Correctness-Oriented Programming with AI

Alperen Keles
University of Maryland, College Park
akeles@umd.edu

What’s Correctness, anyway?

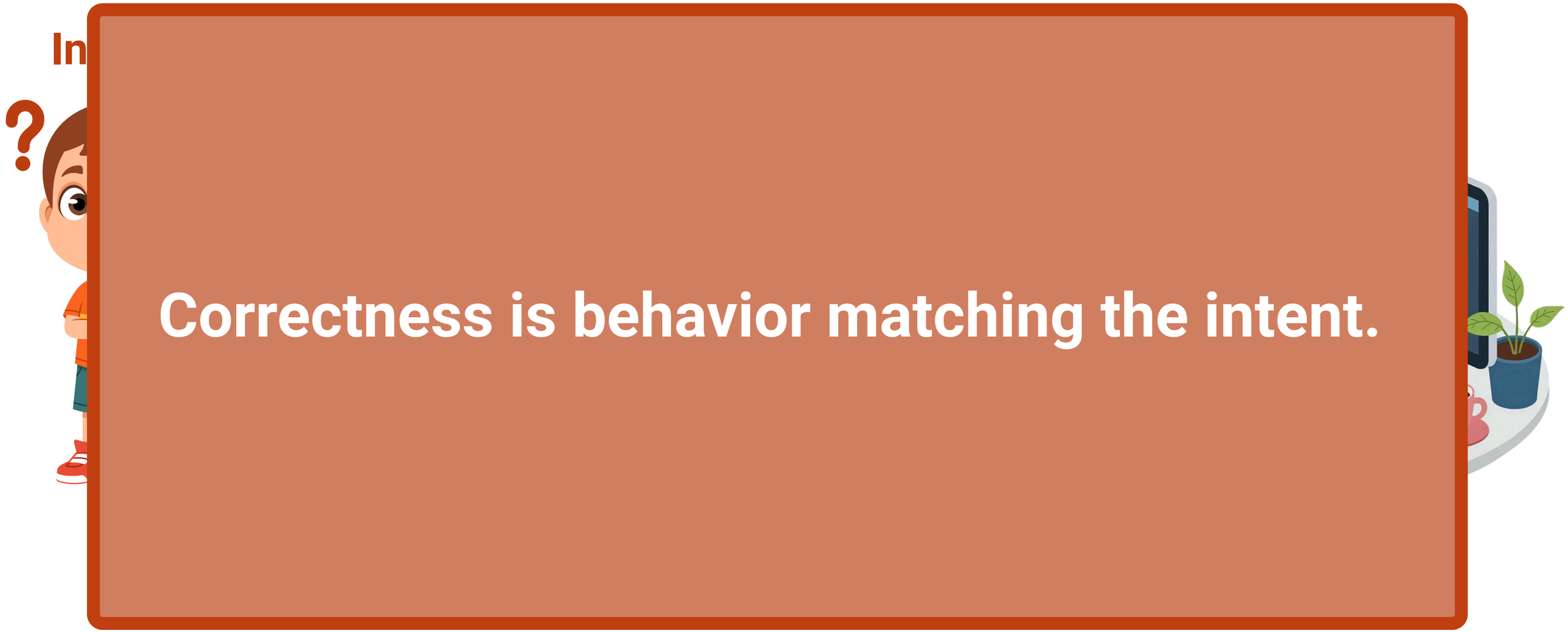
What’s Correctness, anyway?

Intent

Code

Behavior

What’s Correctness, anyway?



How do you measure correctness?

