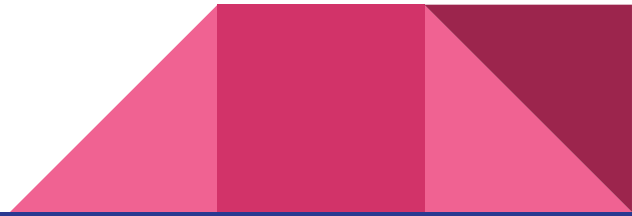
How Safe is C to Rust Translation Using LLMs?

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” [1].
- As such, some may wonder if potentially insecure legacy applications may be re-written in Rust.

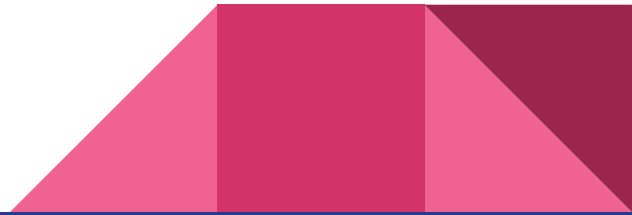


Background

- 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 [2].
    - 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.”
- Having a Rust-based codebase could then in theory result in there being far fewer (if any) chances of “exploits, crashes, or corruption” [3].

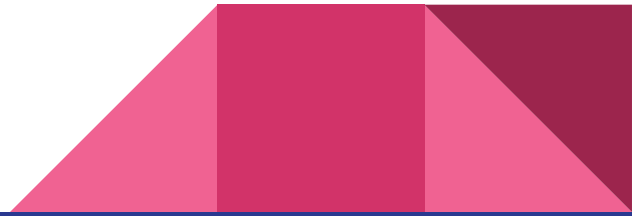
Background

- However, how can we then produce verifiably safe Rust code from unsafe C code with “relevant properties of the C code [...being expressed in] a suitable well-typed Rust program [...with] those properties” [4]?
  - For example, Li et al. [5] find that static analysis of code can only convert roughly 11% of raw C pointers to safe Rust references and that under half of small C programs considered in another report [6] “can be auto-translated to Rust using LLM-based repair.”

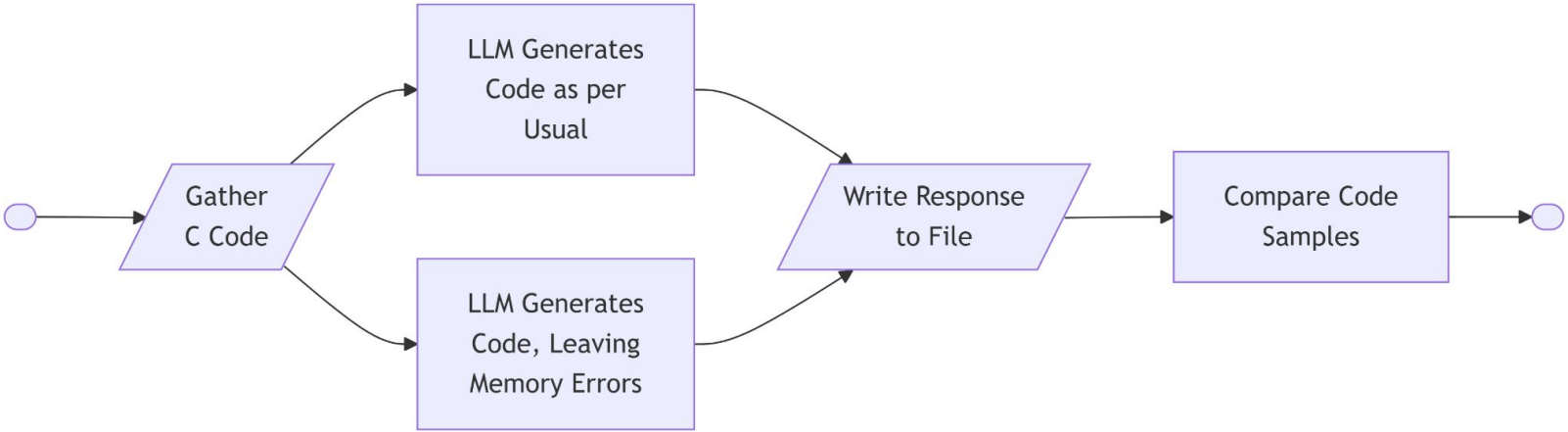


Research Questions

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

Converter Script

```
sk3p7ic@Bluey College/CSCI-5362-Term-Project (main) » python3 converter.py --help
usage: C to Rust Translator [-h] [-k] filename

C to Rust translator using ChatGPT

positional arguments:
  filename

options:
  -h, --help  show this help message and exit
  -k, --keep  Whether memory errors are desired to be kept.

For CSCI 5362, Spring 2025 @ TAMU-SA.
sk3p7ic@Bluey College/CSCI-5362-Term-Project (main) »
```

Example 1: Hello World

C Code:

```
#include <stdio.h>

int main() {
    char msg[] = "Hello, World!";
    printf("%s\n", msg);
}
```

Translated Rust Code:

```
fn main() {
    let msg = "Hello, World!";
    println!("{}", msg);
}
```

