Tim Jenness[1]

[1]*NSF-DOE Vera C. Rubin Observatory / NSF NOIRLab, Tucson, AZ, USA;*
tim.jenness@noirlab.edu

Using LLMs to Port a Large C Library to Rust

Abstract. In 2026 large language model coding agents such as Codex and Claude have developed rapidly and changed the face of software development. They can be used for debugging, performance optimization, and adding features to existing codebases. As the tooling improved we decided to see if we could use these systems to port a large C library to Rust. The library we chose was the Starlink AST library, a library that consists of 200,000 lines of C (not including comments or blank lines) that has been in development for over 30 years and contains some very specific fixes for very rare combinations of FITS headers.

In this paper I will tell the story of agents getting confused (but also agents improving), agents cutting corners and hoping I wouldn't notice, agents finding bugs in the original code but then making sure the Rust had the same bugs, eye-watering token costs, and how the real lesson from this journey is that your ported code is only as good as the test suite of your original library.

1. Introduction

The Starlink AST library (Warren-Smith & Berry 1998; Berry et al. 2016) has been in development since 1997 and provides a generalized approach to world coordinate systems (WCS). Standard FITS WCS (Calabretta & Greisen 2002) describes only undistorted celestial coordinates, and every distortion convention, of which SIP (Shupe et al. 2005) is the only one in common use, needs its own FITS headers. AST instead defines a core set of transformations that can be combined in any order, serially or in parallel. It is object-oriented C, with virtual dispatch tables, reference counting, and a bespoke memory allocator, and the core library is nearly 475,000 lines including its extensive comments, plus 65,000 lines of vendored WCSLIB, ERFA, and PAL.

Rubin Observatory adopted AST for WCS handling in the LSST Science Pipelines in 2016 (Parejko & Owen 2016), hiding it behind a C++ `SkyWcs` class and more recently behind the `SkyProjection` class of the new image data model (Bosch et al. 2026). Long-term support has been a concern ever since, given the single developer, who has now retired, and the steep learning curve imposed by the object-oriented C. Rust was approved for use in the pipelines in early 2025, and a Rust port of AST would bring memory safety, a much stronger type system with explicit error handling in place of the AST global status, and native support for the object model that currently requires large amounts of boilerplate. A port is not a new idea: a native Java port begun by Starlink in 2002 (Bly et al. 2003) was abandoned within the year in favor of a JNI wrapper (Berry 2004).

Large language models (LLMs) have been studied as an aid to language translation for some time (Ibrahimzada et al. 2025; Dehghan et al. 2025), but the largest C to Rust codebase in the referenced work was of order 15,000 lines and the field moves so fast that results from 2025 are already dated. In February 2026, after watching OpenAI Codex modernize the 13,000-line C Python wrapper for AST, I wondered whether a port of the full library was now feasible. A port of this size cannot be reviewed line by line, so there is a real danger that a production library ends up effectively "vibe-coded" (Sarkar & Drosos 2025) but it seemed worth attempting. This paper summarizes the experience; a fuller account, including the design specification, is given in Jenness (2026).

2. The First Prototype

On 2026-03-03 I pointed Codex (GPT-5.3) at the C source with a short prompt and a `TODO.md` file saying that the Rust should drop the boilerplate and focus on the business logic. Two days later there were 25,000 lines of Rust library code and 12,000 lines of tests, and my weekly token allocation was exhausted. The agent built the C library itself and wrote code against it as an "oracle" (its term) with which to compare the Rust. After ten days about 15% of the C logic had been ported, focused on reading a Rubin FITS header with SIP distortions and transforming between pixel and sky coordinates. Single point transforms were 3 to 8 times faster than the C, albeit with no bad value support, so this demonstrated that adopting Rust would not necessarily lose performance.

Progress then stalled as the agent tied itself in knots on the mapping simplification logic, and when I stepped back it was clear that it had repeatedly cut corners to pass tests. It cached the oracle's serialized form of known FITS headers and wrote the cached text back out; it generated "oracle" fixtures from the Rust itself; it added special cases for individual test files; and it put all simplification logic in one file rather than in each mapping class as the C does, because that was less work. It never implemented bad value handling because no C test exercised it. The lesson was that the test coverage of the C library, then mainly in Fortran with no coverage reporting, is what determines the quality of the port.

3. Tests Are Important

In April 2026 I switched to Claude Opus 4.6 and used it to build a CMake build system for the C library, convert the Fortran tests to C, and add coverage reporting to CI. Core coverage was about 50%, and sanitizer builds immediately found latent buffer overruns. Because the FITS reader encodes decades of historical header variants, Claude was then asked to generate headers targeting specific uncovered lines, adding 90 example FITS headers. Later, hundreds of generated fixtures provided a reference corpus for the simplification engine.

4. The Second Port

Work restarted from scratch on 2026-04-27 with Claude Opus 4.6, an API key provided by SLAC, and a written design specification covering memory management, mutability,

and how Rust traits relate to C methods. Work continued through October 2026, interleaved with upstream bug fixes and new fixtures in the C library, with token starvation the main impediment: the monthly API ceiling could be hit in the first week. Newer models reached solutions faster, so the net cost per feature can fall even as the price per token rises. The port, published as a Rust crate, now covers all mappings, frames, and regions, and FITS channels, roughly 80% of the C library, but no C or Python binding.

Despite the specification and the better tests, the second port repeated some mistakes. Simplification logic was again placed in the wrong files, and different internal logic led to different serialized forms. Implementing the FITS channel before `TimeFrame` existed left time conversion logic in the wrong place, and hundreds of failing fixtures made optional early on were forgotten because the test suite was green. Many tokens went on C implementation details which exist only because the C clones mappings on insertion whereas Rust transfers ownership.

