LLMs are Useful for Small Problems

Mike Dodds - *HCSS* - 6 May 2024

galois

Context: Galois / me

Galois: *R&D consulting*

- Security / reliability technologies (formal methods, static analysis, crypto)
- Clients: DARPA, US Gov, some commercial

Me: *logic, automated reasoning, FM + real-world systems development*

- 2004 → 2017: York / Cambridge / York – UK PhD, postdoc, junior professor
- 2017 → now: Galois principal scientist

Context: I am not an AI expert

Me:

- Formal methods expert
- AI/ML idiot

Actual AI experts did the heavy lifting:

- Walt Woods
- Adam Karvonen
- Max von Hippel

Opinion: generative AI isn't very useful, yet

- Generative AI / LLM is a huge deal, maybe dramatically world-changing
- V democratic: pay $20/month for the world's most capable model
- It's easy to make mind-boggling demos

BUT:

- Today, May 2024: ~zero useful tools *(… & I'm looking forward to your talks)*

This talk:

What are LLMs useful for *today*

for *me*

for *small problems* encountered at Galois

Small problems

- Problem 1: Memory Skeleton Discovery
- Problem 2: Rust Macro Refolding
- Problem 3: RFC Protocol Modelling

Applying GPT-4 to SAW Formal Verification, Adam Karvonen
https://galois.com/blog/2023/08/applying-gpt-4-to-saw-formal-verification/

SAW: formal verification for cryptography & other things

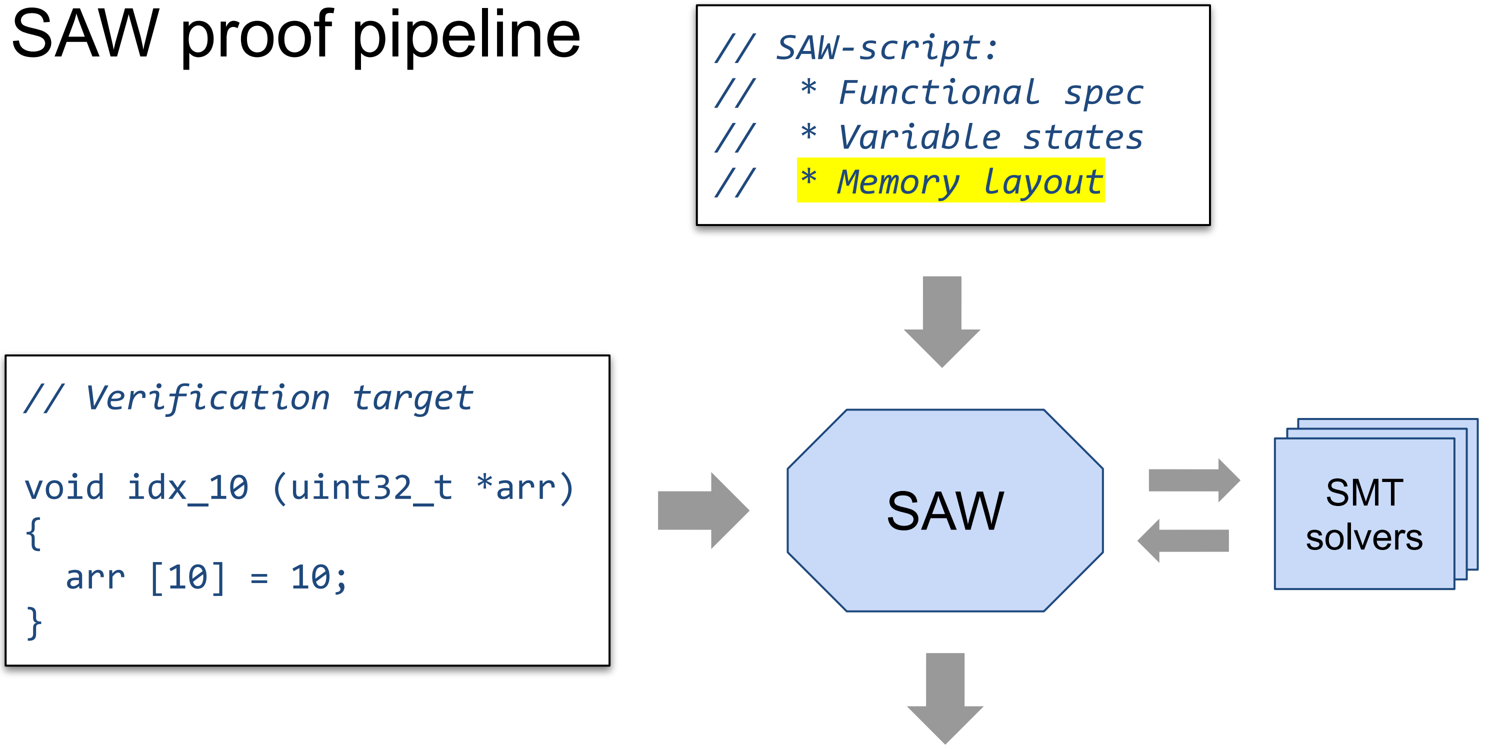
Developed by Galois over ~20 years

Deployed in US + other gov, and industry

Public stuff:

- AWS LibCrypto - verified industry crypt library covering AES, SHA, EC, …
- Supranational - verified BLST signature library for blockchain applications

SAW proof pipeline



```
// SAW-script:
//   * Functional spec
//   * Variable states
//   * Memory Layout
```

```
// Verification target

void idx_10 (uint32_t *arr)
{
  arr [10] = 10;
}
```

Result: verified / failed
(memory safe, functionally correct, …)

Skeletons define the memory layout

```
// SAW-script:
//   * ...
//   * Memory layout

idx_10_skel <- function_skeleton MODULE_SKEL "idx_10" ;
idx_10_skel <- skeleton_resize_arg idx_10_skel "arr" ??? true;
let idx_10_spec = do {
  skeleton_globals_pre MODULE_SKEL;
  prestate <- skeleton_prestate idx_10_skel;
  skeleton_exec prestate;
  poststate <- skeleton_poststate idx_10_skel prestate;
  skeleton-globals-post MODULE_SKEL;
} ;
idx_10_override <- llvm_verify MODULE "idx_10" [] false idx_10_spec
z3 ;
```

What *size of memory* does the function need to execute

Problem: find a sufficient memory size

Solution 1: *think hard, eyeball the code*

- Annoying, time consuming, hard for beginners

Solution 2: *write a fancy static analysis*

- Eg. an abductive analysis such as *Infer*
- Complex, costly, unpredictable, v partial code coverage

Solution 3: *“Dang it, I’ll just ask the LLM”*

- Easy, cheap, stupid
- Works!

