

Software Testing and Static Analysis inside the same LLM



Professor Karl Meinke
joint work with Carl Wang MSc.
Dept. of Computer Science
KTH Royal Institute of Technology

Cyber Security

[+ Add to myFT](#)

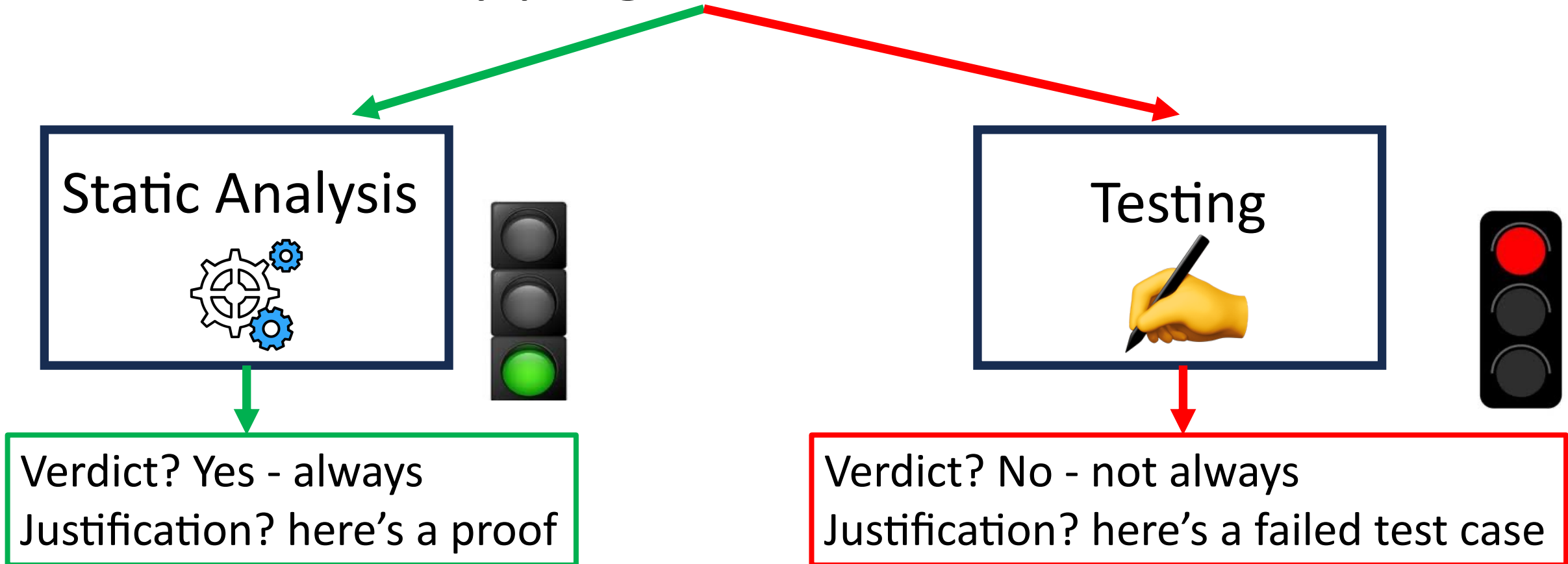
Amazon warns new cyber flaws detected using AI are being ‘broadly exploited’

Researchers used OpenAI and Google models to find severe faults in software used on the majority of websites around the world

“Given the severity of these flaws and PHP’s widespread use, we consider this one of the most impactful security issues in recent memory,” OpenAI said on

Two Routes to SQA

Does my program “work”? 🤔



A Short History of the Algorithms

Static Analysis

Resolution (Prolog) 1960s

Backward Chaining 1980s

Model Checking (SAT) 1990s

LLMs 2018

Testing

Solvers e.g. CAD algorithm 1975

Model Checking e.g. SAT solving 1990s

LLMs 2018

AI4Test before LLMs

Reverse Engineering Models with ML
then plug into model-based testing

Graph Neural Networks
on Control Flow Graphs

Typically lacking sufficient ground-truth data ...

