

FATTW 2026

Formal Verification of LLM-Generated Code

Prof. Lucas C. Cordeiro

Professor of Computer Science · Area Lead, Systems & Software Security (S3)

Director, Arm Centre of Excellence · Department of Computer Science

lucas.cordeiro@manchester.ac.uk

MANCHESTER
1824

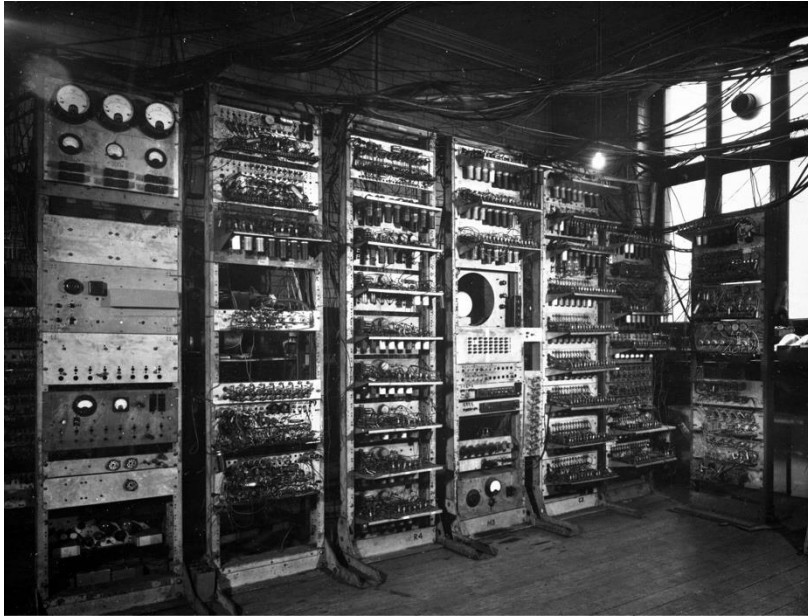
The University of Manchester



UFAM

Manchester, 1948: the first stored program

The first electronic stored-program computer, and the man who asked if programs are right.



The Manchester Baby, 21 June 1948

First electronic stored-program computer



Alan Turing

At Manchester from
September 1948 until his
death in 1954

Why do I mention the baby computer?

A stored program is data: it can be written, changed and generated by another program.

So the question of whether a program is right arrived with the first one.

A year later, Turing asked it in print.

Once programs became data, someone had to ask whether they were right.

1949: Turing asks whether a program is right

“How can one check a routine in the sense of making sure that it is right?”

A. M. Turing, “Checking a Large Routine”, Cambridge, June 1949

1949 · Turing

Answers it: assertions at points in the routine, and a separate argument that it stops.

1967–69 · Floyd, Hoare

Make the same idea a method: assertions and invariants become a logic.

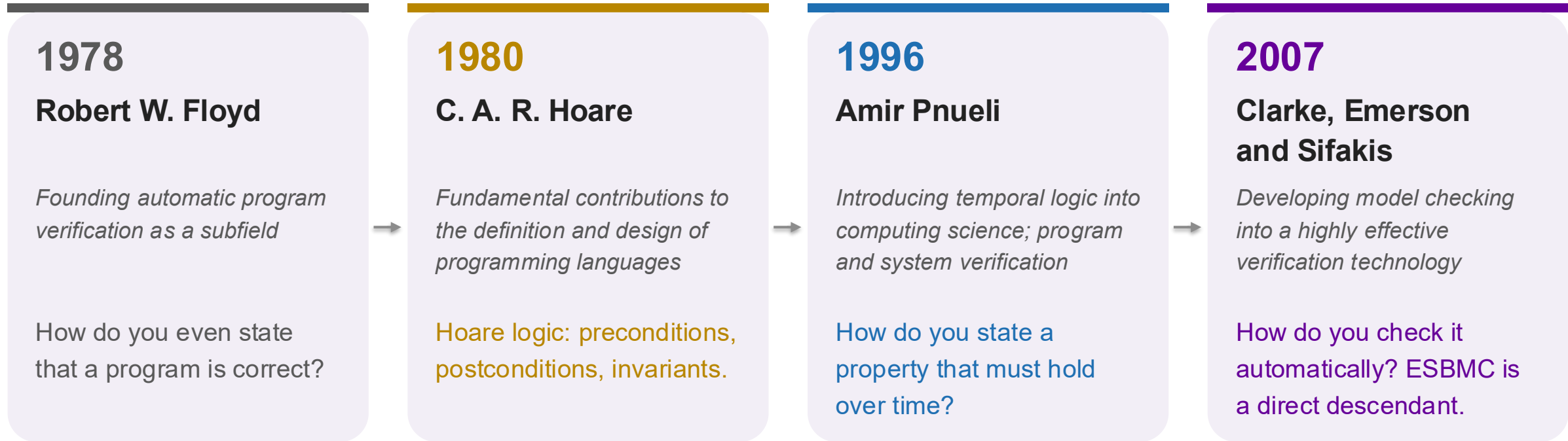
Now · generative AI

Code is written faster than anyone can read it, so the question returns with no human in the loop.

Almost eighty years on, Manchester is still working on the question Turing asked.

Four Turing Awards built the ground this talk stands on

Automated program reasoning: a sixty-year research programme.



Every award was for making correctness machine-checkable; that is still open, and generative AI makes it urgent.

A good field to join: the foundations are settled, the automation is not.

Guess the number

310,531 C programs · 9 LLMs · every one model-checked

What fraction of the best model's code passes formal verification?