BUT: the LLM doesn’t know SAW-Script

Rule of thumb:

- Big public dataset ⇒ high level of LLM capability
- Small/ zero public dataset ⇒ low level of LLM capability

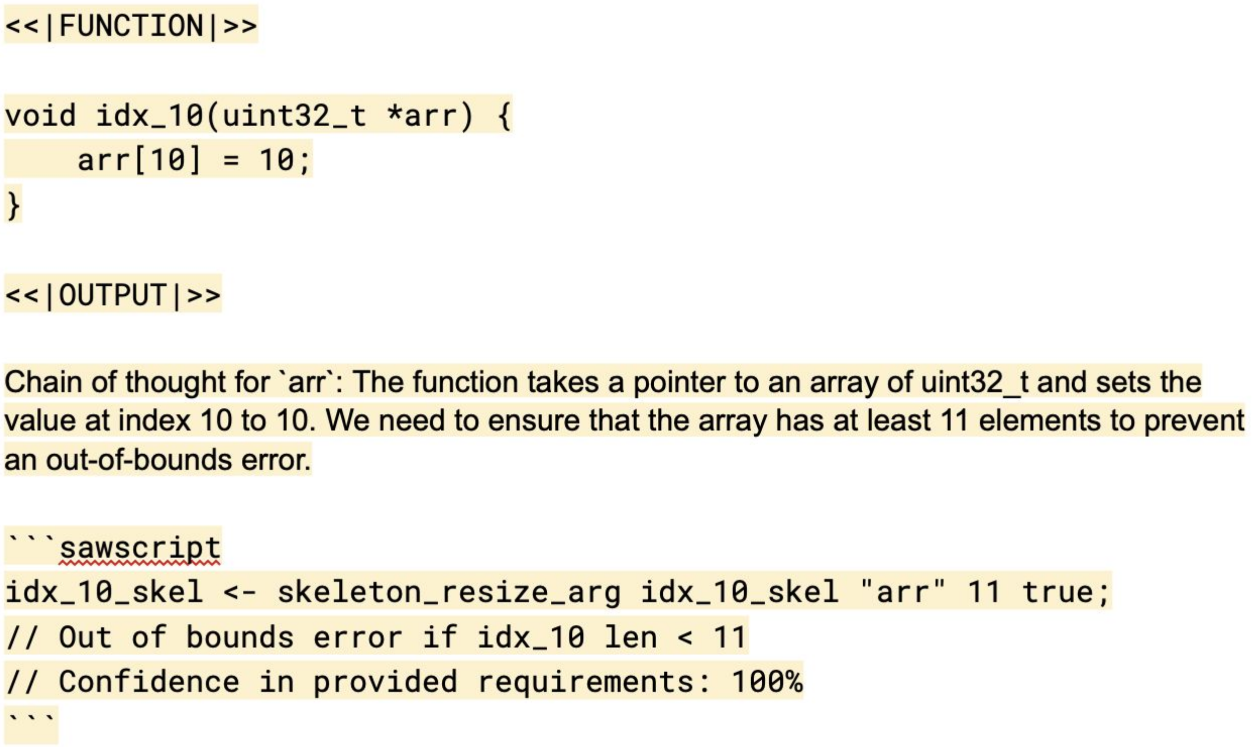
There are vv few examples of SAW-script in public :(

Our approach:

- Teach the LLM how to respond through *few-shot prompting*
- Lean on the LLM’s strong capabilities with C code

Few-shot prompting: teach the AI by example

Teaching prompt:



````
<<|FUNCTION|>>

void idx_10(uint32_t *arr) {
    arr[10] = 10;
}

<<|OUTPUT|>>
````

Chain of thought for `arr`: The function takes a pointer to an array of uint32_t and sets the value at index 10 to 10. We need to ensure that the array has at least 11 elements to prevent an out-of-bounds error.

~~~
```sawscript
idx_10_skel <- skeleton_resize_arg idx_10_skel "arr" 11 true;
// Out of bounds error if idx_10 len < 11
// Confidence in provided requirements: 100%
```
~~~

Now, let’s try it out…

```
enum s20_status_t s20_crypt32(uint8_t *key, uint8_t nonce[static 8],
uint32_t si, uint8_t *buf, uint32_t buflen)
{
  …
  uint8_t n[16] = {0};

  …
  for (i = 0; i < 8; ++i)
    n[i] = nonce[i];
```

Result:

Chain of thought for `nonce`: The function takes an array of uint8_t with a static size of 8. We need to ensure that the array has at least 8 elements as indicated by the declaration.

~~~
```sawscript
s20_crypt32_skel <- skeleton_resize_arg s20_crypt32_skel "nonce" 8
true;
// declaration indicates minimum of 8 elements
// Confidence in provided requirements: 95%
```
~~~

