MANCHESTER 1824
The University of Manchester

CDTS Seed Corn Showcase
11 December 2024

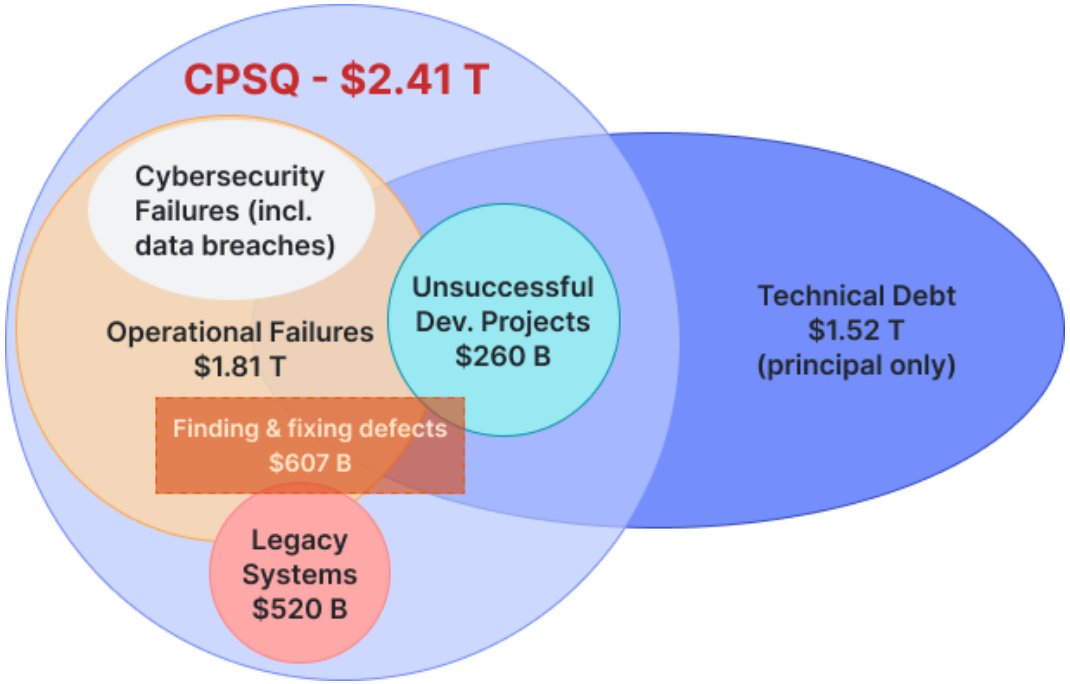
Formal Software Verification with Large Language Models in the Loop

CDTS | TDS | Dept. CompSci | S3
Mustafa A. Mustafa (PI**), Yiannis Charalambous**, Youcheng Sun, Edoardo Manino and Lucas Cordeiro

CDTS | DTC | Dept. Crim
Meropi Tzanetakis and Laura McCulloch

How much could software errors cost your business?

Poor software quality cost US companies **$2.41 trillion in 2022**, while the **accumulated software Technical Debt** (TD) has grown to **~$1.52 trillion**

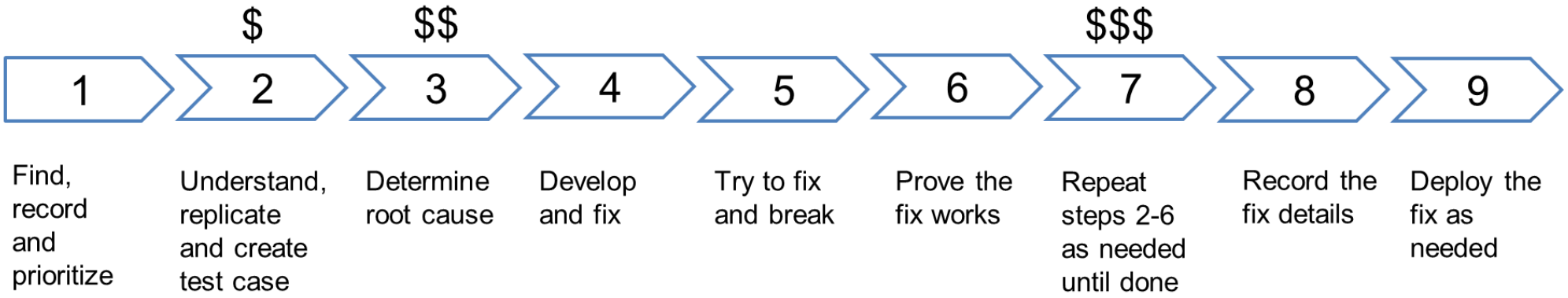


TD relies on temporary easy-to-implement solutions to achieve short-term results at the expense of efficiency in the long run