Model	Programs	VS %	VF %	VU %
CodeLlama-13B	11,838	15.48	52.71	29.52
Llama2-13B	17,779	12.36	51.30	31.78
Gemma-7B	36,787	11.62	67.01	16.30
Falcon2-11B	11,946	10.28	63.16	24.56
Gemini Pro 1.0	38,695	9.49	63.91	24.13
Mistral-7B	9,934	8.36	62.08	25.88
GPT-3.5-turbo	76,168	7.29	65.07	26.09
Falcon-180B	68,463	6.48	62.07	28.67
GPT-4o-mini	38,921	4.23	57.14	36.77
All nine	310,531	8.27	62.07	26.99

$\approx$ 1 in 6

for the best model (15.48%); all nine: 1 in 12 (8.27%)

A failed check $\neq$ a broken program

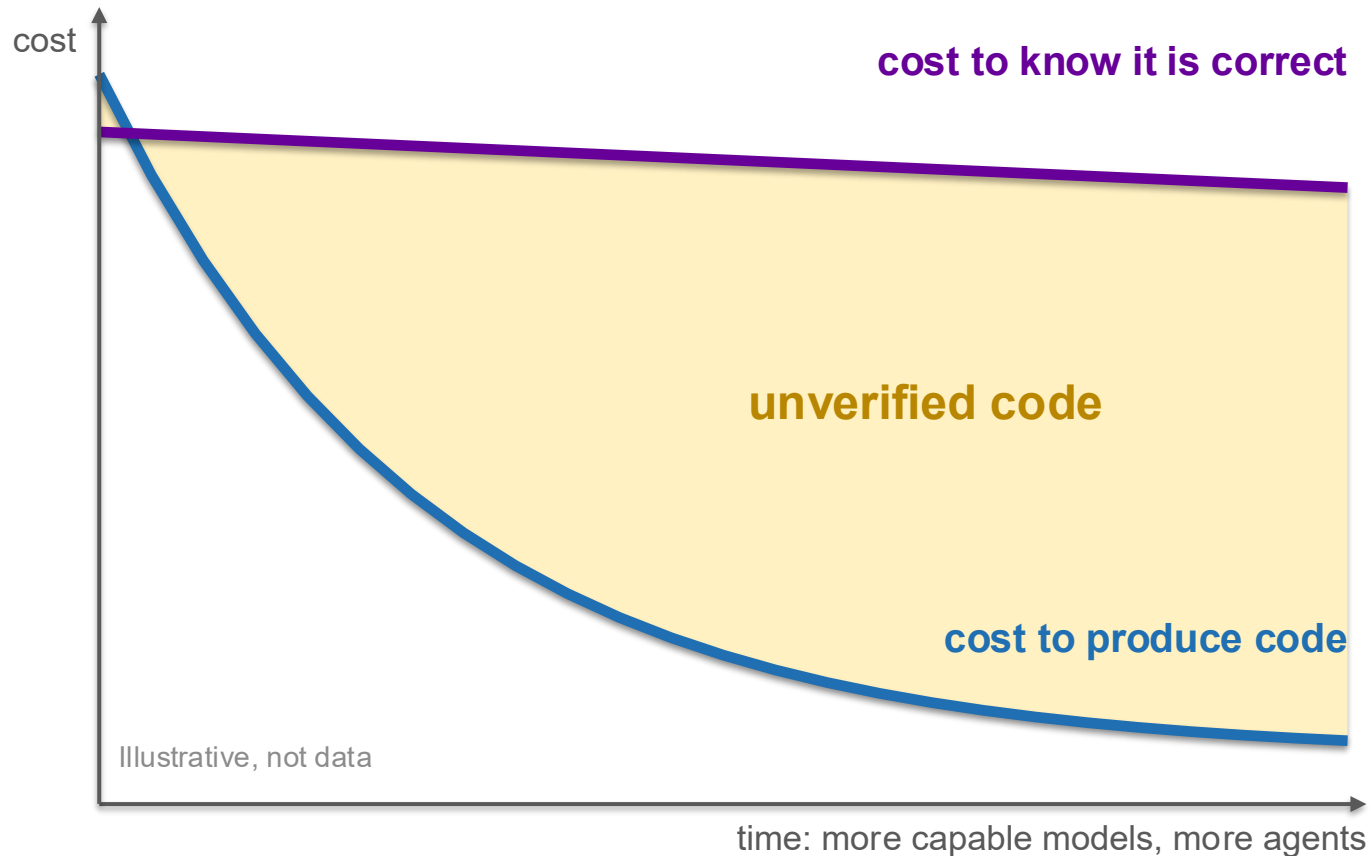
It means we cannot certify it, and that is already the problem.

Not a model-release problem

No model passes even 1 in 6 times; the newest in the study (GPT-4o-mini) scored lowest.

Most AI-generated code cannot be certified today.

Code is getting cheap, and the reviewer is leaving



AI already writes most new code

Google: 75% of new code. Anthropic: over 80% of merged code.

Checking does not, on its own

Review, tests and proofs scale with the code, not with the model.

Agents remove the reviewer

An agent delivers a result, not a step: no human reviews the steps.

The question this talk exists to ask: who verifies that?

Sources: Pichai, Google Cloud Next, Apr 2026 · Anthropic, “When AI builds itself”, Jun 2026

Why not more tests, or a second model?

More tests

Tests sample inputs: in one study, 598 of 661 ESBMC-proven violations escaped every sanitizer run.*

```
int x = read_int(); // user
if (x == 0x13371337) // 1/232
    buf[x] = 0;      // OOB
// no test proves there is no other
```

An LLM reviewer

Swaps one unsound judge for another, often trained on the same data as the generator.

"Looks correct to me."

Formal verification

A proof for every input, or a concrete counterexample. But we used to need specialists to write properties and invariants.

base $\text{Init}(s_0) \wedge T(s_0 \dots s_k) \Rightarrow P(s_0 \dots s_k)$
step $P(s_0 \dots s_k) \wedge T(s_0 \dots s_{k+1}) \Rightarrow P(s_{k+1})$
 k-induction: bounded checks, unbounded proof

And Rust? Safe Rust rules out memory corruption, not logic errors!

That specialist effort is the bottleneck, and it is exactly what a model is good at guessing.

Use the model to guess what a proof needs, never to judge the result.