It’s not quite that easy

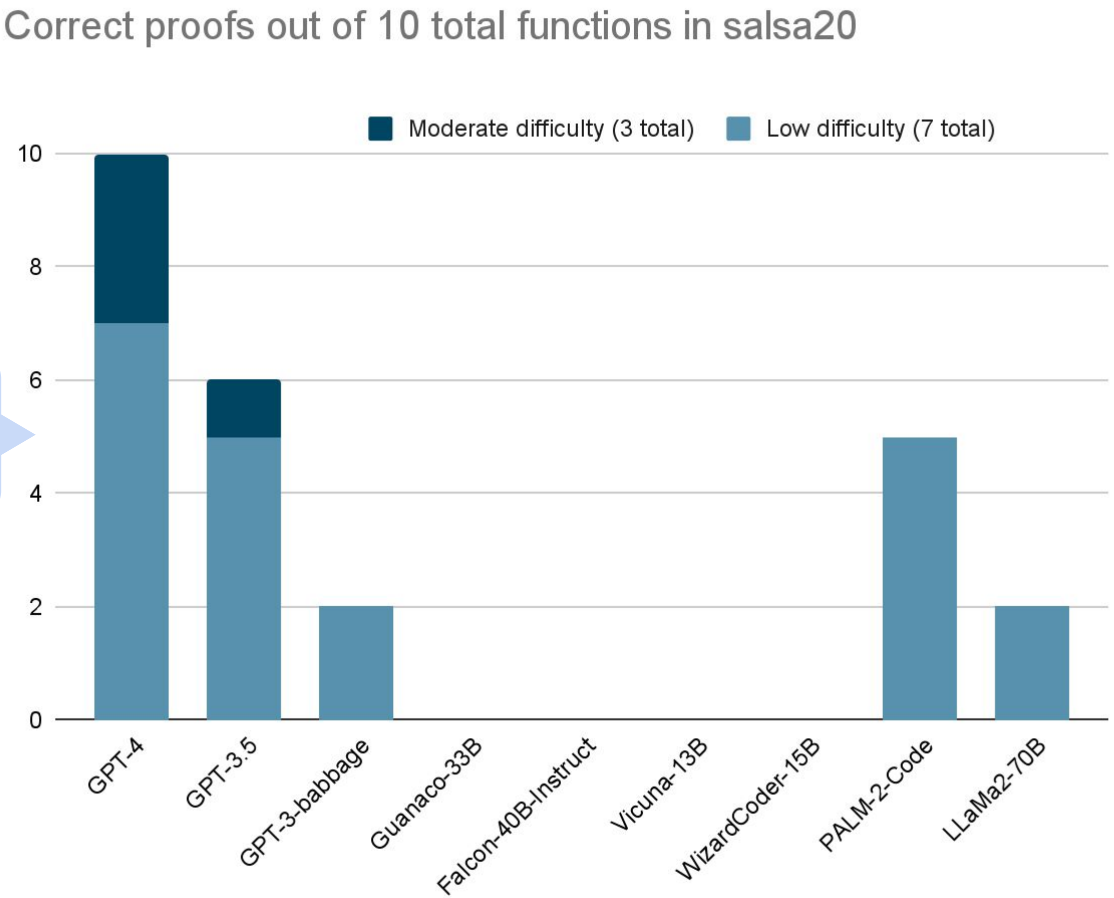
Input:

- We have to carve up the program into prompt-size chunks
- The LLM behaviour is v sensitive to the prompt (but less so with GPT-4!)

Output:

- Parse the results
- Deal with cases where the LLM returns non-useful output
- Suggestion might be wrong (aka the hallucination problem)

Results

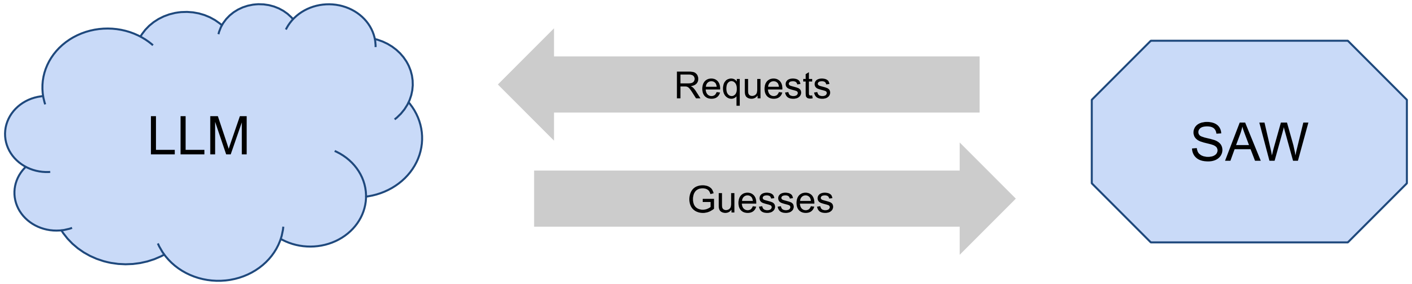


Note: GPT-4 much stronger than the rest *(c. August last year)*

Why this works: SAW and the LLM form a…

neuro - symbolic loop

≈ *AI thing* ≈ *formal thing* ≈ *loop*



Guess a memory size
- Might be wrong
- Might not answer

Check the answer
- Formal proof
- Pass == “valid guess”

Many formal methods problems are just *search*

Guess		***Check***
Write memory skeleton sizes	→	Check the sizes are correct
Add types to a program	→	Typecheck the program
Write loop invariants	→	Check the proof is valid
Synthesize a program that matches a specification	→	Check the program matches the specification
[LLM generator]	→	**[Formal methods checker]**

