xstar-tools: AI-Assisted Modernization of the XSTAR Photoionization Code

Ash Danehkar

Universities Space Research Association, Huntsville, AL, 35805, USA, E-mail: adanehkar@usra.edu

ADASS XXXVI

AI as a tool for software engineering

Introduction

XSTAR is a FORTRAN code for calculating the physical conditions and emitted and transmitted spectra of photoionized gas. It combines ionization and excitation balance with radiative transfer, solving thermal balance or using a prescribed temperature (Kallman & Bautista 2001; Kallman et al. 2004). Its atomic database supplies transition probabilities, cross sections, and rate coefficients (Bautista & Kallman 2001). We present xstar-tools, an open-source modernization developed through AI-assisted software development under human supervision. It provides an independent C++ engine, Python interfaces, and native model-grid tools, aiming to preserve scientific behavior while improving portability and integration.

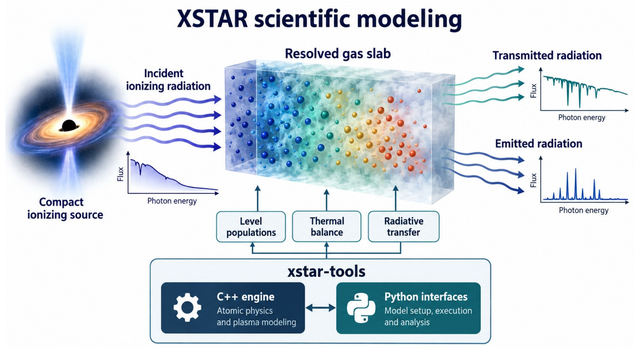


Fig 1. Conceptual illustration of XSTAR modeling and its outputs.

AI-assisted code development

Large language models supported an iterative engineering process, assisting code translation, discrepancy investigation, memory and runtime optimization, test development, and documentation. Human review examined numerical semantics and physical assumptions, while comparisons with FORTRAN XSTAR guided corrections. We accepted revisions after the relevant qualification checks passed and evaluated optimization changes for scientific consistency, memory use, and runtime.

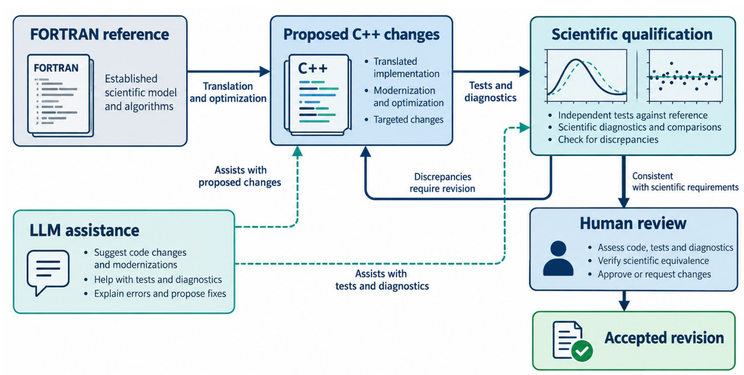


Fig 2. Discrepancies trigger investigation and revision.

Scientific qualification

FORTRAN XSTAR serves as the scientific reference. Qualification examines atomic level populations, heating and cooling rates, convergence trajectories, and emergent spectra. Comparisons of intermediate states help locate discrepancies that final outputs alone may obscure. Numerical acceptance criteria establish agreement with FORTRAN, while exact comparisons of payloads against accepted baselines guard against unintended changes.

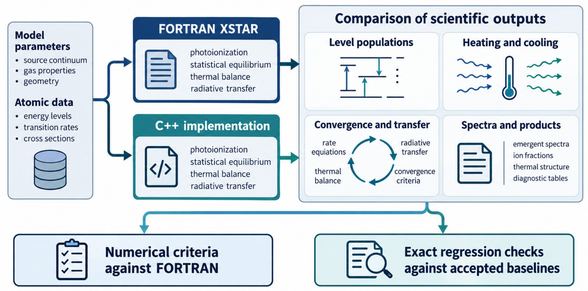


Fig 3. Scientific qualification against FORTRAN XSTAR and C++ baselines.

We compare iteration counts and balance diagnostics with FORTRAN XSTAR to examine convergence trajectories. Table 1 shows selected diagnostics from the FORTRAN magnesium reference model. These values describe the reference calculation, rather than differences between implementations.