Translated Rust Code, Leaving Errors (if any):

```
fn main() {
    let msg = "Hello, World!";
    println!("{}", msg);
}
```

Example 2: Potential Overflow w/ Malloc & Strcpy

C Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
    char msg[] = "Hello, world!";
    char *buff = malloc(strlen(msg));
    strcpy(buff, msg);
    printf("%s\n", buff);
}
```

Translated Rust Code:

```
use std::ffi::CString;
use std::ptr;
fn main() {
    let msg = "Hello, world!";
    let buff = CString::new(msg).unwrap();
    let c_str = buff.as_ptr();

    unsafe {
        // Print the C-style string
        println!("{}", std::ffi::CStr::from_ptr(c_str)
                            .to_string_lossy());
    }
}
```

Example 2: Potential Overflow w/ Malloc & Strcpy

C Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
    char msg[] = "Hello, world!";
    char *buff = malloc(strlen(msg));
    strcpy(buff, msg);
    printf("%s\n", buff);
}
```

Translated Rust Code, Leaving Errors (if any):

```
use std::ffi::CString;
use std::ptr;
fn main() {
    let msg = CString::new("Hello, world!")
        .unwrap();
    let mut buff: *mut i8 = unsafe { libc::malloc(msg.to_bytes().len())
        as *mut i8 };
    unsafe {
        ptr::copy_nonoverlapping(msg.as_ptr(), buff,
                                 msg.to_bytes().len());
        println!("{}", CString::from_raw(buff).to_string_lossy());
    }
}
```

Example 3: Out-of-bounds Write (CWE 787)

C Code:

```
int main() {
  int id_sequence[3];

  /* Populate the id array. */

  id_sequence[0] = 123;
  id_sequence[1] = 234;
  id_sequence[2] = 345;
  id_sequence[3] = 456;
}
```

Translated Rust Code:

```
fn main() {
    let mut id_sequence: [i32; 4] = [0; 4]; // Change size to 4 to accommodate
the last index
    // Populate the id array
    id_sequence[0] = 123;
    id_sequence[1] = 234;
    id_sequence[2] = 345;
    id_sequence[3] = 456;
}
```

Example 3: Out-of-bounds Write (CWE 787)

C Code:

```
int main() {
  int id_sequence[3];

  /* Populate the id array. */

  id_sequence[0] = 123;
  id_sequence[1] = 234;
  id_sequence[2] = 345;
  id_sequence[3] = 456;
}
```

Translated Rust Code:

```
fn main() {
    let mut id_sequence = [0; 3];
    // Populate the id array.
    id_sequence[0] = 123;
    id_sequence[1] = 234;
    id_sequence[2] = 345;
    id_sequence[3] = 456; // This will cause a panic in Rust due to
out-of-bounds access
}
```

Results

File	Compilation Errors	Runtime/Memory Errors
cwe-125		
cwe-416		
cwe-476	yes	
cwe-787		
hello		
lslong-ji		
malloc-cpy		
mysh-ji	yes	
prodcon-ji	yes	
race		yes
unsure-cwe-125		yes
unsure-cwe-416	yes	
unsure-cwe-476	yes	
unsure-cwe-787		yes
unsure-hello		
unsure-lslong-ji	yes	
unsure-malloc-cpy		
unsure-mysh-ji	yes	
unsure-prodcon-ji	yes	
unsure-race		yes

Safe prompt

6 of 10 Executed successfully
3 of 10 Failed to compile
1 of 10 Contained runtime memory error

Unsafe Prompt

2 of 10 Executed successfully
5 of 10 Failed to compile
3 of 10 Contained runtime memory error