More details

Applying GPT-4 to SAW Formal Verification, Adam Karvonen

https://galois.com/blog/2023/08/applying-gpt-4-to-saw-formal-verification/

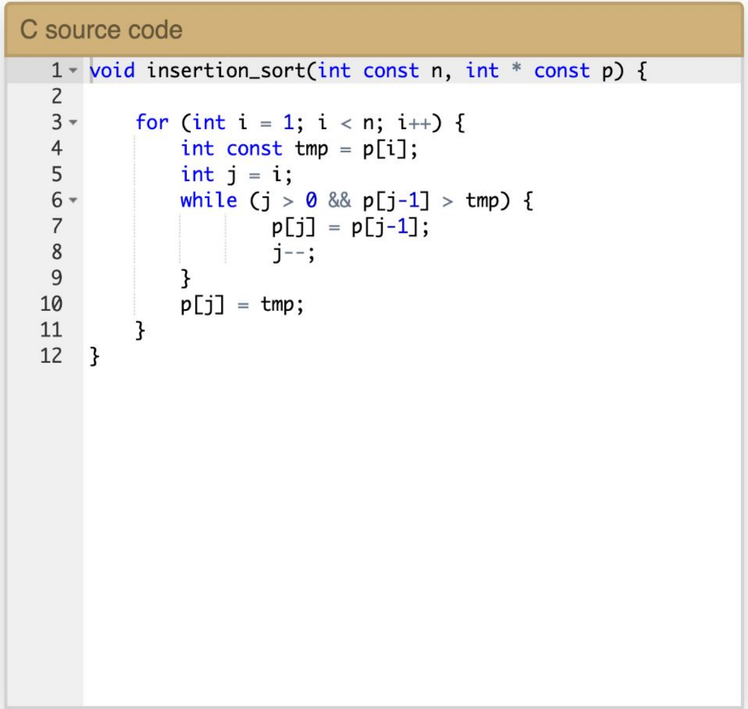
Small problems

- Problem 1: Memory Skeleton Discovery
- Problem 2: Rust Macro Refolding
- Problem 3: RFC Protocol Modelling

Using GPT-4 to Assist in C to Rust Translation, Adam Karvonen

https://galois.com/blog/2023/09/using-gpt-4-to-assist-in-c-to-rust-translation/

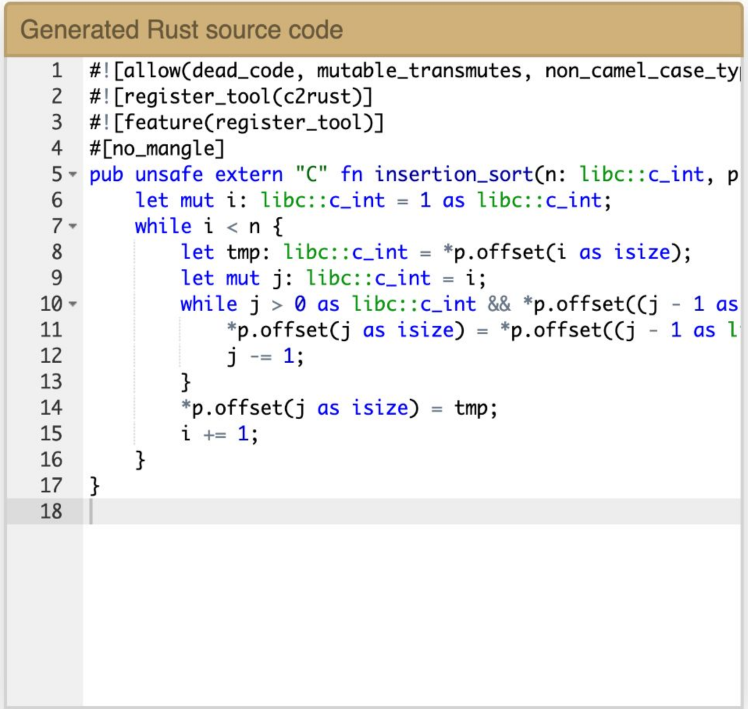
C2Rust: a transpiler from C to (unsafe) Rust



C source code

```c
void insertion_sort(int const n, int * const p) {

    for (int i = 1; i < n; i++) {
        int const tmp = p[i];
        int j = i;
        while (j > 0 && p[j-1] > tmp) {
                p[j] = p[j-1];
                j--;
        }
        p[j] = tmp;
    }
}
```