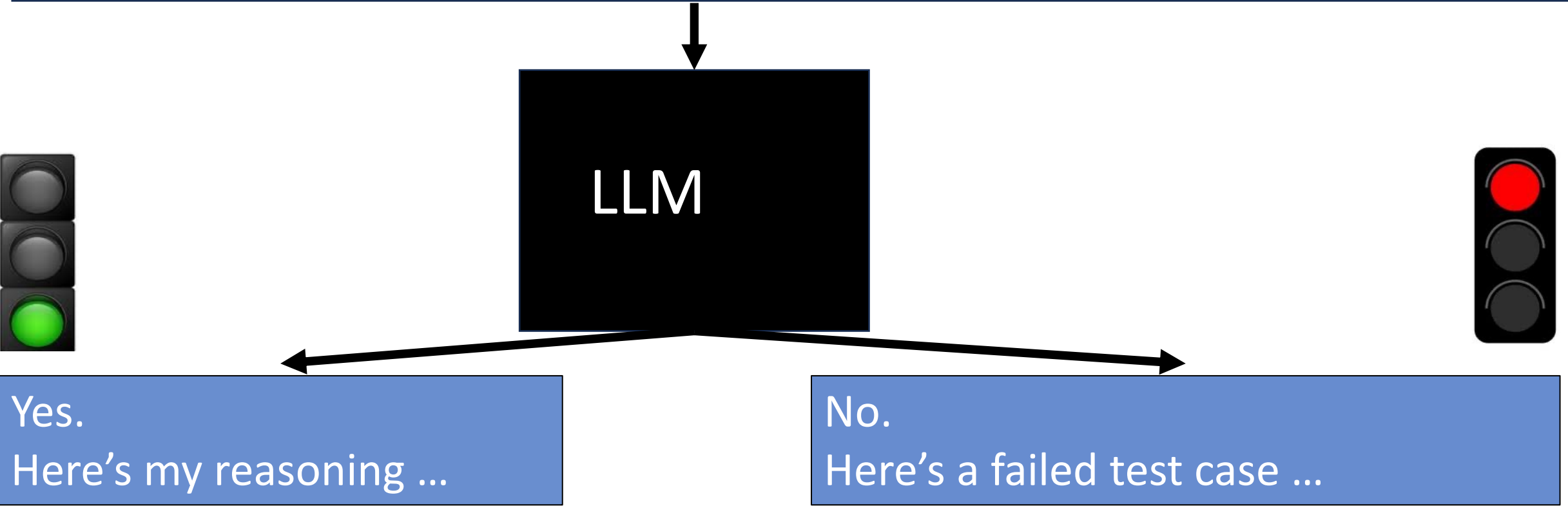
LLM as a Competent Mathematician

“LLMs have the mathematical ability of a competent math grad student”
Terence Yao

Navier-Stokes Problem - September 8, 2026, OpenAI announced a solution using a multi-agent system of roughly 10,000 AI agents working for 88 hours.

A Simple LLM Workflow (Chain-of-Thought)

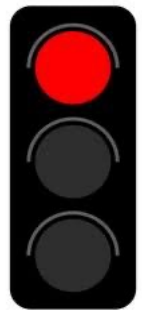
Prompt: Study this code <myCode> and tell me if it satisfies this user requirement <myReq> . If it does, justify your reasoning. If it doesn't, give me a test case which fails the requirement. Take your time to think, there's no hurry.



LLM



Yes.
Here's my reasoning ...



No.
Here's a failed test case ...

Requirements Modeling and Metamorphic testing

What do we mean by:

“the code works” 🤔

A **test oracle** decides whether a test **passes** or **fails**.

But the exact expected output is often unknown.

What if...

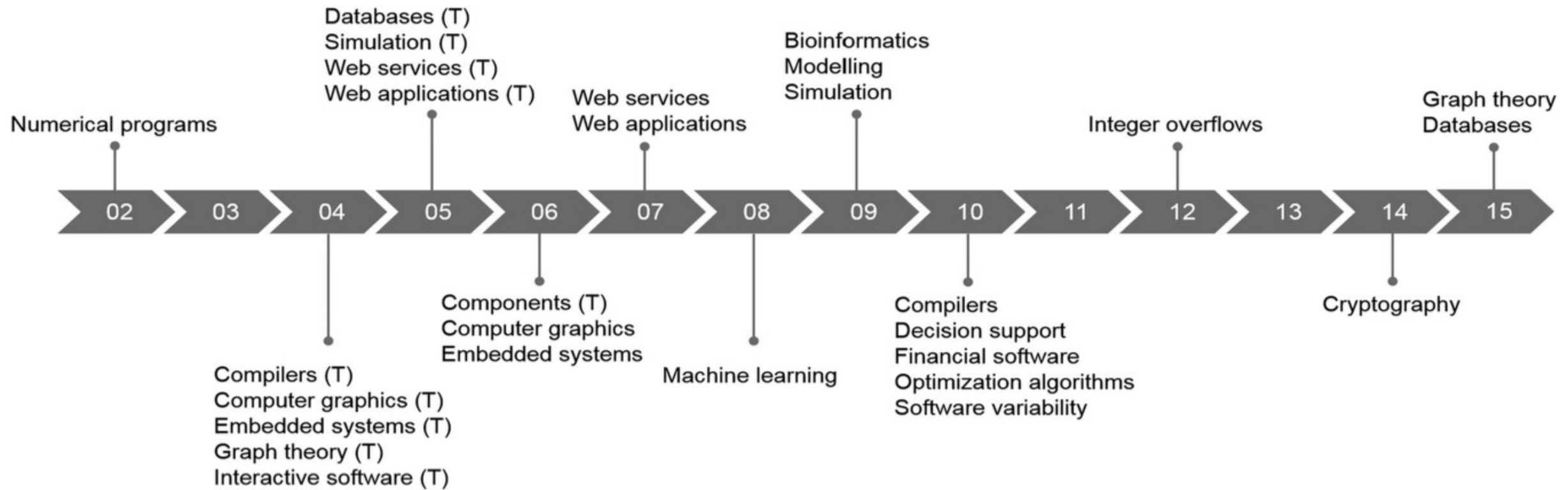
Metamorphic testing (MT)