The cost of poor software quality in the US: A 2022 Report

CISQ
Consortium for Information & Software Quality™

Find, Understand and Fix Bugs



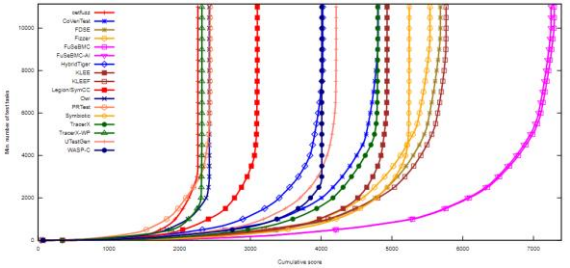
*"A significant percentage (50%+) of a **software project's cost today is not spent on the creativity activity of software construction** but rather on the **corrective activity of debugging and fixing errors**"*

The cost of poor software quality in the US: A 2022 Report

CISQ
Consortium for Information & Software Quality™

Award-Winning Tools for Building Secure Software and AI Systems

- Develop award-winning **theorem provers** and **software verifiers** to ensure safe and secure HW/SW systems:
  - Vampire and iProver (first-order theorem provers)
  - ESBMC, JBMC, and FuSeBMC (source code analysis)
  - MAMBO (binary code analysis)
- Contributions to **software verification** and **automated reasoning**: build **trustworthy software and AI systems**
- We have strong links to the industry, including collaborations with AWS, ARM, Ethereum, Intel, Microsoft, NASA, and THG



Research Questions

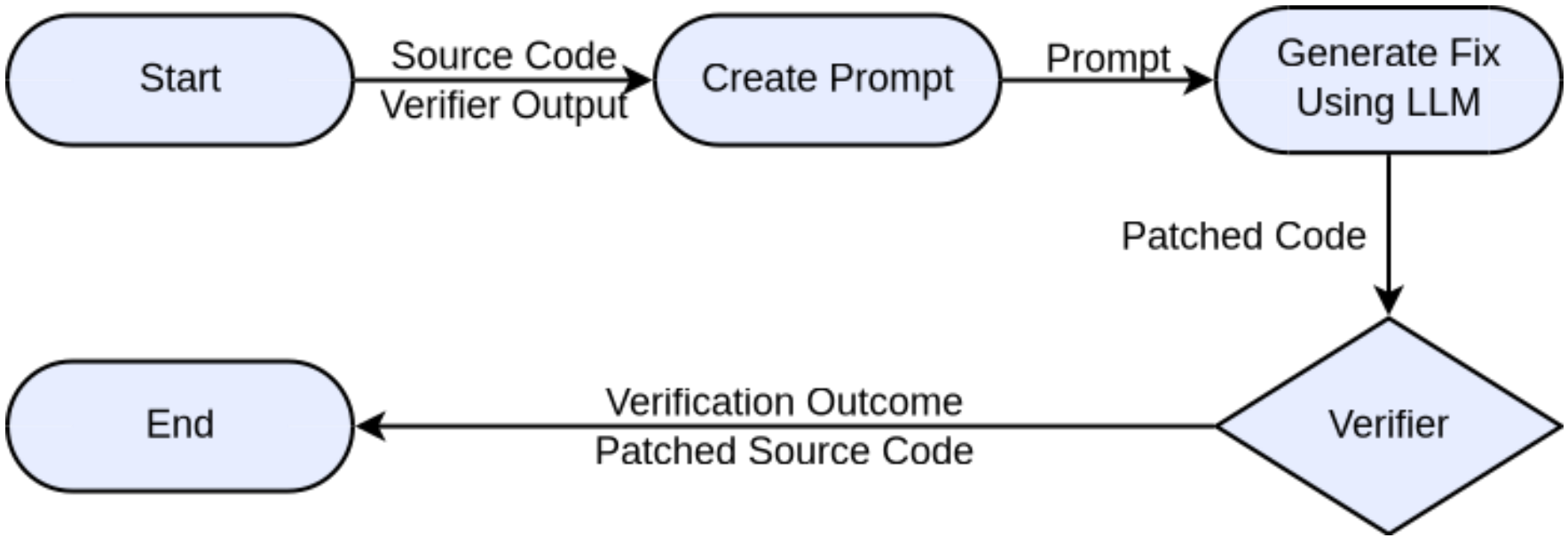
Can we combine **large language models** and **formal verification** to generate efficient, robust and secure AI code?

