Glu's KPI Report

1. Technology watch

A rigorous technology watch is essential for a compiler project like Glu. We continuously monitor the evolution of major compiler frameworks—most notably LLVM, the industry-standard toolchain maintained and widely adopted by organizations such as Apple. Staying abreast of LLVM's roadmap, feature additions, and deprecations ensures that Glu can leverage cutting-edge optimizations, maintain ABI compatibility, and anticipate upstream changes before they impact our users.

To deepen our insights, one of our core team members, Emil—who previously worked on Apple's Swift compiler—was invited to the 2024 LLVM Developers' Meeting. His first-hand exposure to design discussions and roadmap planning gives us a direct line into forthcoming LLVM enhancements and lets us feed back our own use cases. This proactive engagement allows Glu to remain aligned with the broader compiler ecosystem, adopt best practices early, and deliver a stable, high-performance toolchain to our community.

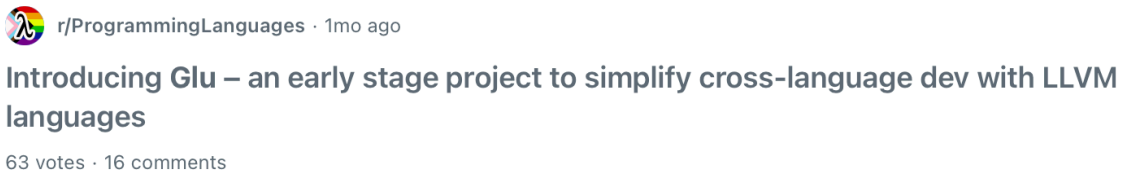
We rely heavily on LLVM's modular libraries—for lexical analysis, intermediate representation, optimization passes, code generation, and more—as the backbone of Glu's architecture. The stability, performance, and breadth of tooling that LLVM provides would be prohibitively difficult to replicate from scratch. Being invited to LLVM's meetings was truly inspiring: it not only validated our work and expertise but also opened doors to collaborate directly with the maintainers and principal contributors. That level of access has accelerated our roadmap, refined our design choices, and reinforced our confidence that Glu will interoperate seamlessly within the global compiler landscape.

2. Community involvement

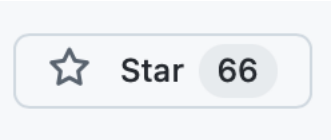
The Glu language, as an open-source project, heavily depends on building a community. In most open-source projects, a community is essential to ensure the project meets it's user's needs and stays focused on the features that make it valuable to a broad audience.

Since Glu is still in the early stages of development, we needed to begin cultivating this community to expose our project to a variety of developers and experts, and also, to gain visibility in the « market ».

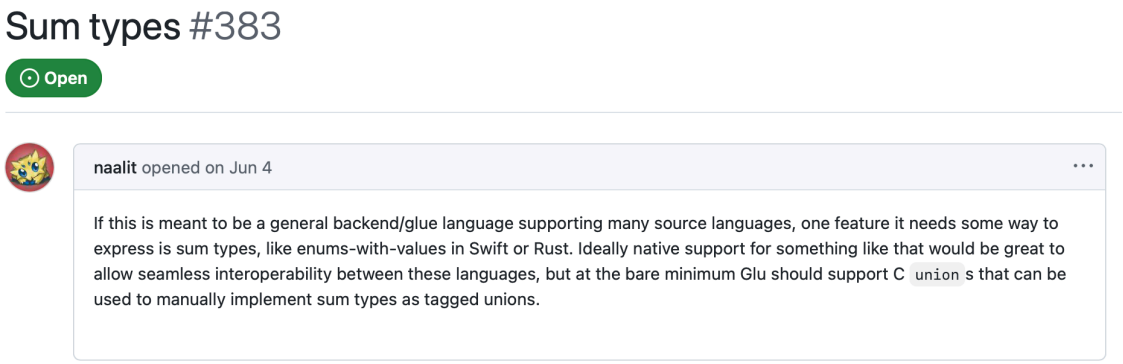
To accomplish this, we posted a detailed presentation of our project's goals and the objectives we've already achieved on Reddit, which enabled us to gather extensive global feedbacks on the project.



Thanks to this post, we collected more then 15 different feedbacks and boosted our repository's follower count from 10 to over 60 developers worldwide, forming the basis of our community.



Also, posting this introduction on Reddit enabled us to reach an audience of enthusiasts as well as experienced developers. This led to the direct involvement of members of our newly formed community in our project — for example, by creating Github issues on our repository:

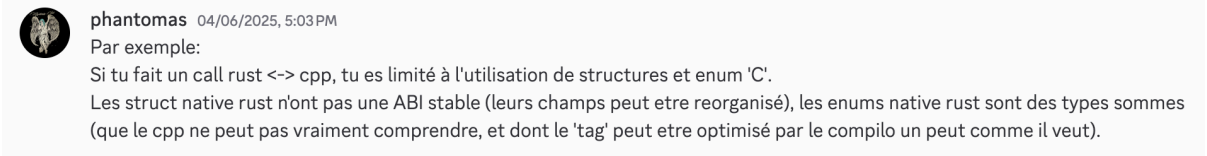


3. Exchange of experts

Given the highly technical nature of our compiler project, we recognised early on the need for deep, domain-specific feedback. To that end, we organised an expert exchange, engaging with several veteran compiler engineers—each boasting over 10 years of hands-on experience in language design, optimisation, and backend code generation.

Through a series of structured discussions, these experts helped us identify subtle pitfalls and performance bottlenecks that are easy to overlook in a greenfield compiler implementation. For example, they warned us that crossing language boundaries can lead to the loss of built-in safety guarantees (null checks, memory safety, type enforcement) and may introduce ABI incompatibilities when linking against external libraries.

They also suggested establishing a strict FFI boundary with generated bindings and incorporating automated ABI-validation tests to catch mismatches early in the build process.



Their guidance proved invaluable in refining our architecture and prioritizing development tasks. By incorporating their recommendations, we were able to avoid critical issues down the line and ensure that our compiler remains robust, efficient, and maintainable.

Appendix

Best reddit comments:

thehenkan • 1mo ago

If your interop is based on decompiling optimised IR, how do you make sure that new LLVM optimisations don't break source compatibility? Or for that matter, source compatibility between debug and release builds?

4 Reply Award Share ...

bcardiff • 1mo ago

One challenge on llvm based compilers is how to implement an incremental compiler and have faster feedback loops. Is there anything in the roadmap or ideas about how to support fast compilation and linking if you make a small change?

12 Reply Award Share ...

Zireael07 • 1mo ago

If it's really a two way street in and out of LLVM IR, this makes it an excellent idea!

Q: Does it compile to WASM?

17 Reply Award Share ...

ESHKUN • 1mo ago

This is actually super cool. I honestly really like y'all's focus on extensibility and developer interoperability. Honestly it would be cool if you could somehow make this work with more than llvm, or more even a custom backend, as this idea seems really powerful for brining languages together.

4 Reply Award Share ...