... instead of trying to know the exact expected outputs, determine how the outputs vary based on input transformation.

$$\text{Sum}(\text{perm}(\text{ListX})) = \text{Sum}(\text{ListX})$$

This is so called a **metamorphic relation (MR)**. However, this requires manual efforts and strong domain knowledge.

Metamorphic Application Areas: a Timeline



Timeline of Metamorphic Applications since 2002

Limitations of LLMs

Reasoning

Multi-step inference — architectures predict the next token, so some researchers read it as statistical recombination rather than grounded logical inference

Prompting helps — Chain-of-Thought (Wei et al.); "let's think step by step" lifts zero-shot reasoning (Kojima et al.)

Limits persist — Rueda et al.: still pattern recognition; Yuan et al. and Strobl et al.: architectural limits on discrete reasoning

Code analysis

Ma et al. — syntax handled well; static and especially dynamic semantics are not, with hallucination on code semantics

Haroon et al. — 575k semantics-preserving mutation tasks: understanding is shallow, leaning on semantically irrelevant features

Li & Shin — mutation-based consistency testing; both GPT-3.5 and GPT-4 weakest on relational changes

RESEARCH QUESTION

To what extent can LLMs correctly judge, prove, or refute the validity of metamorphic requirements on multi-fault-injected Python unit programs?*

Judge

Does the MR hold for this program? TRUE or FALSE

Prove

An informal proof followed by a machine-verifiable Lean 4 proof.

Refute

A test case and an executable Python script that demonstrates the violation.

METHOD

Three experiments of increasing difficulty ...

Experiment 1 Correct Code

Complete, behaviourally correct programs. Informal proof → **Lean 4** proof, with verifier errors fed back for revision (max 10 attempts).

GPT-5.4 · thinking mode

Experiment 2 Single-fault Code

One fault injected with **Mutmut**. Two models × two opposing roles: proof-generation vs. counterexample-generation.

GPT-5.4 · Gemini 3.1 Pro

Experiment 3 Multiple-fault Code

Higher-order mutants, $n = 1 \dots 10$, on five sorting programs.

Can LLM identify all behaviourally relevant faults?

