)}{j(s)} \quad [\text{K}] \tag{9.8}$$

$$W_{\text{est}}(s) = \frac{j(s)}{4\,\sigma_{\text{SB}}\, T_{\text{rad}}^4(s)\, \Delta t\, V(s)} \tag{9.9}$$

All quantities are damped:

$$X_{\text{new}} = X_{\text{old}} + 0.5 \times (X_{\text{est}} - X_{\text{old}}) \tag{9.10}$$

10 Atomic Data System

10.1 Data Sources

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

Table 10.1: Atomic data files.

File	**Format**	**Contents**
`line_list.csv`	CSV	ν, Z, ion, f_{lu}, λ per line
`levels.csv`	CSV	E (eV), g, metastable flag per level
`ionization_energies.csv`	CSV	χ per ion
`tau_sobolev.npy`	NPY	τ_{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

10.2 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

10.3 The CSV Parser

Important: The `macro_atom_data.csv` header starts with an unnamed index column: “`,atomic_number,...`”. The standard `strtok()` function skips leading delimiters, causing a column offset of -1. LUMINA uses a manual field-by-field parser that handles empty fields correctly.

11 CUDA GPU Implementation

11.1 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.

11.2 Memory Layout

Table 11.1: GPU memory allocation.

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

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

11.3 Kernel Launch Configuration

Listing 11.1: Kernel launch

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

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

11.4 Random Number Generation

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

Listing 11.2: Per-thread RNG initialization

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

11.5 Atomic Estimator Updates

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

Listing 11.3: Estimator accumulation on GPU

```
__device__ void update_estimators(
    double *d_j_est, double *d_nu_bar_est,
    int shell_id, double comov_energy,
    double comov_nu, double distance)
{
    atomicAdd(&d_j_est[shell_id],
              comov_energy * distance);
    atomicAdd(&d_nu_bar_est[shell_id],
              comov_energy * distance * comov_nu);
}
```

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

11.6 Performance

Table 11.2: CPU vs GPU performance (NVIDIA RTX 5000 Ada, sm_89).

N_{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 11.1 — Statistical Accuracy. GPU and CPU produce statistically identical results. At 200K packets: W error 1.06%, T_{rad} error 0.58% (relative to TARDIS reference). The scaling follows $\sigma \propto N^{-0.35}$ to $N^{-0.40}$, close to Poisson ($N^{-0.5}$).

11.7 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 $10^{-10} \times \Delta r$.

III

Usage & Applications

12 Installation & Quick Start

12.1 Prerequisites

Table 12.1: System requirements.

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

12.2 Step-by-Step Setup

Listing 12.1: Complete setup procedure

```
# Clone the repository
git clone git@github.com:kjhan0606/lumina-sn.git
cd lumina-sn

# Build CPU version
make

# (Optional) Build GPU version
make cuda

# Verify with a quick test (1000 packets, 5 iterations)
./lumina tardis_reference 1000 5

# Production run (200K packets, 20 iterations)
./lumina tardis_reference 200000 20

# Check output
head lumina_spectrum.csv
```

12.3 Input Directory Structure

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

Listing 12.2: Required input files

```
data/tardis_reference/
  config.json                   # Simulation parameters
  geometry.csv                  # Shell radii and velocities
  density.csv                   # Mass density per shell
  abundances.csv                # Element mass fractions
  electron_densities.csv        # Initial electron densities
  plasma_state.csv              # Initial W, T_rad per shell
  line_list.csv                 # Atomic line data
  tau_sobolev.npy               # Reference optical depths
  transition_probabilities.npy
  macro_atom_data.csv
  macro_atom_references.csv
  line2macro_level_upper.npy
  levels.csv                    # Energy levels
  ionization_energies.csv
  zeta_ions.csv
  zeta_temps.csv
  zeta_data.npy
  atom_masses.csv
```

12.4 Output Files

Table 12.2: Output files produced by LUMINA.

File	**Contents**
`lumina_spectrum.csv`	λ (Å) vs L_λ (erg/s/cm)
`lumina_plasma_state.csv`	Final W, $T_{\rm rad}$ per shell
`lumina_tau_validation.csv`	$\tau_{\rm Sob}$ at shell 0 (debug)

13 The Ejecta Model

13.1 Three-Zone Composition

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

Table 13.1: Default three-zone composition model.