FORTRAN XSTAR reference output · Magnesium model

log(r)	delr/r	log(N)	log(xi)	x_e	log(n)	log(t)	h-c(%)	h-c(%)	log(tau) fwd	log(tau) rev	ntotit
15.25	-36.00	-10.00	1.50	1.20	12.00	4.83	-0.01	0.00	-10.00	-10.00	17
15.25	-36.00	-10.00	1.50	1.20	12.00	4.83	-0.01	0.02	-10.00	-10.00	1
15.25	-7.58	19.67	1.50	1.20	12.00	4.83	0.03	0.04	-2.99	-10.00	9
15.25	-7.28	19.97	1.50	1.20	12.00	4.83	0.02	0.05	-2.69	-10.00	11
15.25	**-7.25**	**20.00**	**1.50**	**1.20**	**12.00**	**4.83**	**0.02**	**0.05**	**-2.67**	**-10.00**	**11**

Table 1. Diagnostics from the FORTRAN XSTAR Mg reference model.

Memory and runtime optimization

For a broad multi-element model, optimization reduced runtime from two hours to 12 minutes and peak memory by roughly two-thirds relative to the earlier C++ implementation. Changes underwent scientific qualification before acceptance. Performance remains model-dependent: the optimized multi-element calculation was faster than FORTRAN on the measured workstation, while low-ionization cases remained slower.

Broad multi-element model • Two distinct comparisons

A | C++ development: same-laptop comparison

Metric	Earlier C++ 0.6.82.33.8	Optimized C++ 0.6.82.40.2.46.1	Improvement
Wall time	≈120 min	≈12 min	**≈10× faster**
Peak memory, as summarized	≈11–12 GB	≈3.7 GB	**≈two-thirds lower**

B | FORTRAN–C++ comparison: workstation

Metric	FORTRAN XSTAR	Optimized C++
Wall time	940.90 s	**660.71 s**
Peak RSS	2.57 GB	**3.78 GB**

Table 2. Historical multi-element performance comparisons.

Native model-grid workflows

The suite provides a native workflow for planning parameter grids, executing XSTAR models, and assembling XSPEC spectral tables. Results are collected in canonical grid order, so job completion order does not affect table placement. The workflow handles model FITS products and ordered STEP-log concatenation separately.

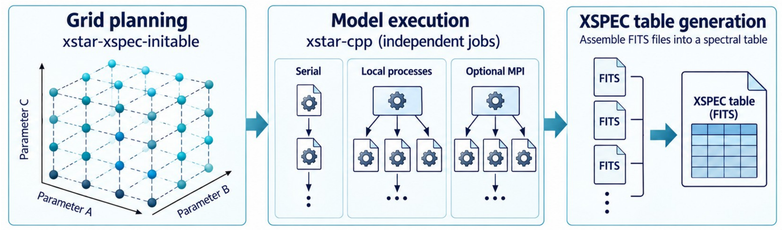


Fig 4. Native parameter-grid execution and XSPEC table assembly.

Single-model execution

```
./xstar-cpp --input xstar.par \
  --data-dir /path/to/xstar/data --output-dir run_xstar
```

Local execution: four independent processes

```
./xstar-xspec --processes 4 --input xstinitable.par \
  --data-dir /path/to/xstar/data --output-dir run_local
```

MPI execution: four ranks

```
mpirun -np 4 ./xstar-xspec-mpi --input xstinitable.par \
  --data-dir /shared/xstar/data --output-dir /shared/run_mpi
```

MPI execution distributes independent runs across ranks, following the parallelization approach of Danehkar et al. (2018).

Conclusions and future work

xstar-tools combines an independent C++ engine with Python interfaces and native model-grid workflows. AI assisted development, while human review and qualification against FORTRAN XSTAR guided acceptance. Profiling substantially reduced runtime and memory relative to earlier C++ implementations.

Future work will reduce intermediate file I/O and extend the framework to post-processing hydrodynamic and magnetohydrodynamic simulations for synthetic spectra.

References

Bautista & Kallman, 2001, ApJS, 134, 139
Danehkar, Nowak, et al. (2018), PASP, 130, 024501
Kallman & Bautista, 2001, ApJS, 133, 221
Kallman, Palmeri, Bautista, et al. 2004, ApJS, 155, 675

Acknowledgments. Supported partially by NASA Grant 80MSFC22M004.

Code: github.com/xstar-tools/xstar-tools