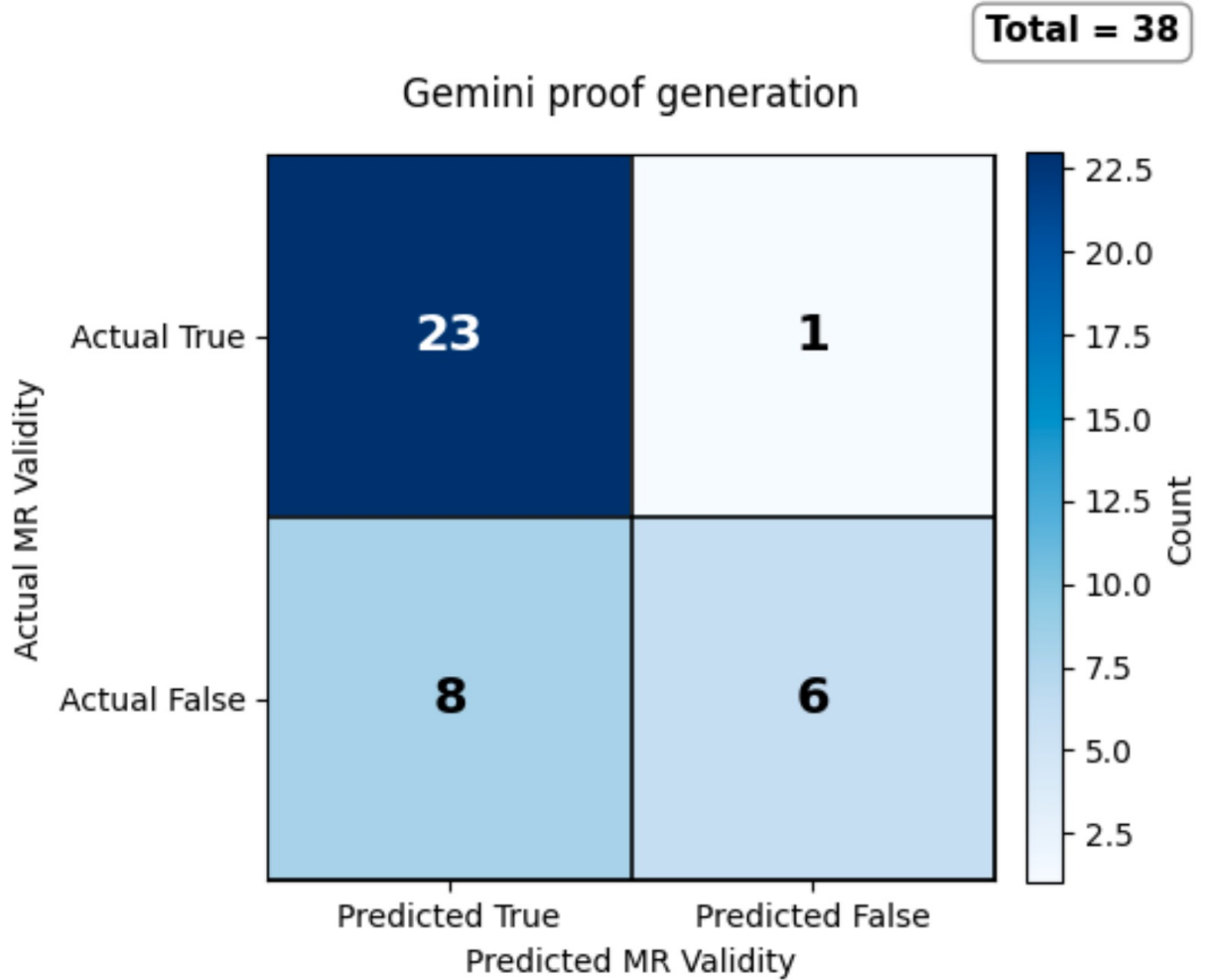
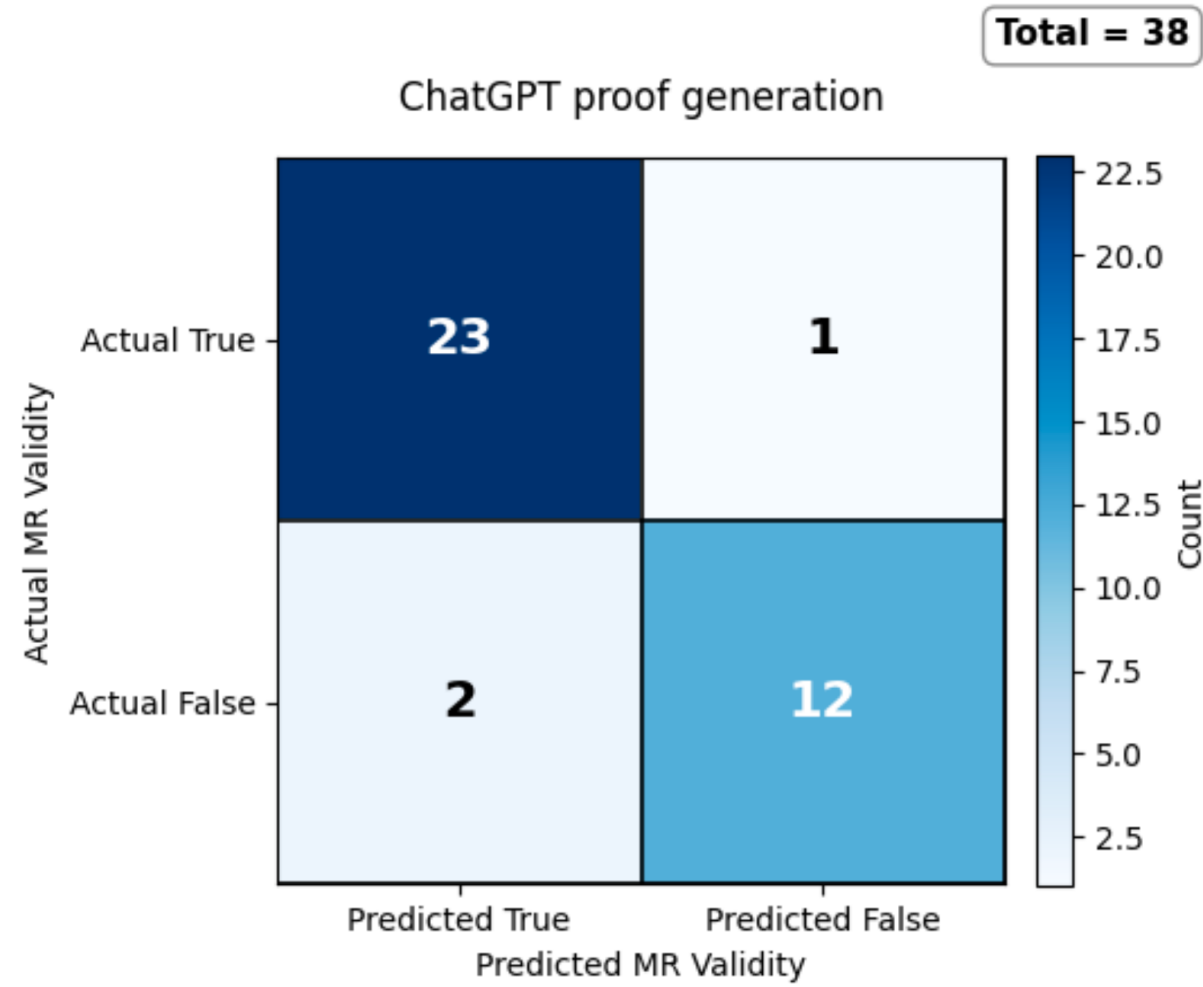
GPT-5.4 · thinking mode