The port also found around one hundred bugs in the C library, including uninitialized memory reads in `MatrixMap` splitting, a `DATE-OBS` parser that dropped leading zeros from fractional seconds, rounding of negative values toward zero at 70 sites, and an infinite loop. Several fixes made the native serialization deterministic because byte-exact comparison with the oracle requires predictable output. The agent will sometimes notice a bug in the original and then faithfully reproduce it, so I added a standing instruction that suspected upstream bugs must be recorded separately.

5. Lessons Learned

A port is only as good as the test coverage of the original. The agent takes the easiest path to satisfying the prompt, and asking it to be faithful to the source is not enough without tests that demonstrate exactly what is required; adding coverage of edge cases before the port starts pays for itself many times over, and the agent can write those tests. Write a plan first, refer to it from `CLAUDE.md` or equivalent, and keep a bounded ledger of open work, including every original test, so that nobody has to rescan both source trees to see what has been ported. Correctness and completeness beat performance: do not optimize until every test passes. Port algorithms, not lines, using native features such as `Option` and `Result` for status handling, and annotate where each block came from. Enforce coding standards with commit hooks since this is far stronger than notes in `CLAUDE.md`. Declare the original to be the source of truth, allow heuristics only where the C has the same heuristic, and when the agent flails have it instrument the oracle to trace the exact code path for a passing test. Both ports showed high initial velocity because the core mappings are the quick part; the final 20% of edge cases and optimizations is where byte-exact output, which a mission critical library needs, becomes hard. Finally, it became clear during this work that a Pro or Max Team account is much more cost effective than an API key — the team accounts have weekly and hourly caps but the high end accounts can get far more work done for much less money than you spend on an API key.

6. Conclusions

A coding agent such as Claude Opus or Fable is now capable of porting a large library to another language provided the original is well tested. The test suite, a written plan, and

a reference oracle mitigate most of the problems seen when agents write large volumes of new code without such boundaries. The LSST Science Pipelines contains 150,000 lines of C++ in libraries that are mostly far less complex than AST, and converting them to Rust is now a plausible enterprise.

Acknowledgments. This material is based upon work supported in part by the National Science Foundation through Cooperative Agreements AST-1258333 and AST-2241526 and Cooperative Support Agreements AST-1202910 and AST-2211468 managed by the Association of Universities for Research in Astronomy (AURA), and the Department of Energy under Contract No. DE-AC02-76SF00515 with the SLAC National Accelerator Laboratory managed by Stanford University. Additional Rubin Observatory funding comes from private donations, grants to universities, and in-kind support from LSST-DA Institutional Members. I thank David Berry and Jim Bosch for their feedback on the initial design spec and for answering numerous questions. I also thank Michael Shehadeh from SLAC IT for his help with the Claude configuration and API key at SLAC, and Cristian Silva for giving me access to a separate Claude Team account. The project would not have been possible without the generous allocations of tokens from SLAC IT.

References

Berry, D. S. 2004, in Astronomical Data Analysis Software and Systems (ADASS) XIII, edited by F. Ochsenbein, M. G. Allen, & D. Egret, vol. 314 of Astronomical Society of the Pacific Conference Series, 412

Berry, D. S., Warren-Smith, R. F., & Jenness, T. 2016, Astronomy and Computing, 15, 33. eprint: 1602.06681

Bly, M. J., Giaretta, D., Currie, M. J., & Taylor, M. 2003, in Astronomical Data Analysis Software and Systems XII, edited by H. E. Payne, R. I. Jedrzejewski, & R. N. Hook, vol. 295 of Astronomical Society of the Pacific Conference Series, 445

Bosch, J. F., Jenness, T., & Dubois-Felsmann, G. P. 2026, in Software and Cyberinfrastructure for Astronomy IX, edited by J. Ibsen, & V. Alberti (SPIE), vol. 14155 of Proc. SPIE, 1415517. URL https://doi.org/10.1117/12.3102964

Calabretta, M. R., & Greisen, E. W. 2002, A&A, 395, 1077

Dehghan, S., Sun, T., Wu, T., Li, Z., & Jabbarvand, R. 2025, Translating Large-Scale C Repositories to Idiomatic Rust. URL https://doi.org/10.48550/arXiv.2511.20617

Ibrahimzada, A. R., Ke, K., Pawagi, M., Abid, M. S., Pan, R., Sinha, S., & Jabbarvand, R. 2025, Proc. ACM Softw. Eng., 2

Jenness, T. 2026, Experimenting with Porting a Large C Library to Rust Using Coding Agents, Data Management Technical Note DMTN-349, NSF-DOE Vera C. Rubin Observatory. URL https://doi.org/10.71929/rubin/17426051

Parejko, J., & Owen, R. 2016, WCS and Distortion Requirements and Existing Options, Data Management Technical Note DMTN-010, NSF-DOE Vera C. Rubin Observatory. URL https://dmtn-010.lsst.io/

Sarkar, A., & Drosos, I. 2025, in Proceedings of the 36th Annual Conference of the Psychology of Programming Interest Group (PPIG 2025)

Shupe, D. L., Moshir, M., Li, J., Makovoz, D., Narron, R., & Hook, R. N. 2005, in Astronomical Data Analysis Software and Systems XIV, edited by P. Shopbell, M. Britton, & R. Ebert, vol. 347 of Astronomical Society of the Pacific Conference Series, 491

Warren-Smith, R. F., & Berry, D. S. 1998, in Astronomical Data Analysis Software and Systems VII, edited by R. Albrecht, R. N. Hook, & H. A. Bushouse, vol. 145 of Astronomical Society of the Pacific Conference Series, 41