* Mahmud, Sikder, Ji, Zhang & Lei, "The Illusion of Safety: Multi-Tier Verification of AI vs. Human C++ Code", arXiv:2607.00107, 2026 (preprint).

Cooperation, not replacement: the fast and the sound

Language models

- Fast, fluent, general
- Guess what humans find hard: loop invariants, contracts, specifications, test inputs
- **No guarantee whatsoever**

The interface

is the research problem

What is passed, in what form, and what happens when it is wrong

Proof engines

- Exhaustive
- Verdicts are machine-checkable; a counterexample is executable
- **Expensive; need a spec**

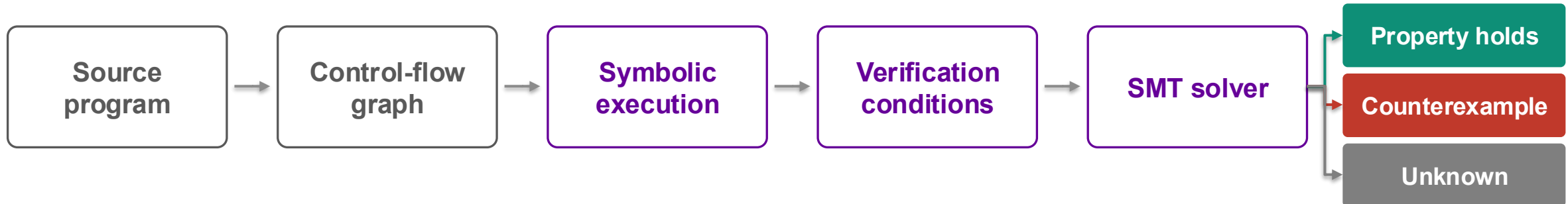
The design rule: Let the model suggest, but let the proof engine confirm. If an incorrect invariant or test causes a delay, it delays execution, not correctness; a human detects a wrong specification

LLMs are fast but unsound. Verifiers are sound but slow. A careful interface gives you both.

ESBMC: one intermediate representation language

★ ASE 2023 Most Influential Paper Award (ASE 2009 paper)

ESBMC is an open-source, context-bounded model checker that **detects or proves the absence of runtime errors** in single- and multi-threaded C, C++, CUDA, CHERI, Java/Kotlin, Python, Rust and Solidity



Open source

Apache 2.0 (mainly)

Two useful answers

A proof, or an input that makes the program fail; otherwise, it says *unknown*

Funded by

Arm, Intel, NVIDIA, EPSRC, Ethereum F., Rust F., EU H2020 ELEGANT, UKRI

The output is evidence a machine can re-check, not a confidence score.

k -induction: from bounded search to unbounded proof

Base case: $B(k)$

BMC to depth k : finds real bugs

Forward condition: $F(k)$

All loops unrolled within k : a proof

Inductive step: $I(k)$

Holds for k steps, so for the next: a proof

The algorithm ($--k$ -induction)

```
for k = 1, 2, ... up to max-k:  
    if B(k) finds a bug: return FAILED  
    if F(k) holds:      return SUCCESSFUL  
    if I(k) holds:      return SUCCESSFUL  
return UNKNOWN
```

Unbounded loop

UNKNOWN

+ one invariant

$I(k)$ proves, $k = 2$

Plain k -induction stalls on the unbounded loop; one invariant closes the proof.

Contracts: prove each function once, reuse it everywhere

```
int my_abs(int x) {
  __ESBMC_requires(x > INT_MIN);
  __ESBMC_ensures(__ESBMC_return_value >= 0);
  __ESBMC_ensures(__ESBMC_return_value == x
                  || __ESBMC_return_value == -x);
  return x < 0 ? -x : x;
}
```

Why contracts

Guard against change → an edit that breaks it fails, in the body or at a call.

Scale → callers use the contract: 50 calls in 1.8 s, not 9.4 s inlined.

1 · Enforce: the body meets it

```
$ esbmc abs.c \
  --enforce-contract my_abs \
  --function my_abs
PASSED contract ensures
VERIFICATION SUCCESSFUL
```

2 · An edit that breaks it fails

```
// abs_bad.c: returns x
$ esbmc abs_bad.c \
  --enforce-contract my_abs \
  --function my_abs
FAILED contract ensures
VERIFICATION FAILED
```

3 · Replace: callers use the contract

```
$ esbmc abs.c \
  --replace-call-with-contract \
  my_abs
PASSED contract requires
PASSED r >= 0 && r < 100
VERIFICATION SUCCESSFUL
```

Prove the function once against its contract; every caller then uses the contract, not the body.

A counterexample is a test; a proof is a certificate

```
// illustrative example
int copy_bytes(const unsigned char *in, size_t n,
               unsigned char out[4]) {
    if (n > 4) return -1;
    for (size_t i = 0; i <= n; i++)    /* off-by-one */
        out[i] = in[i];
}
```

Two kinds of witness

Violation → a concrete input, replayed as a regression test.

Correctness → a certificate an independent validator re-checks.

1 · ESBMC finds the bug

```
$ esbmc copy_bytes.c \
  --unwind 6 \
  --generate-ctest-testcase
n = 4
array bounds violated
VERIFICATION FAILED
```