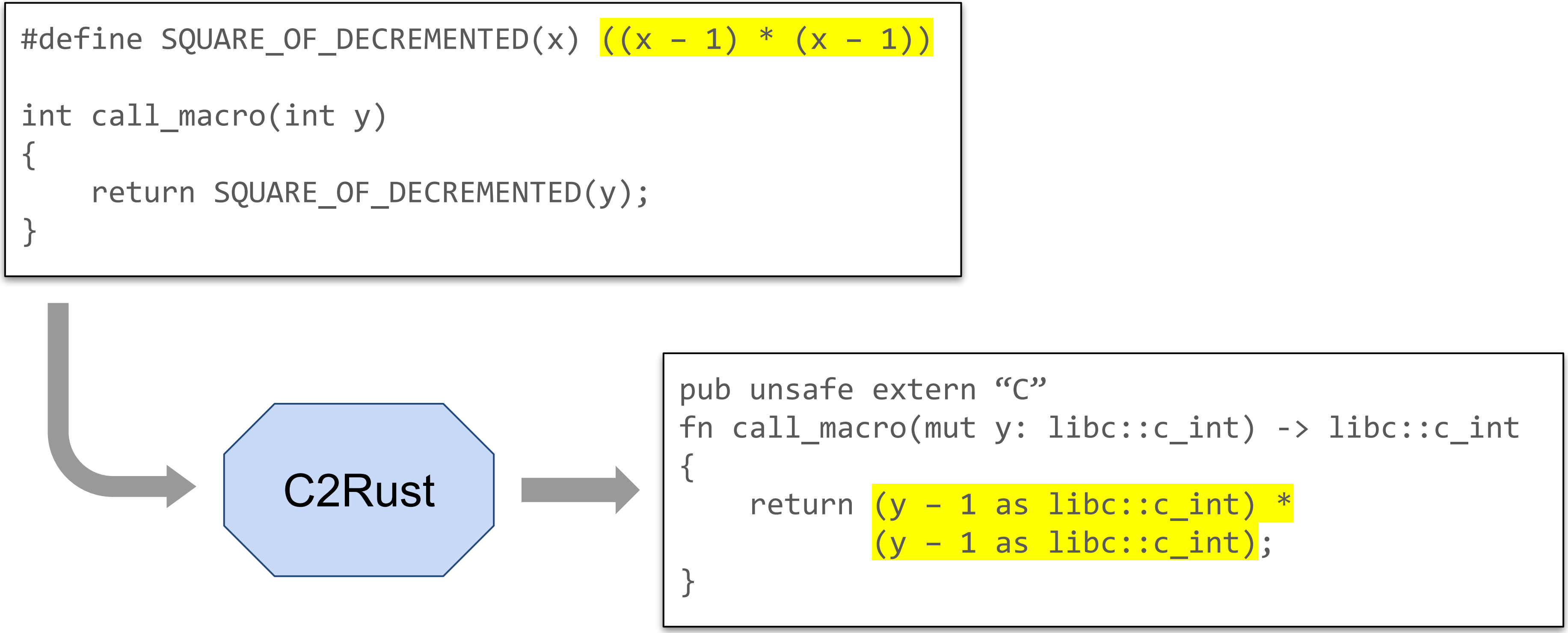


Generated Rust source code

```rust
#![allow(dead_code, mutable_transmutes, non_camel_case_ty
#![register_tool(c2rust)]
#![feature(register_tool)]
#[no_mangle]
pub unsafe extern "C" fn insertion_sort(n: libc::c_int, p
    let mut i: libc::c_int = 1 as libc::c_int;
    while i < n {
        let tmp: libc::c_int = *p.offset(i as isize);
        let mut j: libc::c_int = i;
        while j > 0 as libc::c_int && *p.offset((j - 1 as
            *p.offset(j as isize) = *p.offset((j - 1 as l
            j -= 1;
        }
        *p.offset(j as isize) = tmp;
        i += 1;
    }
}
```

Try it yourself! http://c2rust.com

Problem: C2Rust clobbers C macros



```
#define SQUARE_OF_DECREMENTED(x) ((x – 1) * (x – 1))

int call_macro(int y)
{
    return SQUARE_OF_DECREMENTED(y);
}
```

C2Rust

```
pub unsafe extern “C”
fn call_macro(mut y: libc::c_int) -> libc::c_int
{
    return (y – 1 as libc::c_int) *
           (y – 1 as libc::c_int);
}
```

C programmers really love macros!

Extreme example: 4k loc C program → 24k loc after C2Rust *(6x increase!)*

Our test application: `rav1d` (video codec)

- 953loc in C, 4303 loc in Rust (*4.5x increase*, mostly macros)
- 20 different macros used 85 different times
- Longest macro was 45 lines in the original C codebase

Problem: refold the macros

Solution 1: *think hard, rewrite the code*

- Annoying, time consuming, hard for beginners

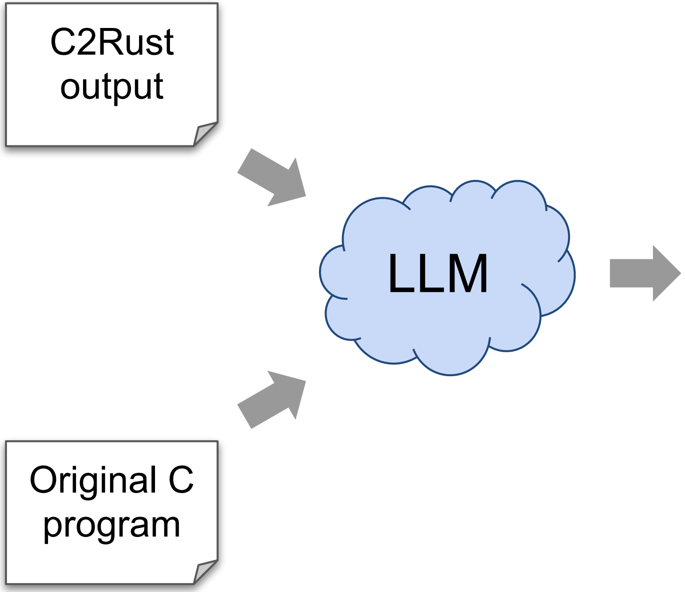
Solution 2: *write a fancier transpiler*

- Complex, costly, unpredictable

Solution 3: *“Dang it, I’ll just ask the LLM”*

- Easy, cheap, stupid
- Works!

Ideal behavior: fold the macro back into the Rust code



C2Rust output

Original C program

LLM

```
macro_rules! square_of_decremented {
    ($x:expr) => {
        ($x – 1 as libc::c_int) *
        ($x – 1 as libc::c_int)
    };
}

pub unsafe extern “C”
fn call_macro(mut y: libc::c_int) -> libc::c_int {
    return square_of_decremented!(y);
}
```

Again: guess-and-check / N-S loop

Guess: two-phase process to generate / insert macros

- Prompt with original code + C2Rust version → output: candidate macro
- Prompt with C2Rust code + macro → output: code with folded macros

Check:

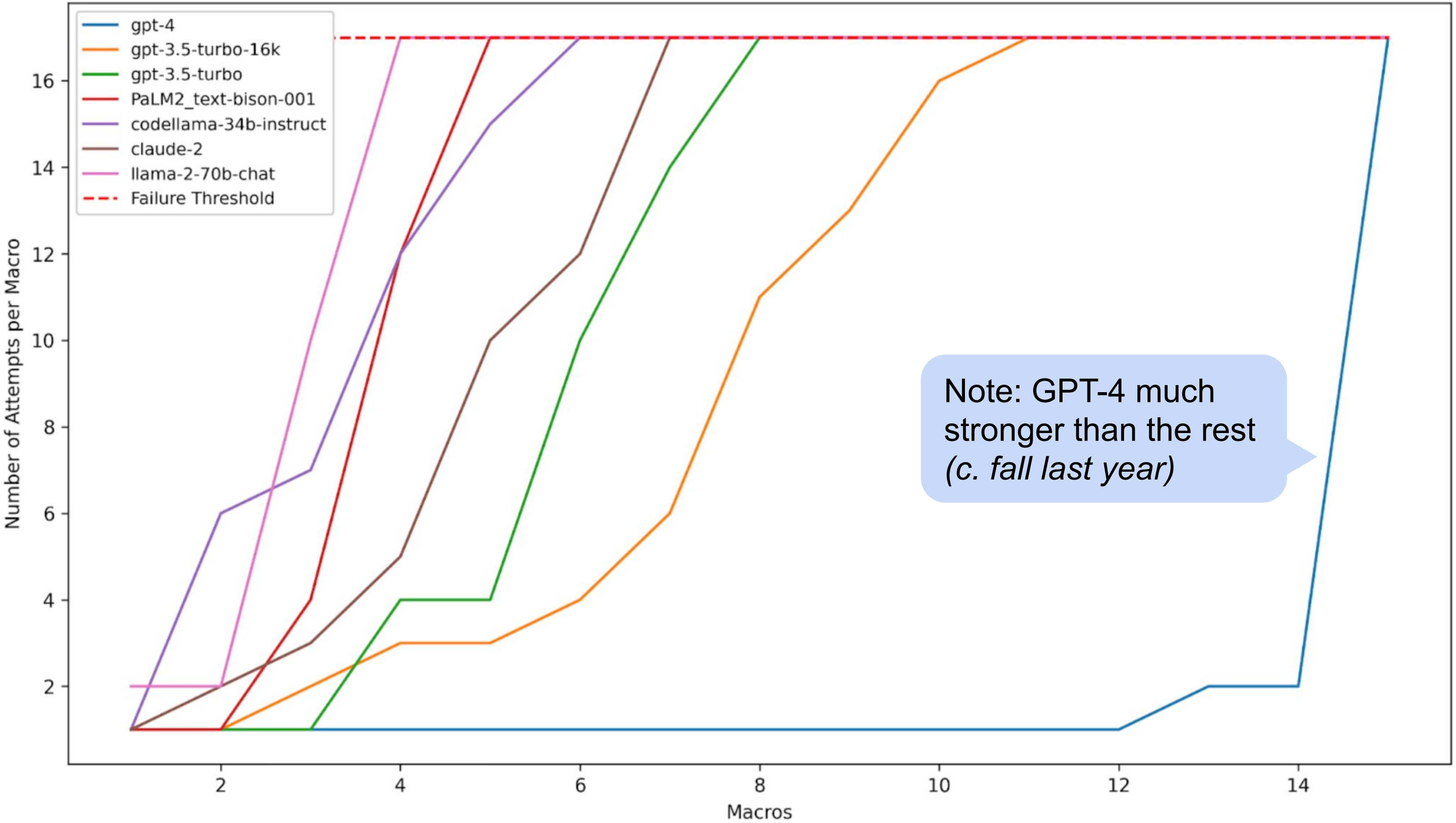
- *Insight:* the original and folded code should have the same compiled form
- Compile the function to HIR (Rust compiler IR)
- Equal HIR == correct folding

Result: the LLM can refold macros!

Test application: `mc_tmpl.rs`, a file from the `rav1d` codebase

Results:

- All 20 macros successfully constructed
- Inserted 46 out of the 60 possible macro usages
- File length decreased by 1,600 lines
- 2,900 lines were deleted or rewritten



Note: GPT-4 much stronger than the rest *(c. fall last year)*

First success for a range of example macros (bottom right is better)

More details:

Using GPT-4 to Assist in C to Rust Translation, Adam Karvonen

https://galois.com/blog/2023/09/using-gpt-4-to-assist-in-c-to-rust-translation/

Small problems:

- Problem 1: Memory Skeleton Discovery
- Problem 2: Rust Macro Refolding
- Problem 3: RFC Protocol Modelling

Coupling LLMs with FM for RFC Analysis, Max von Hippel
Galois white paper *(ask me for a copy)*

Protocols are specified in RFCs

Eg. TLS, TCP, and many many others

Varied content and structure:

- pseudocode,
- finite state machine (FSM) diagrams
- message sequence charts (MSCs),
- packet structure diagrams,
- structured text (with if/then statements and semantical indentation),
- mathematical formulae
- plain English

We’d like to have formal specifications of protocols

ASCII RFCs are:

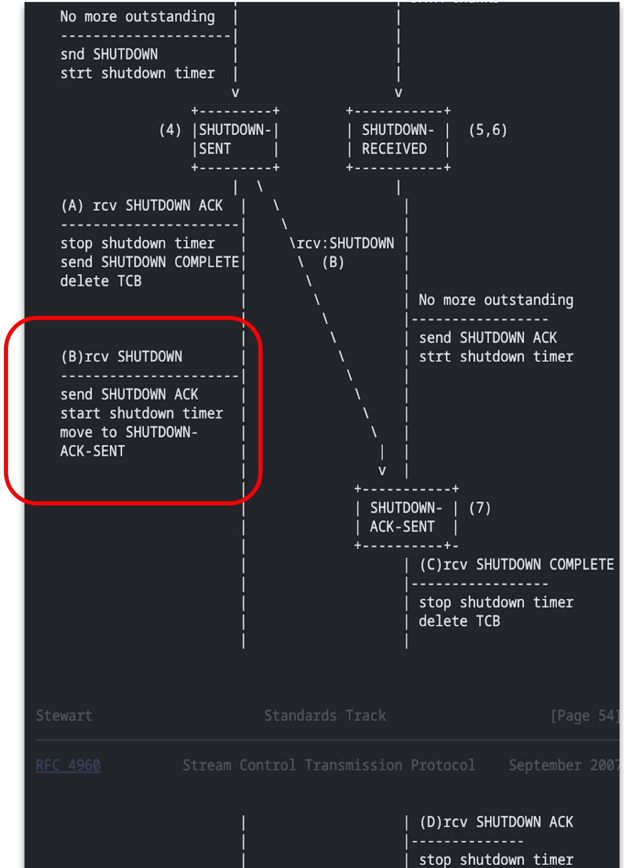
- Untestable
- Ambiguous

Formal model (Tamarin / Spin / Promela / AC2 / Coq …)

- Unambiguous
- Testable / verifiable
- A tool for reaching agreement with human protocol designers (maybe?)

BUT: current RFCs are messy and ambiguous ☹️

Graphical ambiguity in RFC 4960 (left), partially resolved in RFC 9260 (right).