What are the **challenges to industry adoption** of software verification tools?

AUTOMATED PROGRAM REPAIR OF AI CODE WITH LLM AND FORMAL VERIFICATION

Objective

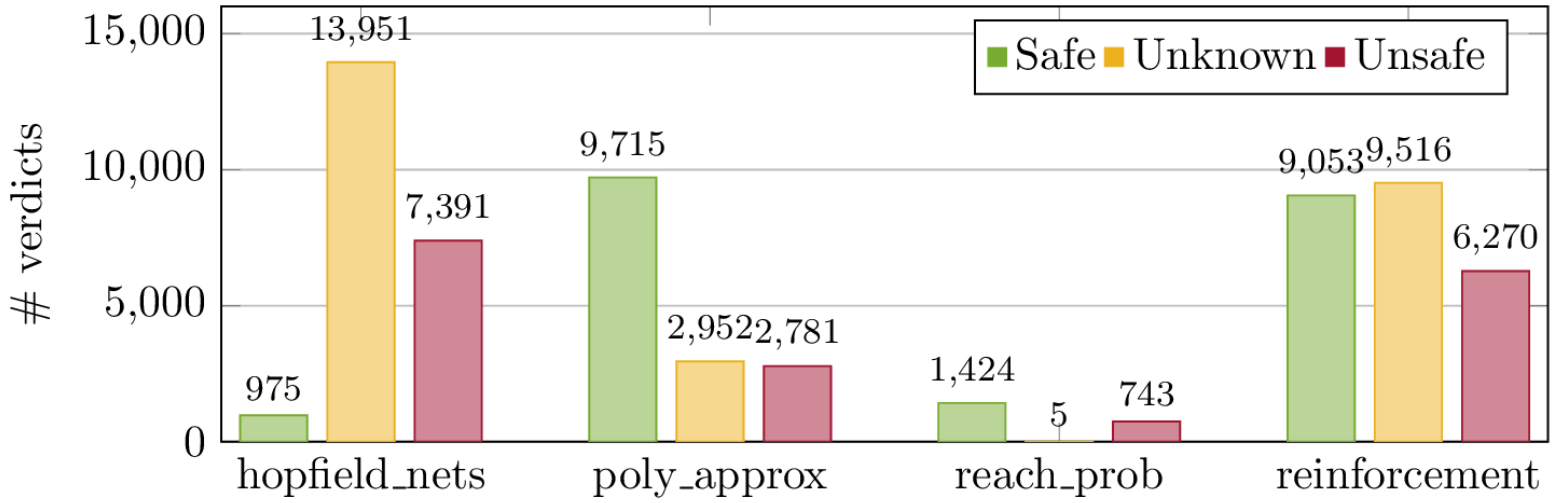
Combine **large language models** and **formal verification** to generate **efficient, robust and secure AI code**



Yiannis Charalambous, Edoardo Manino, Lucas C. Cordeiro: Automated Repair of AI Code with Large Language Models and Formal Verification. CoRR abs/2405.08848 (2024)

Dataset

Expand NeuroCodeBench from **505** to **80k** samples containing memory safety vulnerabilities



Unsafe code will be used to show the LLM problematic examples

Unsafe Code → X Train

Safe Code → Y Train

Safe code will be used to show the LLM how to fix the vulnerable code

Prompt Engineering

Use **prompt engineering** techniques to find the best prompt to use for APR of **AI C Code**

- **Best Prompt: 18% Successful Repairs**
- Showing a **Single Line** of source code is better than showing **Contextually**
- Descriptive **Persona** prompts work the best
- The best role is **Automated Code Repair Tool**
- Showing the source code after the verifier output is better
- Providing raw feedback from a verifier makes the performance worse