2 · It becomes a C program

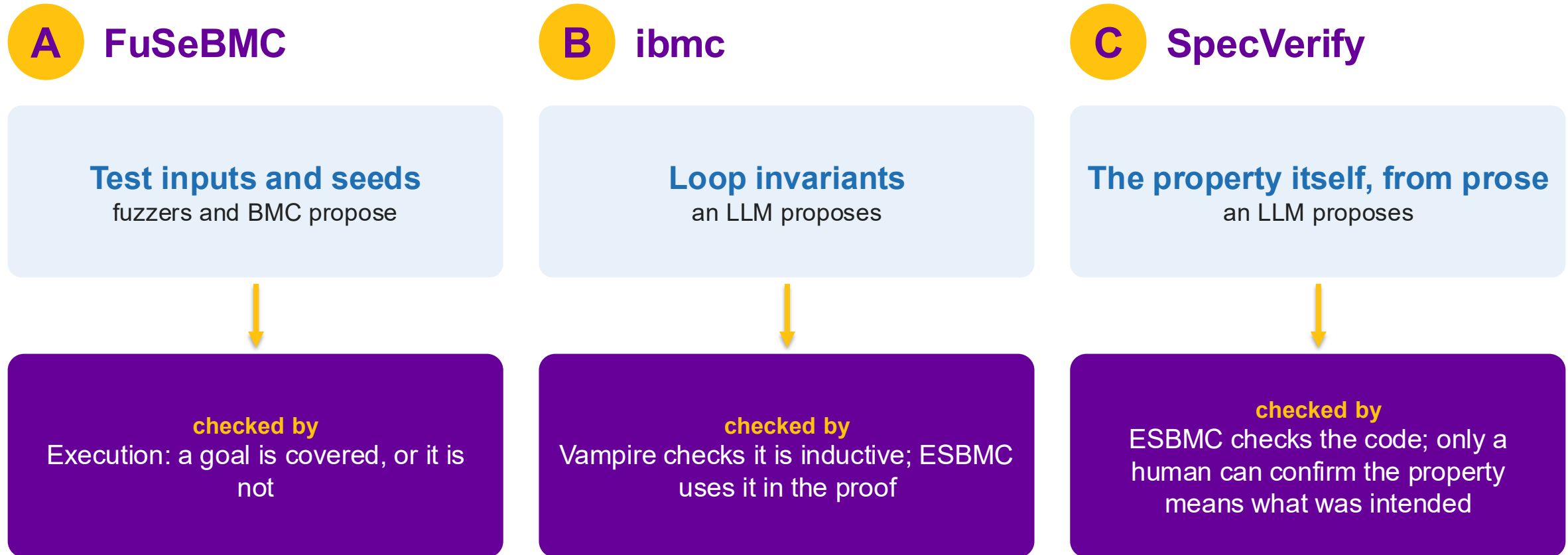
```
// test_case.c (generated)
...nondet_uint(void) {
    static unsigned int const
        v[] = { 4 }; ...
$ ./test_case
ASan: stack-buffer-overflow
```

3 · Fix $i < n$: test passes, proof holds

```
$ ./test_case → exit 0

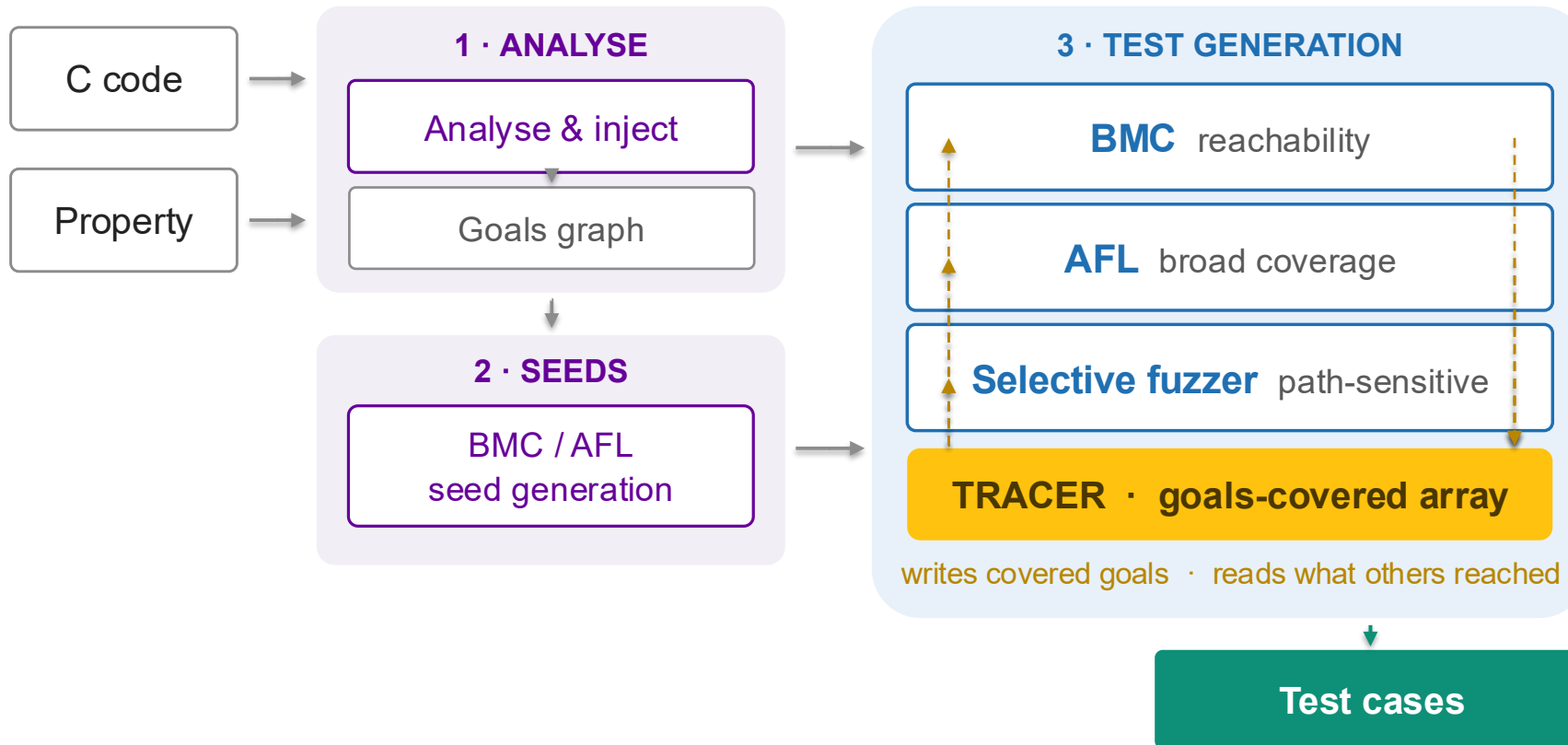
$ esbmc copy_bytes-fixed.c \
  --k-induction
VERIFICATION SUCCESSFUL
all states reachable, k = 6
```

Three places a guess can enter



The proposal is cheap and the check is never optional; in C, the check also needs a human.