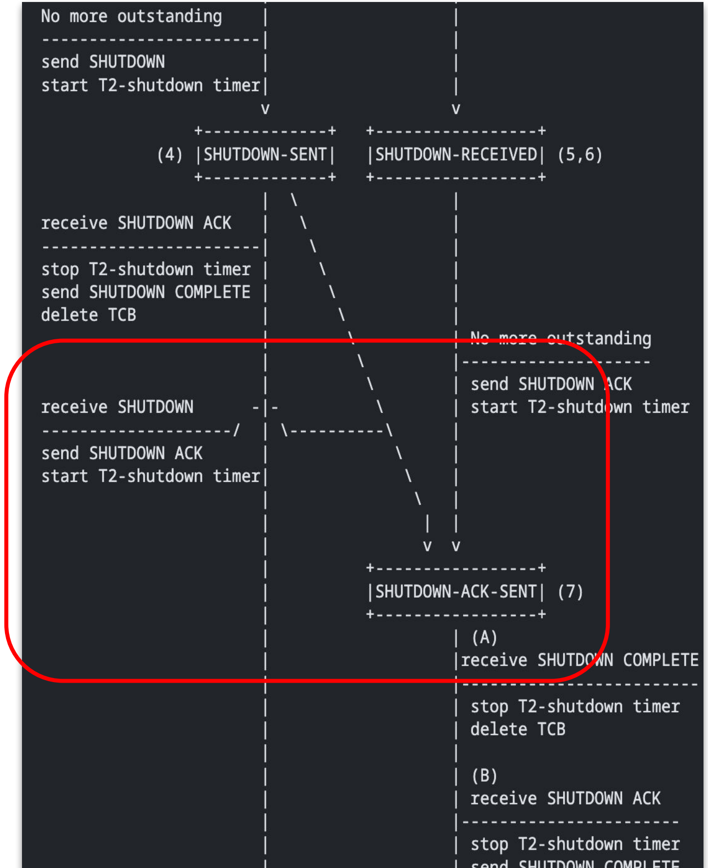


```
   No more outstanding  |                   |
   ---------------------|                   |
   snd SHUTDOWN         |                   |
   strt shutdown timer  |                   |
                        v                   v
                   +---------+        +-----------+
               (4) |SHUTDOWN-|        | SHUTDOWN- |  (5,6)
                   |SENT     |        | RECEIVED  |
                   +---------+        +-----------+
                        |  \                |
   (A) rcv SHUTDOWN ACK  |   \               |
   ----------------------|    \              |
   stop shutdown timer   |     \rcv:SHUTDOWN |
   send SHUTDOWN COMPLETE|      \  (B)       |
   delete TCB            |       \           |
                         |        \          | No more outstanding
                         |         \         |-----------------
                         |          \        | send SHUTDOWN ACK
   (B)rcv SHUTDOWN       |           \       | strt shutdown timer
   ----------------------|            \      |
   send SHUTDOWN ACK     |             \     |
   start shutdown timer  |              \    |
   move to SHUTDOWN-     |               \   |
   ACK-SENT              |                |  |
                         |                v  |
                         |             +-----------+
                         |             | SHUTDOWN- | (7)
                         |             | ACK-SENT  |
                         |             +----------+-
                         |                   | (C)rcv SHUTDOWN COMPLETE
                         |                   |-----------------
                         |                   | stop shutdown timer
                         |                   | delete TCB
                         |                   |


Stewart                     Standards Track                    [Page 54]

RFC 4960          Stream Control Transmission Protocol    September 2007

                         |                   | (D)rcv SHUTDOWN ACK
                         |                   |--------------
                         |                   | stop shutdown timer
```



```
No more outstanding    |                   |
-----------------------|                   |
send SHUTDOWN          |                   |
start T2-shutdown timer|                   |
                       v                   v
                +-------------+   +-----------------+
            (4) |SHUTDOWN-SENT|   |SHUTDOWN-RECEIVED| (5,6)
                +-------------+   +-----------------+
                       |  \                |
receive SHUTDOWN ACK   |   \               |
-----------------------|    \              |
stop T2-shutdown timer |     \             |
send SHUTDOWN COMPLETE |      \            |
delete TCB             |       \           |
                       |        \          | No more outstanding
                       |         \         |--------------------
                       |          \        | send SHUTDOWN ACK
receive SHUTDOWN      -|-          \       | start T2-shutdown timer
--------------------/  | \----------\      |
send SHUTDOWN ACK      |             \     |
start T2-shutdown timer|              \    |
                       |               \   |
                       |                |  |
                       |                v  v
                       |          +-----------------+
                       |          |SHUTDOWN-ACK-SENT| (7)
                       |          +-----------------+
                       |                   | (A)
                       |                   |receive SHUTDOWN COMPLETE
                       |                   |-------------------------
                       |                   | stop T2-shutdown timer
                       |                   | delete TCB
                       |                   |
                       |                   | (B)
                       |                   | receive SHUTDOWN ACK
                       |                   |-----------------------
                       |                   | stop T2-shutdown timer
                       |                   | send SHUTDOWN COMPLETE
```

Problem: write a formal model

Solution 1: *think hard, write the model*

- Annoying, time consuming, hard for beginners

Solution 2: *write a fancy parser for RFCs*

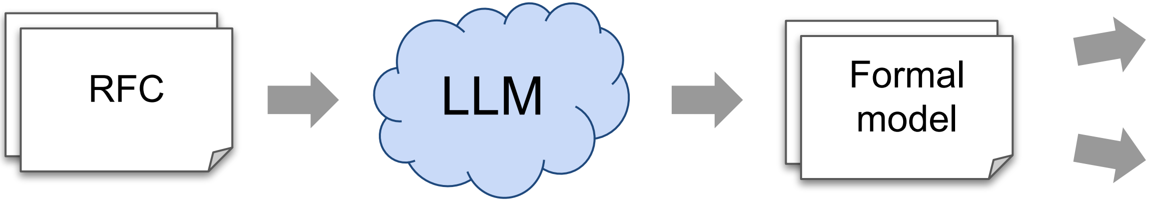
- Complex, costly, unpredictable

Solution 3: *“Dang it, I’ll just ask the LLM”*

- Easy, cheap, stupid
- ~~Works!~~

… er, it works surprisingly well, but not perfectly.