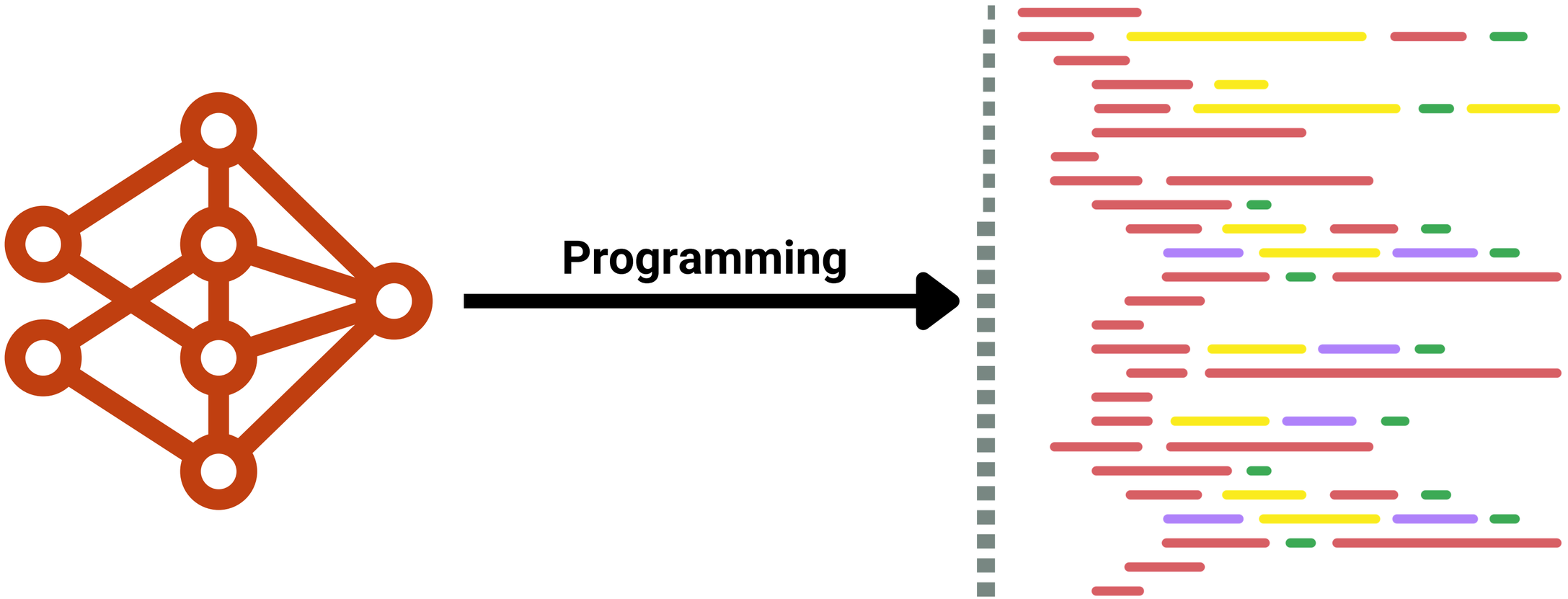
Intent

Code

Behavior

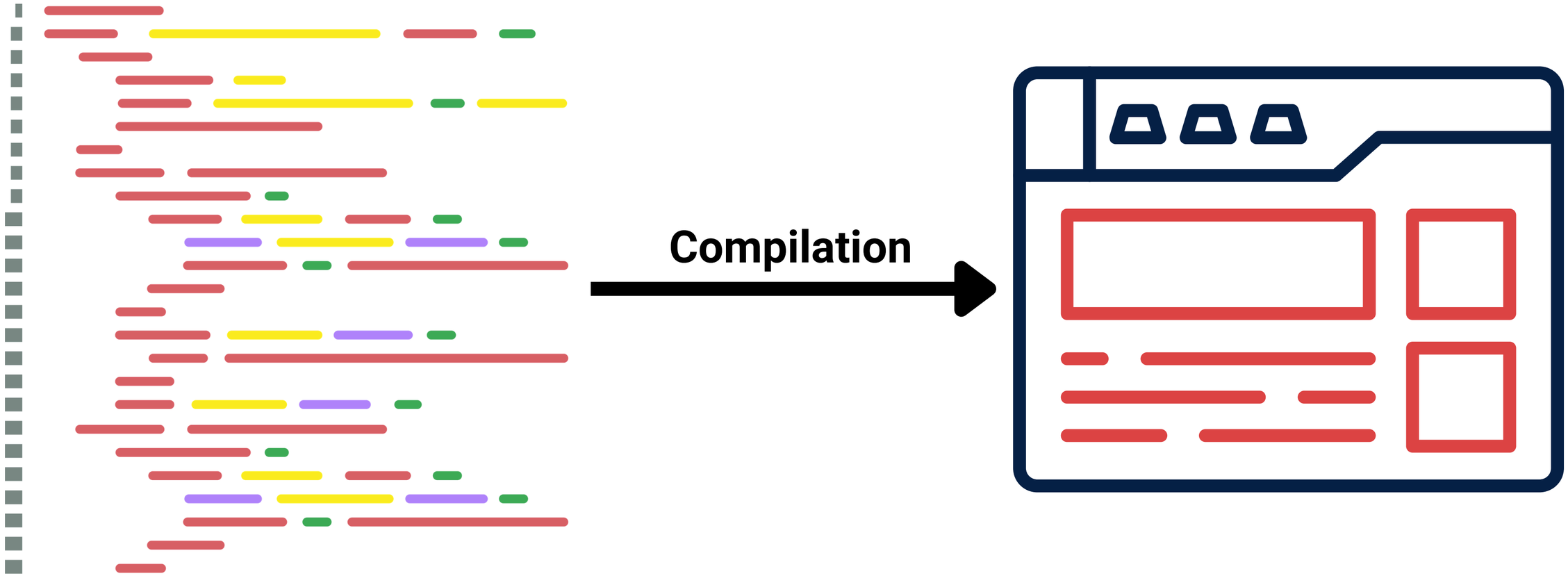
How do you measure correctness?