FuSeBMC: a portfolio with a shared coordinator



Three engines

Each strong where the others are weak

One coordinator

The Tracer records covered goals in shared memory

It feeds back

New coverage becomes a seed for the next round

Result: 1st in C.Overall, C.Cover-Error and C.Cover-Branches at Test-Comp 2026.

Judged by blind competition, not by press release

Test-Comp 2026

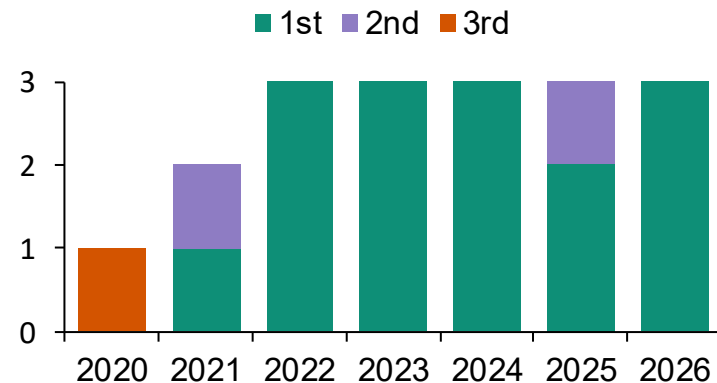
FuSeBMC: 1st in C.Overall, C.Cover-Error and C.Cover-Branched.

SV-COMP 2026

ESBMC-kind: 2nd in C.ReachSafety (gold: CPAchecker).

Rust Foundation, Aug 2025

ESBMC approved for the Rust standard-library verification effort, with Flux, Kani and VeriFast.



Medals per year: ESBMC 2020, FuSeBMC 2021–26

The same discipline, every year

- Benchmarks change yearly, so scores aren't comparable year to year.
- The method is constant: public tasks, a blind run, organisers who are not us.

Annual, blind, third-party benchmarking, the same evidence any competitor is judged on.

The bound problem: 100 unrollings for one assertion

```
x = 0; y = 50
# copy 1
x = x + 1
if x > 50: y = y + 1
# copy 2
x = x + 1
if x > 50: y = y + 1
# ... 98 more copies
assert y == 100
```

An inductive invariant replaces the loop

Holds on entry, preserved by every iteration:

```
__ESBMC_loop_invariant(0 <= x and x <= 100)
__ESBMC_loop_invariant(not (x <= 50) or y == 50)
__ESBMC_loop_invariant(not (x > 50) or y == x)
```

The bound disappears.

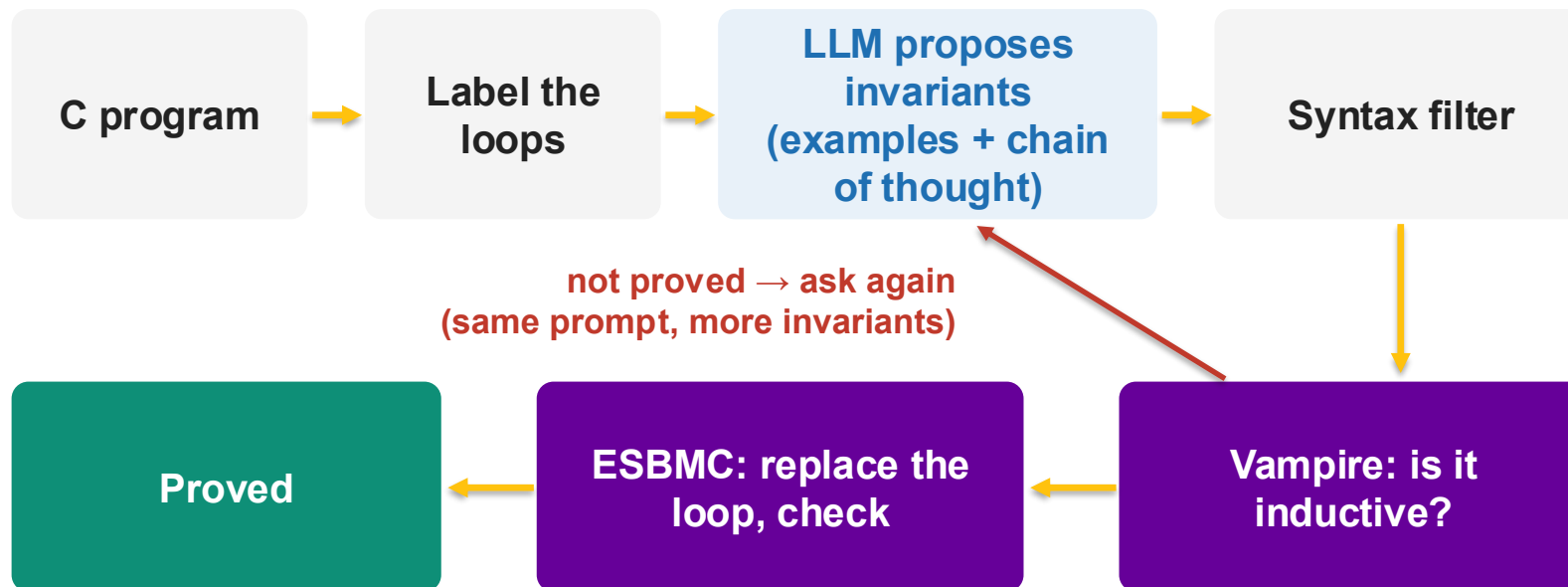
Correct, but only after 100 unrollings.
(Illustrative; ibmc's benchmarks are C.)