Ideal result: LLM turns the RFC into formal model



Proof tools …

Code generators …

Test generators …

- ACL2
- Protobuf
- Tamarin
- Spin
- …

RFCs are varied → *many small experiments*

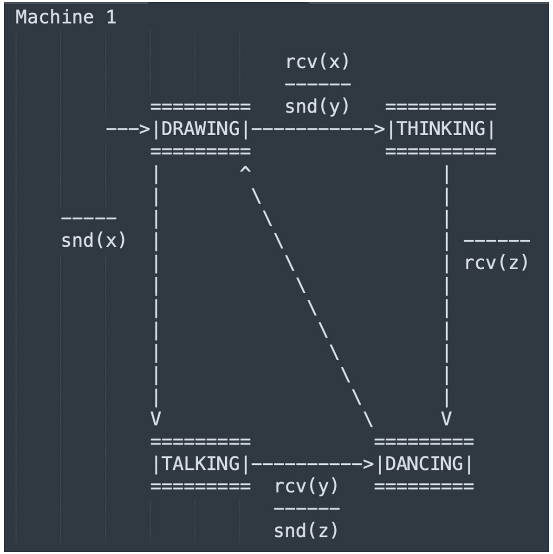
Example 1: Synthetic protocol diagram → ACL2

Input: PNG of an ASCII protocol (see right)

Output: an ACL2 model

- Protocol diagram → ACL2: close, but not perfect. Some human assistance needed
- *The LLM does not like diagonal arrows*

- ACL2 protocol debugging: much more successful
- Suggested protocol fixes that resolved mistakes



```
Machine 1

                        rcv(x)
                        ------
            =========   snd(y)   ==========
        --->|DRAWING|----------->|THINKING|
            =========            ==========
            |       ^                 |
            |        \                |
    -----   |         \               |
    snd(x)  |          \              | ------
            |           \             | rcv(z)
            |            \            |
            |             \           |
            |              \          |
            |               \         |
            |                \        |
            |                 \       |
            V                  \      V
            =========          =========
            |TALKING|---------->|DANCING|
            =========  rcv(y)   =========
                       ------
                       snd(z)
```

Synthetic protocol diagram

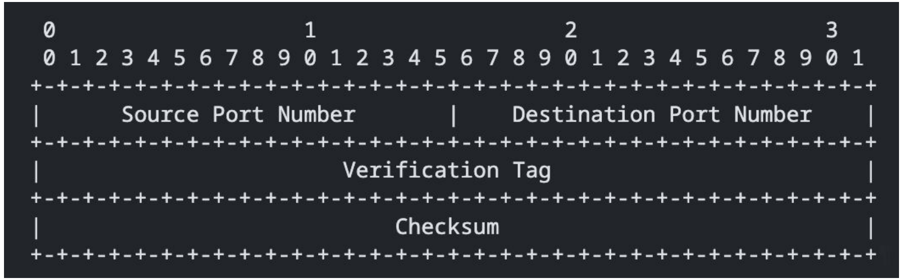
Example 2: Packet diagrams → Protobuf code

Input: PNG of packet diagram

Output: model / Protobuf code

LLMs today are bad at this task!

- Consistently misinterpreted the input, produced syntactically invalid output, or made other mistakes
- Unable to consistently count bits
- Could not produce consistently syntactically correct Protobuf code.



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Source Port Number       |    Destination Port Number    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Verification Tag                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Checksum                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Packet diagram (from RFC 9260)

Results

We experimented with ~20 RFC → model workflows

The LLM can take raw RFC text and sometimes produce close-to correct models!

Observations:

- LLM does better when tasks are split into sub-tasks
- The biggest improvements in performance come from prompt engineering
- LLM is very bad at logical reasoning and math
- GPT-4 was way better than the rest (c. winter 2023)

“What about neuro-symbolic loops, smart guy?”

RFC modelling is hard to fit into this paradigm!

- No ground truth - experts may not agree, systems may not match RFCs
- Humans needed - can’t automatically check for correctness
- Lots of intra-RFC dependencies - hard to decompose

This isn’t really a small problem, more like several big problems!

- Human-to-LLM interaction
- Closed-box testing of hypothesis models
- Merging models under ambiguous data

- ➔ Problem 1: Memory Skeleton Discovery
- ➔ Problem 2: Rust Macro Refolding
- ➔ Problem 3: RFC Protocol Modelling

… so what did we learn?

LLMs are useful for small problem

Big problems: hard to check for success, hard to control hallucinations

Small problems: LLMs can be useful!

Counterpoint: *“you’re using a huge sledgehammer to crack a tiny nut”*

BUT:

- Many tasks are ‘solved in theory’ but very fiddly to actually automate
- Many tasks are ‘easy’ but arduous for humans at scale
- If you have a sledgehammer, why not hit things with it? :D

Ideal characteristics for a ‘small’ LLM task

- Easy to check if the task was completed correctly
- Partial success is still valuable
- The input is ‘messy’ but well represented in the wild
- Task can be decomposed into small chunks
- Easy for humans in the small, but arduous thanks to quantity

Integration is a barrier

Our experiments:

- Hand-rolled python scripts to call the API and parse the results
- Hand-prompting the LLM (Ollama / ChatGPT)

Ideal future:

- Call into an LLM through a language interface
- Easy ways of parsing LLM results to data
- LLMs construct well-formed

Some speculation

- LLMs / generative AI will stay unacceptably unreliable for the near future
- We should look for guess-and-check loops at multiple scales
- We should look for problems that are ‘easy’ but arduous for humans at scale
- Many big problems contain small problems
- Grinding down the small problems will make the big problems more tractable
- Early LLM successes will often look like ‘small problems’

Thanks!

miked@galois.com

galois

Further reading

Applying GPT-4 to SAW Formal Verification, Adam Karvonen

https://galois.com/blog/2023/08/applying-gpt-4-to-saw-formal-verification/

Using GPT-4 to Assist in C to Rust Translation, Adam Karvonen

https://galois.com/blog/2023/09/using-gpt-4-to-assist-in-c-to-rust-translation/