Common to all three: zero-shot Chain-of-Thought prompting · one isolated conversation per test instance to prevent cross-task contamination · manually verified MRs as ground truth · fixed random seed for mutant selection.



Experiment 2 · Single-Fault Code

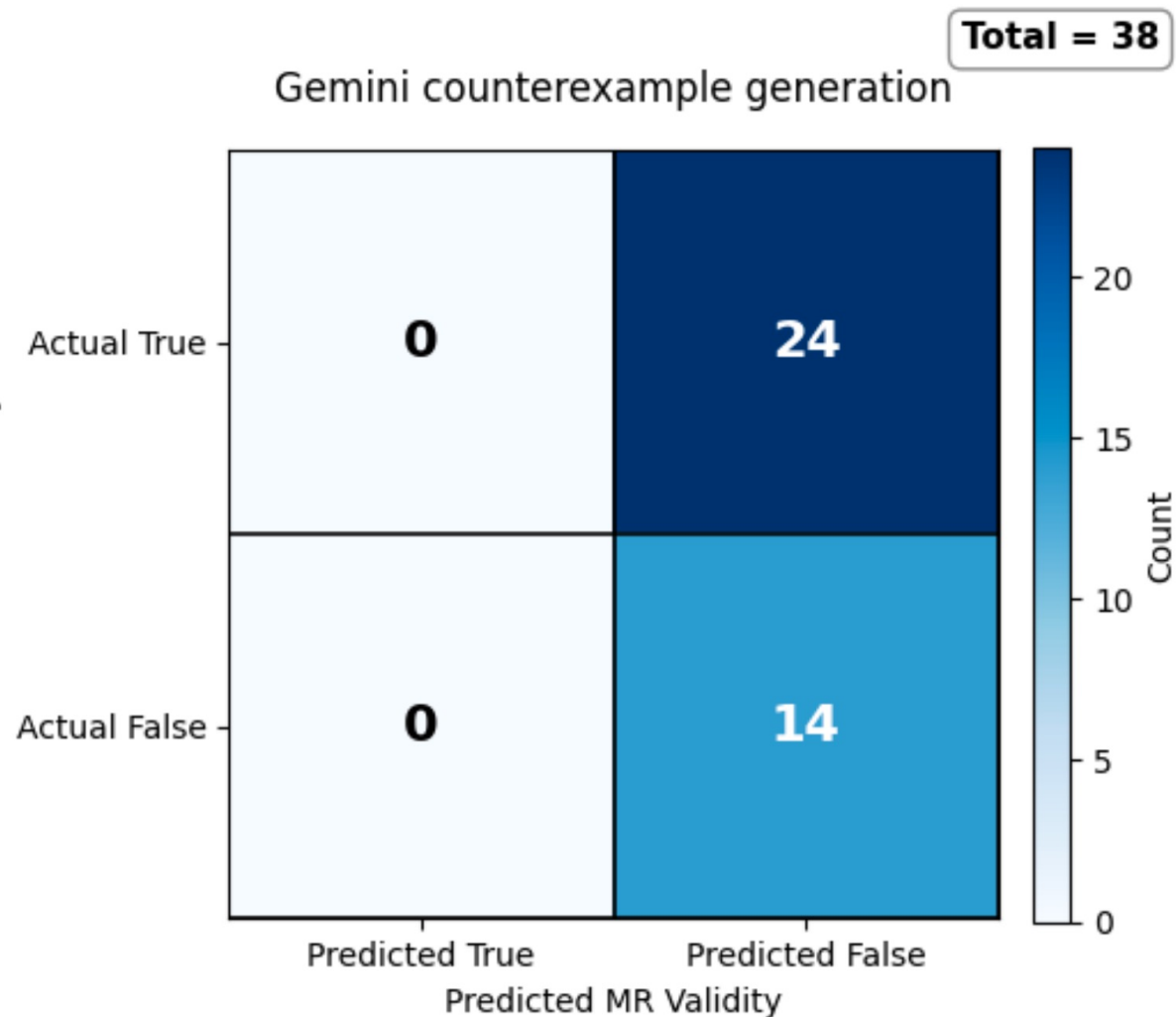
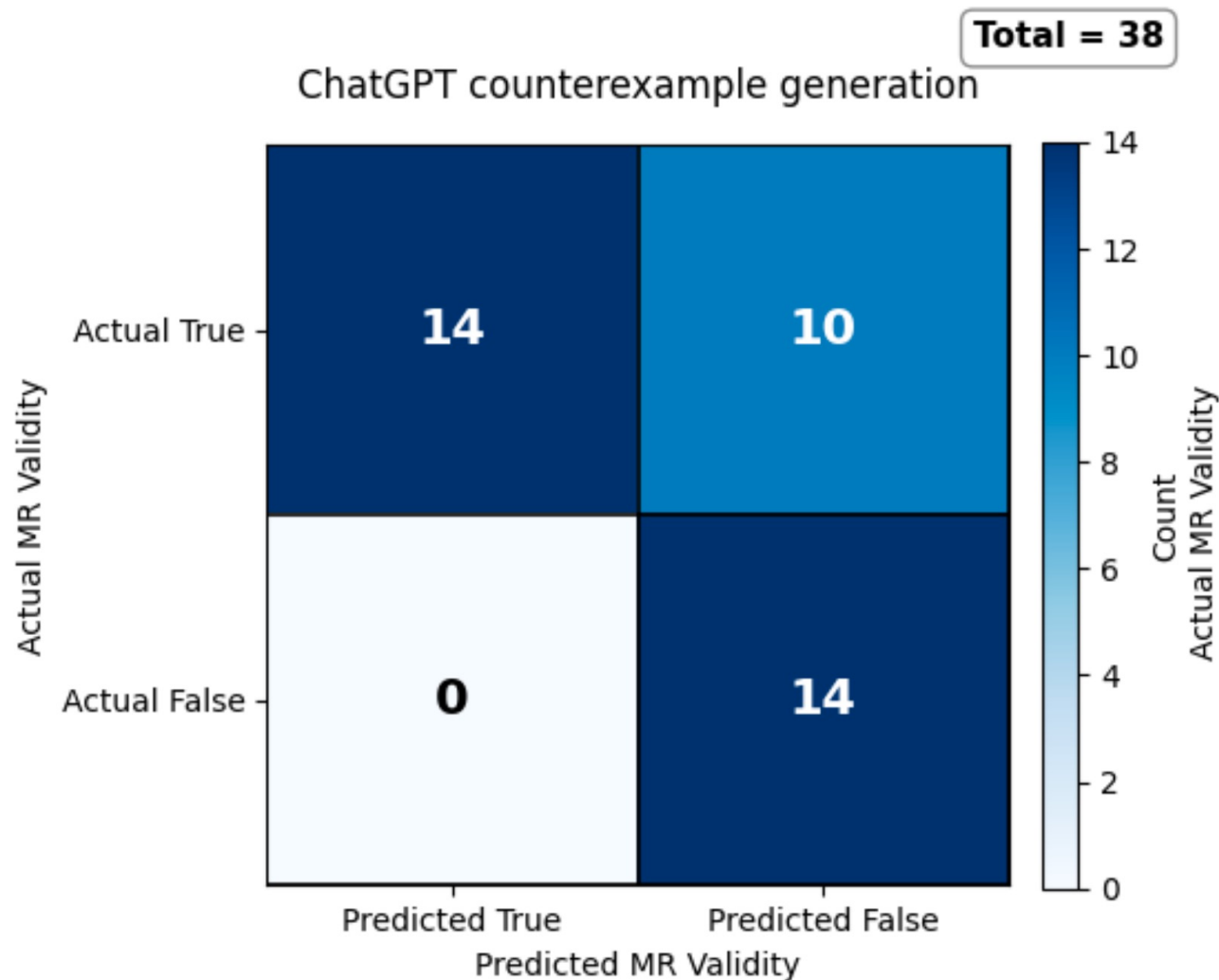
Proof confusion – slight shared bias