A wrong invariant costs time, never correctness.

Invariant synthesis is the classic bottleneck in automated verification, and a good fit for LLMs.

ibmc: the LLM proposes, two proof engines dispose

★ ACM SIGSOFT Distinguished Paper Award, ASE 2024



Sound, not complete

The LLM's output is never trusted, only checked: it can cost time, never add a wrong proof.

Constrain the output

67–75% of tries don't parse.

Guide the format, then filter

Filtering alone limits flexibility.

Balance the prompt

Too much explanation distracts.

Nothing the LLM proposes reaches the proof until Vampire has shown it is inductive.

Soundness is preserved by construction: nothing the model says is trusted, only checked.

Up to 101 of 133 programs proved, none of them wrongly

Benchmark programs solved

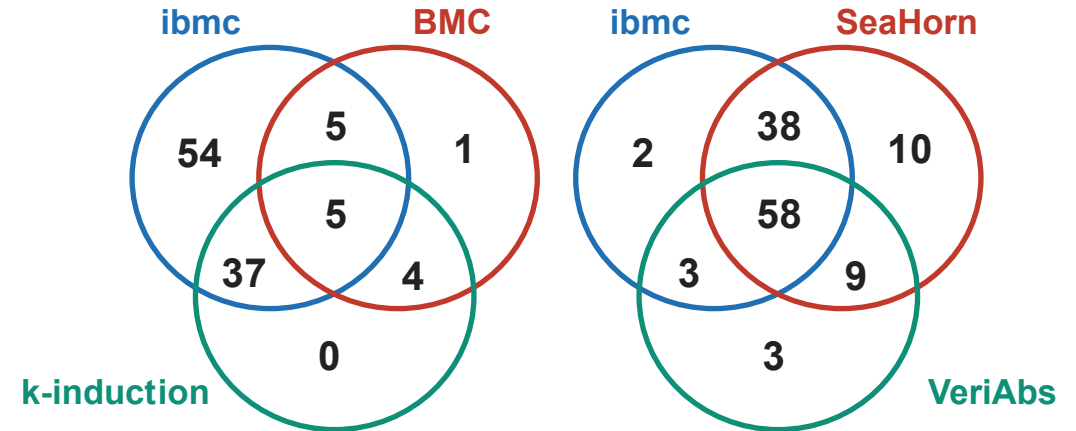


*Union of six prompt configurations; any single one solves ~58% (≈ 77 of 133), still well above k-induction (46).

Sound by construction

None of the 101 contradicts the benchmark's ground truth. The LLM cannot make a verdict wrong, only slower.

Which programs each engine reaches

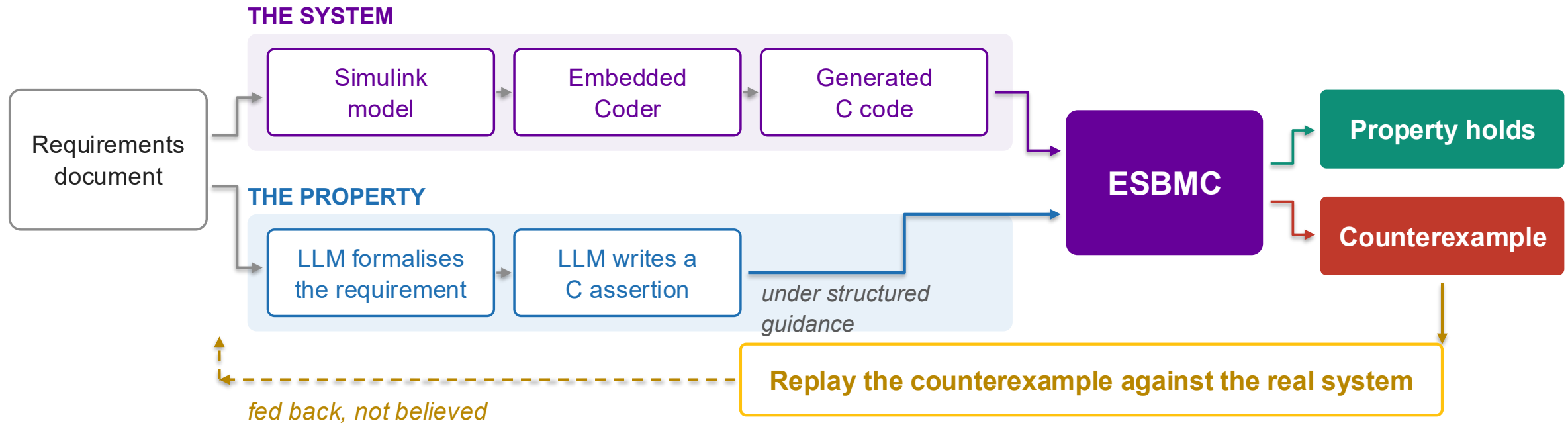


Complementary, not redundant

Different engines miss different programs: the argument for a portfolio.

Each set of invariants proves programs the other misses: LEMUR's 6, ours 10.