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

13.2 Broken Power-Law Density

The density profile is a broken power law:

$$\rho(v) = \begin{cases} \rho_0 \left(\dfrac{v}{v_{\text{inner}}}\right)^{n_{\text{inner}}} & v < v_{\text{break}} \\ \rho_{\text{break}} \left(\dfrac{v}{v_{\text{break}}}\right)^{n_{\text{outer}}} & v \geq v_{\text{break}} \end{cases} \tag{13.1}$$

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

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

13.3 Physical Parameter Space

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

Table 13.2: Full 15D parameter space with ranges.

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

14 Parameter Fitting

14.1 Fitting Strategy

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

14.1.1 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

14.1.2 Phase 2: Refinement

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

14.1.3 Phase 3: Production

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

14.2 Objective Function

The composite scoring function includes:

$$\text{Score} = \text{RMS}_{\text{spec}} + 0.5\,|\Delta d_{\text{Si II}}| + 0.2\,|\Delta \log v_{\text{Si II}}| + 0.1\,|\Delta\lambda_{\text{min}}| \tag{14.1}$$

where:

- RMS_{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

15 Physical Constants & Reference Values

Table 15.1: 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	1.380649×10^{-16} erg/K
σ_{SB}	Stefan–Boltzmann	5.670374×10^{-5} erg/cm^2/s/K^4
σ_T	Thomson cross-section	6.6525×10^{-25} cm^2
m_e	Electron mass	9.10938×10^{-28} g
e	Electron charge	4.80321×10^{-10} esu
$\pi e^2/(m_e c)$	Sobolev coefficient	2.6540×10^{-2} cm^2/s
$T_{\mathrm{rad,const}}$	T_{rad} estimator constant	1.2523×10^{-11} K·s

16 Comparison with Other Radiative Transfer Codes

Numerous radiative transfer codes exist for modelling supernova spectra and light curves. This chapter places LUMINA-SN in context by comparing its design philosophy, physics scope, and computational performance against the most widely used alternatives.

16.1 Overview

Table 16.1 summarises the key features of each code.

Table 16.1: Comparison of supernova radiative transfer codes.

Feature	**LUMINA**	**TARDIS**	**SYN++**	**SEDONA**	**ARTIS**	**CMFGEN**
RT method	MC	MC	Param.	MC	MC	CMF-ALI
Geometry	1D	1D	1D	3D	3D	1D
NLTE	Full[a]	Dilute	LTE	LTE/NLTE	Full	Full
Macro-atom	Yes	Yes	No	Partial	Yes	—
GPU accel.	CUDA	No	No	No	No	No
Time-dep.	No	No	No	Yes	Yes	No
Language	C/CUDA	Py/Cy	C++	C++	C++	Fortran

[a]Full statistical equilibrium for Si, Ca, Fe, S (2017 levels); other species use the nebular approximation.

16.2 TARDIS

TARDIS [5] is the direct ancestor of LUMINA-SN. Both codes implement the same Monte Carlo radiative transfer formalism in 1D homologous expansion with the macro-atom method of Lucy [9, 10].

LUMINA advantages over TARDIS:

- **GPU acceleration**: CUDA transport kernel provides $\sim 10\times$ speedup over TARDIS's Cython implementation (200K packets: 0.7 s vs 7.2 s).
- **Full NLTE solver**: Statistical equilibrium with radiative + collisional rates for Si/Ca/Fe/S, solved via cuBLAS batched LU on GPU. TARDIS uses the dilute-LTE (nebular) approximation for all species.

- **Minimal dependencies**: Pure C99/CUDA binary with no Python runtime, enabling HPC deployment without Conda environments.
- **Integrated ML pipeline**: Latin Hypercube sampling + neural emulator + SBI/MCMC inference chain for automated parameter fitting.

TARDIS advantages over LUMINA:

- **Broader scope**: Models CC-SNe, kilonovae (with lanthanide opacities), and other transients.
- **Larger community**: 40+ contributors, comprehensive documentation, active development.
- **Flexible plasma module**: Pluggable ionization/excitation solvers via the `plasma` framework.
- **Deeper Si II trough**: TARDIS achieves 93% absorption depth vs LUMINA's 75–83%, likely due to subtle differences in the macro-atom source function implementation.

16.3 SYN++

SYN++ [15] is a parametric spectrum synthesis tool descended from SYNOW. It assumes a sharp photosphere emitting a blackbody, with resonance-scattering line profiles computed in the Sobolev approximation. Each ion is characterised by a minimum velocity, optical depth, and excitation temperature.