Tinkering



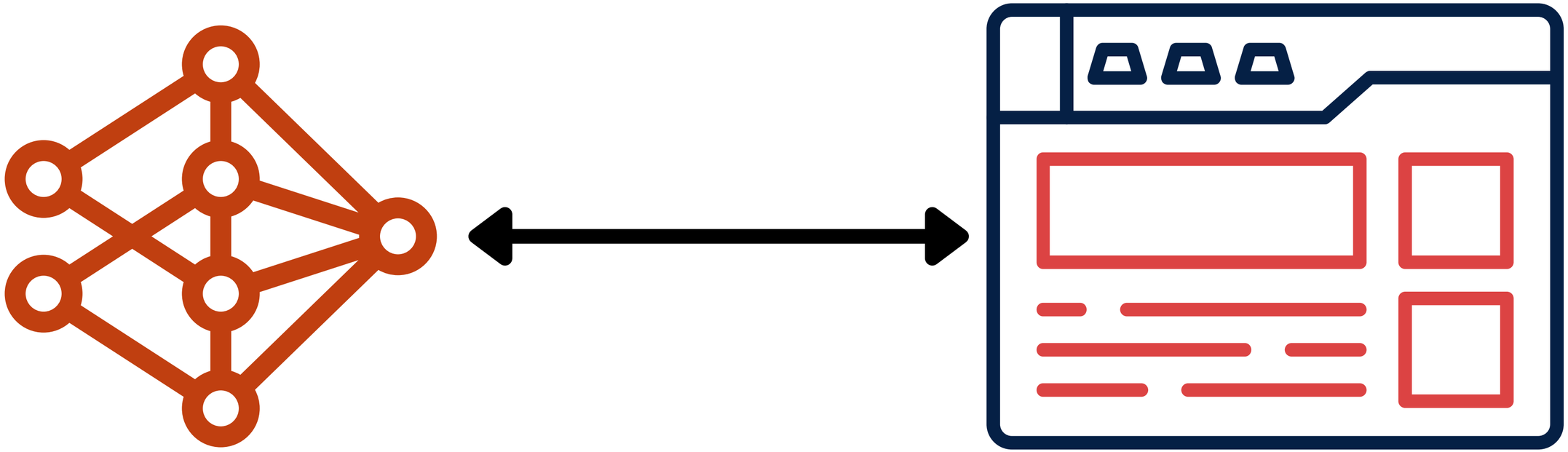
How do you measure correctness?