SpecVerify: from NL requirements to a checked property



Industrial benchmarks

Lockheed Martin CPS challenge problems, Simulink to C

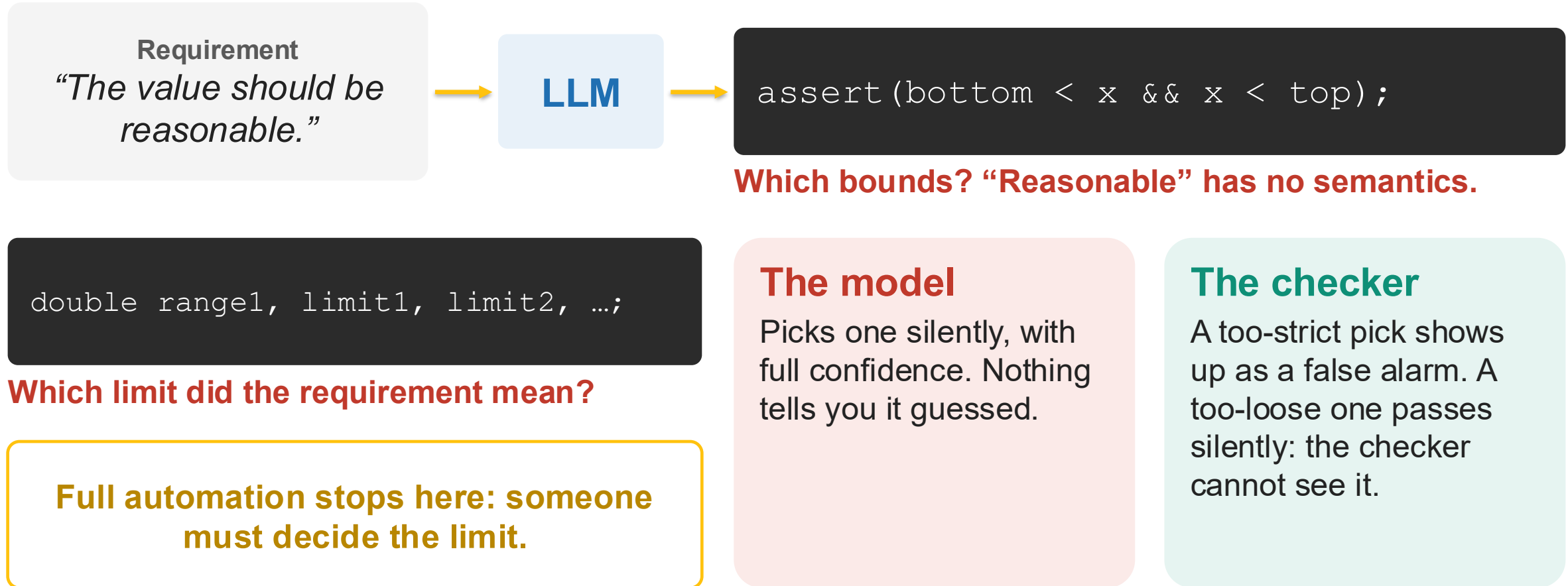
Two phases

Formalise the requirement, then turn it into a C assertion

The loop that closes it

The LLM is never the final check; any failing property is fed back

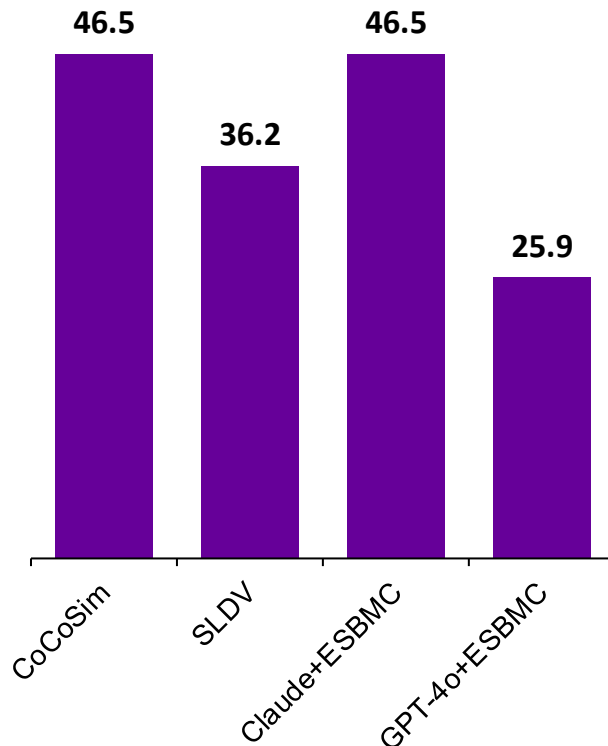
“The value should be reasonable”



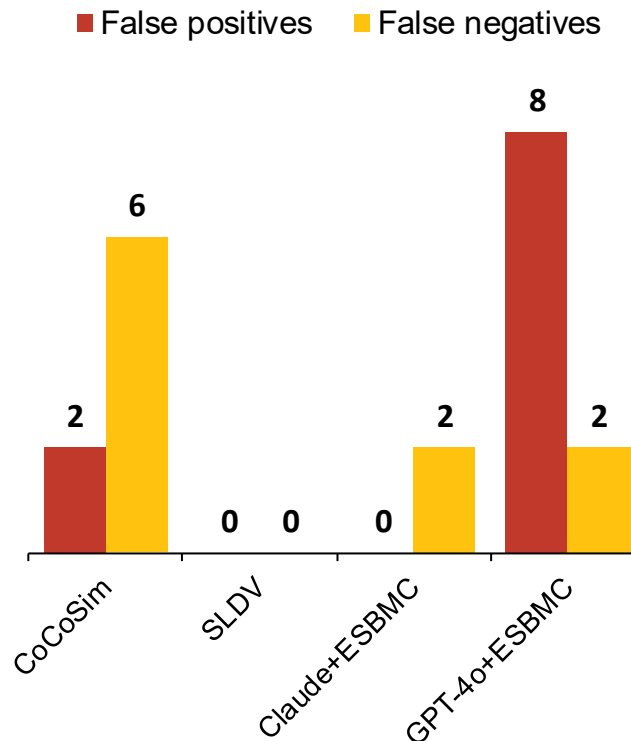
Here a wrong guess can cost correctness, not just time: the specification needs a human.

Expert-level agreement, with no false positives

Verification rate (%)



False positives / false negatives



79.31%

match expert-written properties (46 of 58)

0

false positives: the verifier never cried wolf

2

previously unreported floating-point errors found

Read the chart carefully

The verification rate, 46.5% (27 of 58), equals CoCoSim's. SLDV also has no false alarms but verifies fewer (36.2%).

Claude 3.5 Sonnet + ESBMC still missed 2 violations: a wrong specification costs correctness.