LLM Code Conversion Capabilities

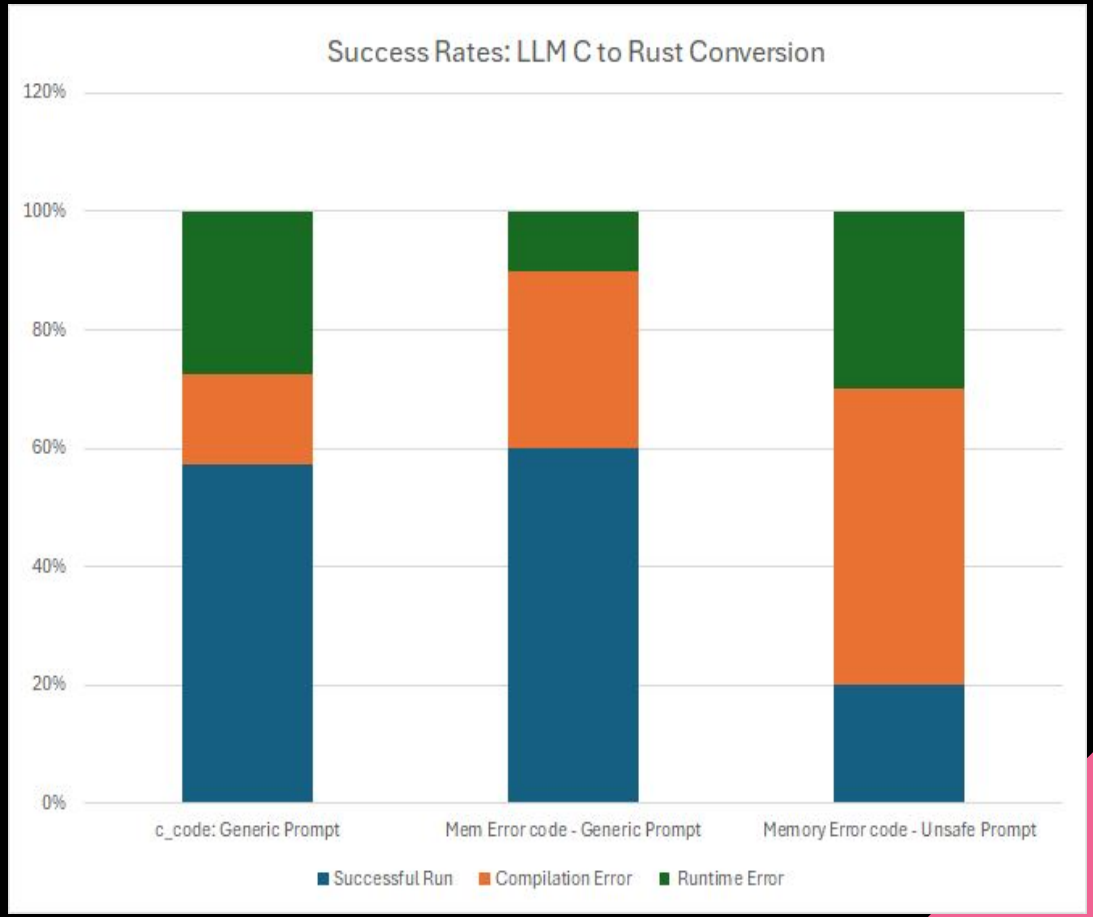
- After evaluating LLM conversion of memory error infested c code, we tested sample set of small .c files (15-100 lines of code) [7].

- 86/150 .rs files successfully ran, while others experience compilation and runtime errors.

```
Running binary: CelciusToKelvinConv
Output of binary CelciusToKelvinConv:
Enter the desired temperature in Celsius:
Converted value: 281.15 K

Running binary: SquareRoot
Output of binary SquareRoot:
2.8284271250498643

Running binary: Addition
Execution failed for binary Addition:
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.00s
     Running `target/debug/Addition`

thread 'main' panicked at Addition.rs:15:17:
index out of bounds: the len is 1 but the index is 1
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace


==================================================
EXECUTION SUMMARY
==================================================
Total programs: 150
Successfully ran: 86
Compilation errors: 23
Runtime errors: 41
==================================================
```

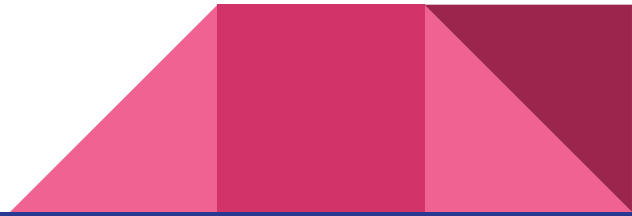
https://github.com/gouravthakur39/beginners-C-program-examples

Results


Success Rates: LLM C to Rust Conversion
120%
100%
80%
60%
40%
20%
0%
c_code: Generic Prompt
Mem Error code - Generic Prompt
Memory Error code - Unsafe Prompt
Successful Run
Compilation Error
Runtime Error

Observations

- **Common Issues**:
  - Mismatched types, usually for function arguments and return values.
  - Missing dependencies - several files attempted to use crates like libc, chrono, and users, which were not properly linked or included in the Cargo.toml file.
  - Runtime panics due to memory, index out-of-bounds, or thread-related issues.
- **Warnings**:
  - Many files have unused imports, unused variables, or deprecated method usage.



References

[1] Rust Programming Language, *Rust Programming Language*. 2025. [Online]. Available: https://www.rust-lang.org

[2] M. Emre, R. Schroeder, K. Dewey, and B. Hardekopf, “Translating C to safer Rust,” *Proceedings of the ACM on Programming Languages*, vol. 5, no. OOPSLA, pp. 1–29, 2021.

[3] S. Klabnik and C. Nichols, *The Rust Programming Language*. USA: No Starch Press, 2018.

[4] DARPA, *Translating All C to Rust*. 2025. [Online]. Available: https://www.darpa.mil/research/programs/translating-all-c-to-rust

[5] M. Emre, P. Boyland, A. Parekh, R. Schroeder, K. Dewey, and B. Hardekopf, “Aliasing limits on translating C to safe Rust,” *Proceedings of the ACM on Programming Languages*, vol. 7, no. OOPSLA1, pp. 551–579, 2023.

[6] H. F. Eniser et al., “Towards translating real-world code with LLMs: A study of translating to Rust,” *arXiv preprint arXiv:2405.11514*, 2024.

[7] G. Thakur, "beginners-C-program-examples," GitHub repository, 2019. [Online]. Available: https://github.com/gouravthakur39/beginners-C-program-examples

Questions?