- **LUMINA advantage**: Self-consistent radiation field—ionization balance, dilution factors, and level populations are computed from the Monte Carlo simulation rather than input by hand. Fluorescence and macro-atom redistribution produce realistic P Cygni profiles with emission components.
- **SYN++ advantage**: Extremely fast (seconds per model), making it ideal for rapid line identification, spectral classification, and interactive fitting. Its simplicity is a strength for survey-scale work.

SYN++ and LUMINA serve different purposes: SYN++ is a *fitting tool* for quick diagnostics, while LUMINA is a *physics code* for quantitative abundance and density studies.

16.4 SEDONA

SEDONA [4] is a multi-dimensional, time-dependent Monte Carlo code designed to compute both spectra and light curves from first principles.

- **LUMINA advantage**: GPU acceleration ($\sim 10\times$) and NLTE solver for key diagnostic ions. LUMINA's ML fitting pipeline enables automated parameter estimation, whereas SEDONA is typically run on pre-computed explosion models.
- **SEDONA advantage**: Full 3D geometry with time-dependent radiation transport, γ-ray deposition from ^{56}Ni decay, and multi-epoch light curve computation. It can model asymmetric explosions, viewing-angle effects, and nebular-phase spectra.

SEDONA is an "explosion-to-observables" code that starts from hydrodynamic models, while LUMINA is optimised for single-epoch spectral fitting against observations.

16.5 ARTIS

ARTIS [7, 14] is a 3D time-dependent Monte Carlo code with full NLTE, developed primarily for SN Ia. It incorporates macro-atom physics and γ-ray transport.

- **LUMINA advantage**: GPU acceleration and integrated parameter search. A single-epoch LUMINA model runs in $\sim$14 s on GPU, enabling 5,000-model grid searches in $\sim$19 hours. ARTIS runs are typically CPU-only and take hours per model.
- **ARTIS advantage**: Full 3D geometry, time dependence, complete NLTE for all species, and γ-ray energy deposition. ARTIS can model both photospheric and nebular phase spectra from the same explosion model.

16.6 CMFGEN

CMFGEN [3] solves the radiative transfer and statistical equilibrium equations simultaneously in the comoving frame using the accelerated lambda iteration (ALI) method. It treats millions of lines in full NLTE.

- **LUMINA advantage**: Monte Carlo naturally handles line overlap and fluorescence without explicit frequency-by-frequency integration. GPU acceleration makes large parameter surveys feasible (thousands of models per day vs one CMFGEN model per several hours).
- **CMFGEN advantage**: Full NLTE for *all* species simultaneously (not just 4 elements), self-consistent radiation–matter coupling via ALI, wind clumping, and decades of validation across diverse astrophysical objects (OB stars, WR stars, LBVs, SN Ia, CC-SNe). CMFGEN represents the gold standard for spectroscopic precision.

16.7 PHOENIX

PHOENIX [2] is a general-purpose NLTE atmosphere code applicable to an extraordinarily wide range of objects: stars, brown dwarfs, exoplanet atmospheres, novae, and supernovae.

- **LUMINA advantage**: SN Ia-specific optimisation, GPU acceleration, and lightweight codebase ($\sim$5K lines vs $\sim$200K).
- **PHOENIX advantage**: Broadest applicability of any radiative transfer code, full NLTE with the most extensive atomic database, and 30+ years of validation.

16.8 Summary: LUMINA's Niche

LUMINA-SN occupies a unique position in the landscape of supernova radiative transfer codes:

Important: LUMINA = TARDIS physics + GPU acceleration + NLTE + ML fitting pipeline.

No other code combines Monte Carlo macro-atom transport with GPU acceleration and an integrated Bayesian inference framework. This makes LUMINA the tool of choice for **large-scale, automated spectral fitting of SN Ia** at photospheric epochs.

The trade-offs are clear: LUMINA sacrifices multi-dimensionality (vs SEDONA, ARTIS), time dependence (vs SEDONA, ARTIS), and universal NLTE coverage (vs CMFGEN, PHOENIX) in exchange for **speed** and **automation**. For the specific problem of fitting observed SN Ia spectra near maximum light, this trade-off is highly favourable.

Table 16.2: Approximate single-model execution times (200K packets, 20 iterations for MC codes).

