How Safe is C to Rust Translation Using LLMs?

Kenneth Fulton, Joshua Ibrom
CSCI 5362, Spring 2025

Background

- C is an older language which requires programmers to manage memory, leaving there to be the potential for memory issues such as *use after frees*, *double frees*, and *race conditions*.
- Rust is a newer systems-level programming language which boasts a “rich type system and ownership models [that] guarantee memory-safety and thread-safety” while still yielding an application that is “blazingly fast” (https://www.rust-lang.org/).
- As such, some may wonder if potentially insecure legacy applications may be re-written in Rust.

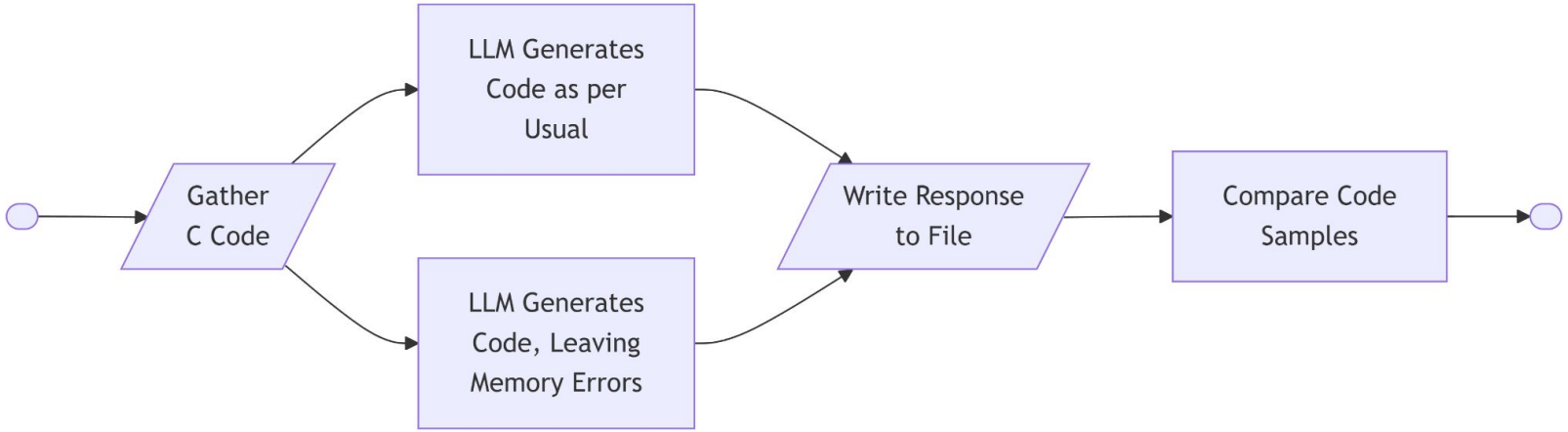
Background

- DARPA *(Defense Advanced Research Projects Agency)* has launched TRACTOR *(TRanslating All C TO Rust)*, a project in which they will use Large Language Models *(LLMs)* to convert their C code to Rust code (https://www.darpa.mil/research/programs/translating-all-c-to-rust).
  - With this project, DARPA aims to produce a Rust codebase which is of “the same quality and style that a skilled Rust developer would produce.”

Research Questions

1. How reliably can LLMs resolve memory safety errors when converting C code to Rust?
2. How efficiently can LLMs translate C code to Rust?
3. How can you prove that Rust code is safe?

Approach


Gather C Code
LLM Generates Code as per Usual
LLM Generates Code, Leaving Memory Errors
Write Response to File
Compare Code Samples

Experimental Method and Expected Outputs

https://colab.research.google.com/drive/1EnQlBhOB8hhBteE70pHg6b6issYZCwHH

Questions?