From now on, act as an {role} that repairs AI C code. You will be shown AI C code, along with ESBMC output. Pay close attention to the ESBMC output, which contains a stack trace along with the type of error that occurred and its location. Provide the repaired C code as output, as would an {role}. Aside from the corrected source code, do not output any other text. The ESBMC output is {esbmc} The source code is {source}

Iterative APR Framework

Build an automated framework that uses **iterative repair** to fix **AI C Code**

- **Best Prompt: 25% Successful Repairs**
- **Iterative APR** is more successful than single shot APR
- The best temperature for APR is **0.0**
- The best number of retries is **2**
- The best way to show the history of repairs to the LLM is **Forward History**

Final Verdict: Iterative APR is better than single-attempt APR

Future Work

- More LLMs, larger experiments, finetuning
- Augment formal verification (ESBMC) with testing
- Apply this APR method to open-source software projects

CHALLENGES TO INDUSTRY ADOPTION OF SOFTWARE VERIFICATION TOOLS

Social aspects: Research puzzle

- Why are software verification tools used or not used within industrial contexts?
- Qualitative research approach
- 26 semi-structured interviews with software developers and software engineers
- Exploring and understanding experiences, behaviours and meanings

Experience with tools
Not using tools: 5
Hybrid experience: 6
Currently using tools: 15

- Data: +22hrs interviews
- Approx. 450 pages of transcripts
- Qualitative Content Analysis
- Validity: findings briefing session

Research Findings: Themes

Trustworthiness

- Maturity of the tools
- Complementary tools

Scalability

- Codebase and time challenges
- Verifying in smaller sections

Usability

- Code language knowledge
- Maintenance of tools
- Need for contextual information
- Ned for education and training
- Requiring specialist knowledge
- Different code language styles

Company and Management

- Varying goals of the company
- Finance and budgets
- Labour costs
- Managerial influence
- Workload and prioritisation

Trustworthiness

Maturity of the tools

Generally, very accurate outcomes of mature automated software verification tools.

*"We normally don't use those **tools** that are quite new for the tool we are using. They already **exist** in, in the academia of almost ten years or, let's say at least 5 years. **At least some years, is almost 10 year, even 20 years now. So, they're very stable** and that is actually another criteria where we quite often assess, you know whether the two are stable or not and we need a stable tool."* (Henry, using)

Scalability

Challenges with time and codebase

The more complex your system is, the more difficult it becomes to use the tools.

"If anyone ran it on a regular basis, it took hours to run. It was something we ***ran once a week to verify that we hadn't completely broke the system****, which is good and bad. Yes, you find out eventually that you've buggered up the system, but you find out up to a week later."* (Harry, using)

Usability

Need for education and training

Documentation is sometimes hard to understand.

*"For the research tools, it could be improved, because **you always have a paper attached to the tool, and it's not so user-friendly**. **It's made from researchers to researchers.** It's been presented at the conference, so **it's quite technical**, and not all of them have a nice, like, user-friendly documentation."* (Ed, hybrid)

Company and Management

Finance and Budgets

Tool costs must be justified to management:

If they do not provide value, they will not be considered.

"I think that making the case to the business that we should do, this would be extremely difficult. Like, you would need to demonstrate why the delta between the testing we do now and the testing we could get to without formally verifying, like, the systems and the code compared to, like, whatever we gain from formal verification or software verification in general. ***We'd have to justify why that's worth the expenditure, and that's quite hard, right?*** *I think in general, you know, when it comes to justifying any spend,* ***and if it's not direct spend on building features that are up to some reasonable bar, you're always fighting a tough battle****. You know, you'll know about the many examples of tradeoffs between security and cost, and that goes on in every bit of commercial software development that exists out there."* (Rob, not using)

Recommendations

- **Engage companies** in the development of those tools
- **Knowledge sharing** within industry and between academia and industry
- **Further training** to overcome challenges regarding expert knowledge
- **Raising management-level awareness** about the benefits of the tools

Q&A

You can reach us at

Mustafa.Mustafa@mancehster.ac.uk

Lucas.Cordeiro@manchester.ac.uk

Meropi.Tzanetakis@mancehster.ac.uk

Yiannis.Charalambous-3@student.manchester.ac.uk