Code	**Time**	**Hardware**
SYN++	~1 s	1 CPU core
LUMINA	~14 s	1 GPU (RTX 5000 Ada)
TARDIS	~60–120 s	1 CPU core (Cython)
ARTIS	~hours	MPI cluster
SEDONA	~hours	MPI cluster
CMFGEN	~hours	1–8 CPU cores
PHOENIX	~hours	MPI cluster

Table 16.3 clarifies the distinction between “full NLTE” implementations.

Table 16.3: NLTE implementation scope across codes.

Code	**NLTE physics**	**Species coverage**
CMFGEN	Full stat. equil. + ALI	All species (millions of transitions)
PHOENIX	Full stat. equil. + ALI	All species
ARTIS	Full stat. equil. (MC)	All species
LUMINA	Full stat. equil. (MC + J_ν)	Si, Ca, Fe, S (2017 levels)
TARDIS	Dilute-LTE (nebular approx.)	N/A (no NLTE solver)
SYN++	LTE (Boltzmann)	N/A
SEDONA	Mixed (LTE + optional NLTE)	Configuration-dependent

As shown in Table 16.3, LUMINA’s NLTE solver applies the *same physics*—full statistical equilibrium with radiative and collisional bound–bound rates, photoionisation (Kramers), and recombination (Milne)—as CMFGEN and PHOENIX. The difference is purely in **species coverage**: LUMINA solves NLTE for the four elements most critical to SN Ia optical diagnostics, while CMFGEN and PHOENIX solve it for all species simultaneously. For SN Ia photospheric spectra, the optical depth is overwhelmingly dominated by Si, Ca, Fe, and S lines, so this is a well-motivated approximation.

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] P. H. Hauschildt and E. Baron. “Numerical solution of the expanding stellar atmosphere problem”. In: *Journal of Computational and Applied Mathematics* 109 (1999), pp. 41–63. DOI: `10.1016/S0377-0427(99)00153-3`.

[3] D. J. Hillier and D. L. Miller. “The Treatment of Non-LTE Line Blanketing in Spherically Expanding Outflows”. In: *The Astrophysical Journal* 496 (1998), pp. 407–427. DOI: `10.1086/305350`.

[4] D. Kasen, R. C. Thomas, and P. Nugent. “Time-dependent Monte Carlo Radiative Transfer Calculations for Three-dimensional Supernova Spectra, Light Curves, and Polarization”. In: *The Astrophysical Journal* 651 (2006), pp. 366–380. DOI: `10.1086/506190`.

[5] 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`.

[6] J. B. Kingdon and G. J. Ferland. “Rate Coefficients for Charge Transfer between Hydrogen and the First 30 Elements”. In: *The Astrophysical Journal Supplement Series* 106 (1996), pp. 205–211. DOI: `10.1086/192335`.

[7] M. Kromer and S. A. Sim. “Time-dependent three-dimensional spectrum synthesis for Type Ia supernovae”. In: *Monthly Notices of the Royal Astronomical Society* 398 (2009), pp. 1809–1826. DOI: `10.1111/j.1365-2966.2009.15256.x`.

[8] L. B. Lucy. “Improved Monte Carlo techniques for the spectral synthesis of supernovae”. In: *Astronomy & Astrophysics* 345 (1999), pp. 211–220.

[9] L. B. Lucy. “Monte Carlo techniques for time-dependent radiative transfer in 3-D supernovae”. In: *Astronomy & Astrophysics* 384 (2002), pp. 725–735.

[10] L. B. Lucy. “Monte Carlo transition probabilities”. In: *Astronomy & Astrophysics* 403 (2003), pp. 261–275.

[11] 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.

[12] 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.

[13] M. M. Phillips. “The absolute magnitudes of Type IA supernovae”. In: *The Astrophysical Journal Letters* 413 (1993), pp. L105–L108.

[14] S. A. Sim. “Multidimensional simulations of radiative transfer in Type Ia supernovae”. In: *Monthly Notices of the Royal Astronomical Society* 375 (2007), pp. 154–168. DOI: `10.1111/j.1365-2966.2006.11271.x`.

[15] R. C. Thomas, P. E. Nugent, and J. C. Meza. “SYNAPPS: Data-Driven Analysis for Supernova Spectroscopy”. In: *Publications of the Astronomical Society of the Pacific* 123 (2011), pp. 237–248. DOI: `10.1086/658867`.

Index