The model choice matters: with GPT-4o, the same pipeline produced 8 false positives.

Benchmarks have an answer key; real code does not

SO FAR · BENCHMARKS WITH KNOWN ANSWERS

A · FuSeBMC

Test-Comp tasks, scored blind

B · ibmc

133 code2inv programs, verdicts known

C · SpecVerify

58 Lockheed Martin requirements



NEXT · REAL CODE, NO ANSWER KEY

What goes missing

No ground truth to score against

No written specification

No model of the environment the code runs in



the new guess: the harness

An LLM agent writes the harness and its assumptions

checked by

ESBMC, a seeded-bug twin that must fail, and the maintainer who merges the fix

Same design rule, new guess: the LLM writes the harness, and the maintainer becomes the answer key

Production code: bugs found, properties proved

8

codebases, every harness public

2

CVEs in the AWS SDK for C++

18

fixes merged upstream

Codebase	Findings	Fixed upstream	Proved
AWS SDK for C++	12	7, incl. 2 CVEs	22 · bounded inputs
NVIDIA OpenSMA firmware	11	1 reported, fixed	22 · unbounded
vLLM	8	5	21 · bounded inputs
Google Chromium Dashboard	10	3	22 · bounded inputs
AWS Neuron NKI samples	2	2	19 · mixed

An LLM agent drafted the harnesses; ESBMC, seeded-bug twins, and maintainers confirmed them

AWS SDK for C++: two CVEs in one Base64 decoder

CVE-2026-19642 · heap write

"AAAA="

The size counts '='; the loop does not. 2-byte buffer, write to buffer[2]

CVE-2026-19643 · read (signed char)

"\xFF\xFF\xFF\xFF"

Signed char 0xFF becomes a huge index into a 256-entry table

1.11.862

fixes both CVEs

+2

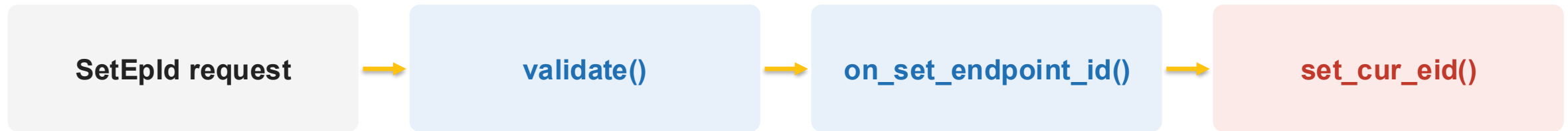
DateTime overflows fixed

+3

cJSON defects fixed

The counterexample was the bug report: a concrete input that can run with the sanitisers

NVIDIA OpenSMA: one bad index aborts the firmware



```
routing_table.ec.cur_eid.at(interface) = eid;    // cur_eid has 2 entries
```

Counterexample: iface = 2

Any 2–17 passes the validator; at() fails its bounds check and, built with -fno-exceptions, aborts: denial of service, not memory corruption.

Confirmed and fixed by NVIDIA

“The bound check is now mirrored into set_cur_eid()” (issue #1, May 2026).

The getter had the bounds check; the setter did not. ESBMC found the asymmetry.

vLLM: verifying the engine that serves the models

A malformed flag passes every validator, then crashes, hangs or misbehaves.

29

verification targets

8

live findings, 6 upstream issues

5 of 6

issues fixed upstream

`--block-size 0`

ZeroDivisionError at engine init

fixed

`--hash-block-size -k`

infinite loop at init

fixed

`--max-logprobs <neg>`

silently accepted

fixed

`--num-gpu-blocks-override 0`

bare AssertionError at engine init

open

The stack that serves the models is ordinary software, and it breaks in ordinary ways.

Five lessons from doing this on real code

1

Zero false positives beats a higher proof rate

Developers stop trusting a tool that raises false alarms, and start ignoring it (Coverity; Google).

2

Maintainers fix what arrives with a concrete input

18 fixes merged upstream across five codebases.

3

The generator's quality still propagates

SpecVerify with GPT-4o instead of Claude 3.5 Sonnet: 8 false positives, not 0.

4

We do not win every benchmark

SeaHorn proves more than ibmc; CoCoSim matches SpecVerify. Portfolios beat champions.

5

Verifiers have traps too

Zero VCCs, disabled unwinding assertions, contracts that silently check nothing: the evidence needs checking.

These are the results a paper talk leaves out.

Three challenges we must solve together

1 Verification at generation speed

Agents generate code in seconds; industrial proofs take minutes to hours. Checking must reuse earlier proofs

Industry brings
agent traces

Academia brings
incremental proof reuse

2 Guarantees for agent workflows

Verifying each agent is not enough to verify the workflow they form

Industry brings
real workflows

Academia brings
compositional contracts

3 Evidence we can trust

DO-178C already accepts formal evidence (DO-333). Who checks the checker?

Industry brings
certification needs

Academia brings
witnesses, validators

Every one of these is a joint problem: neither community solves it alone.

The people and funders behind this work

Whose work you saw

FuSeBMC

Kaled Alshmrany · Mohannad Aldughaim ·
Ahmed Bhayat

ibmc

Muhammad A. A. Pirzada · Giles Reger ·
Ahmed Bhayat

SpecVerify

Weiqi Wang · Marie Farrell · Liping Zhao

FormAI

Norbert Tihanyi · Tamas Bisztray · Mohamed
Amine Ferrag · Ridhi Jain

ESBMC

Tong Wu · Rafael Sá Menezes, and all ESBMC
contributors



IN CLOSING

**LLMs are fast but unsound.
Proof engines are sound but slow.
Cooperation gives you both,
if you design the interface on purpose.**

Prof. Lucas C. Cordeiro

lucas.cordeiro@manchester.ac.uk

cs.manchester.ac.uk/research/expertise/systems-and-software-security/

github.com/esbmc/esbmc

Thank you. Questions?

MANCHESTER
1824

The University of Manchester



UFAM