Experiment 2 · Single-Fault Code

Counterexample confusion – shared complementary bias!



Discussion 1

Two findings cut across all three experiments

1 · Pattern recognition is still the main issue

The models were most reliable when the code matched a familiar, behaviorally correct algorithm. Once mutation pushed the implementation away from that pattern, they drifted between reasoning about the original algorithm and the actual code.

2 · External verification was necessary

Lean 4 and executable Python scripts were necessary for establishing ground truth and for exposing the causes of failure. The ability of producing plausible informal proofs is not equal to understanding.

LLMs are best used as generators of candidate evidence inside a human-assisted, ground truth supported workflow, not as reliable autonomous validators of metamorphic relations.

Discussion 2

Threats to validity

News

Artificial intelligence

The battle for mathematical truth

A key step for showing that claimed solutions for mathematical problems are correct could be at risk from hacking by artificial intelligence, reports **Matthew Sparkes**

Ramana Kumar had used AI to discover and exploit two separate bugs in two separate kernels within the Lean code.

Lean now ships with **4 kernels** but still prone to **compiler errors**!

Discussion 2

Threats to validity

- **Closed-source, moving targets.** Publicly reachable models via web interfaces, with no transparency.
- **Small, simple dataset.** 44 short Python unit programs, each tested with one MR.
- **Limited fault diversity.** Mutmut defaults, VR over-represented at 47.7%.
- **No prompt ablation.** Prompts may not represent best-case performance.
- **Single researcher evaluation.** Validity and reasoning assessed by one researcher.
- **Lean.** Failing to produce valid Lean script may not always equal to failing to reason.

Literature Study

What had been studied

Generating candidate MRs using LLMs

Identifying relevant variables using LLMs

Translating MRs into executable form using LLMs

*Hazott et al.; Luu et al.; Shin et al.;
Tsigkanos et al.; Zhang et al.*

What was missing

- **Validating** MRs using LLMs

Experimental Setup

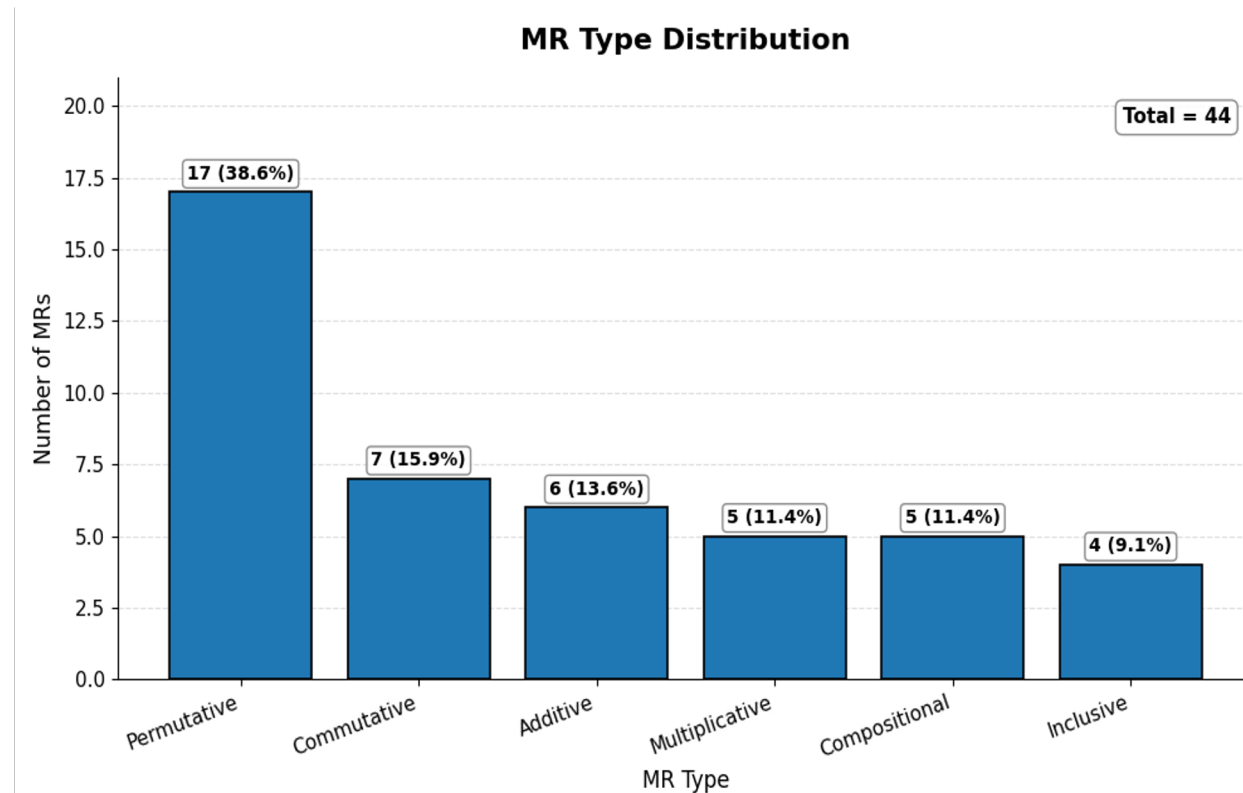
44 unit programs · one MR each

Dataset and metamorphic relations

Re-implemented in Python from the dataset of Nair et al.
Each program is paired with one manually verified MR.

Fault injection with Mutmut

Four fault types — AOR, ROR, LCR, VR. Randomly selected, fixed seed



Experiment 1 · Correct Code

From informal to formal reasoning

44/44

correct informal proofs

118

Lean script failures across 154 attempts

93.2%

valid & complete Lean proofs after iterative correction (41/44)

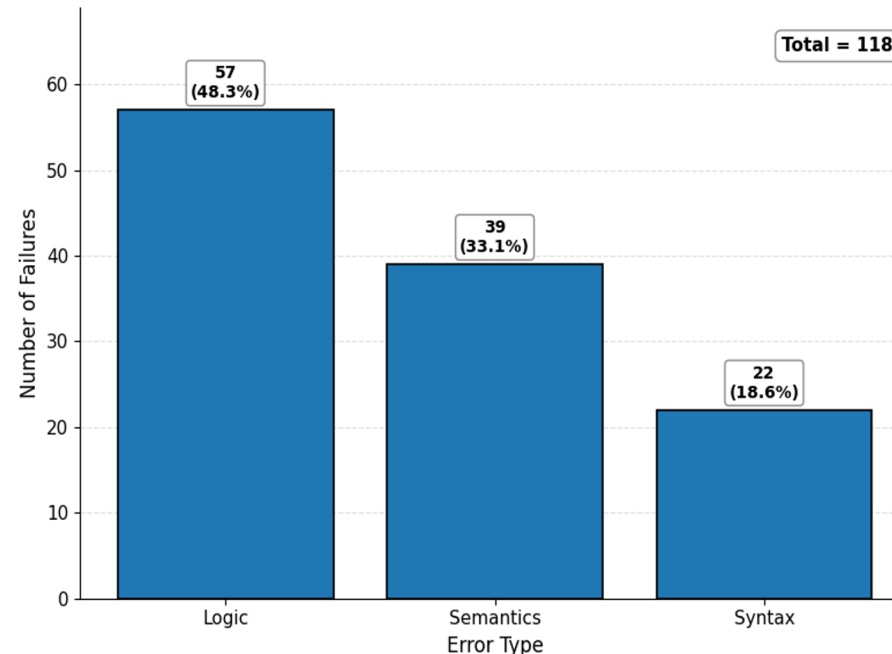
3.80

avg. failures per case for compositional MRs

Script Failures by Error Type

Failure breakdown

- **48.3% logical** — type mismatches, skipped steps, invalid assumptions
- **33.1% semantic** — non-existent or outdated Mathlib functions/symbols
- **18.6% syntactic** — bad imports, stray characters



Experiment 3 · Multiple-Fault Code (n = 2...10 faults)

More faults = less completeness

22

out of 50 cases
Successful

25

out of 50 cases
partially successful

3

out of 50 cases
unsuccessful

1

out of 50 cases
wrong validity judgment

Reasoning completeness decreases as n grows — but the final MR validity verdict stayed almost always right.