Tinkering



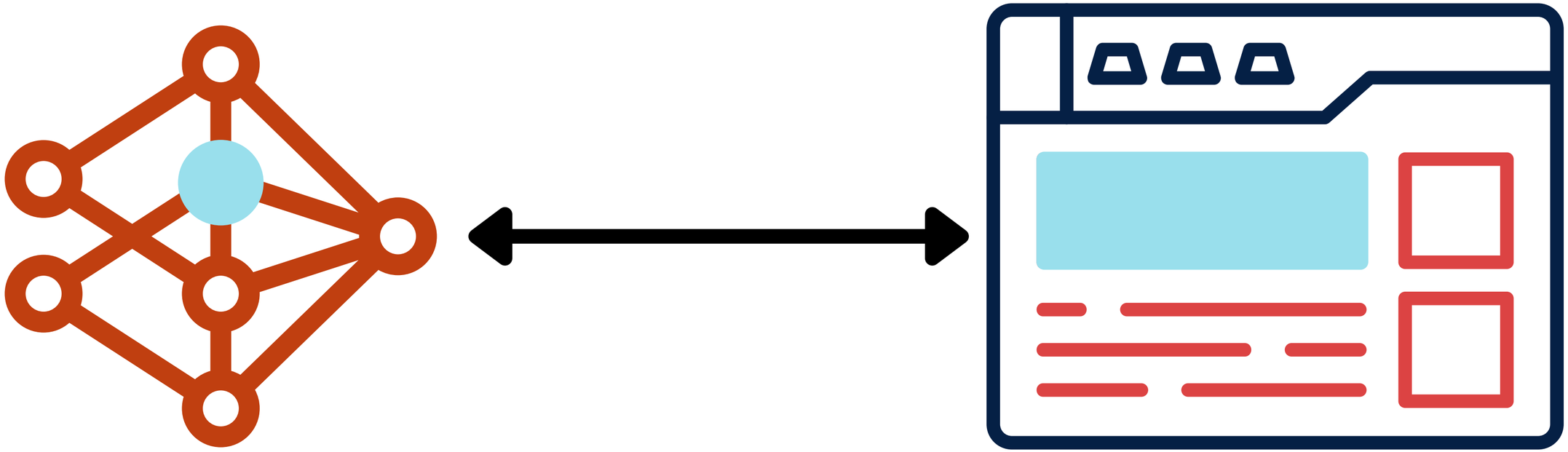
How do you measure correctness?

Tinkering



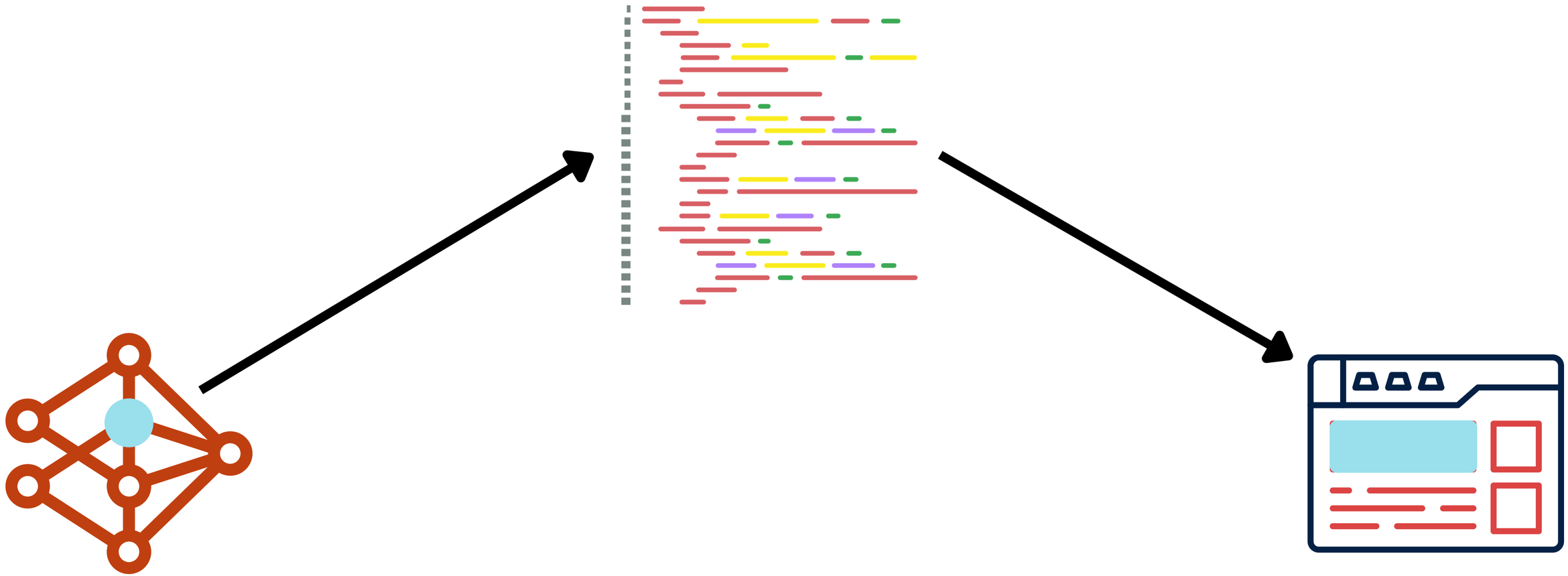
How do you measure correctness?

Tinkering



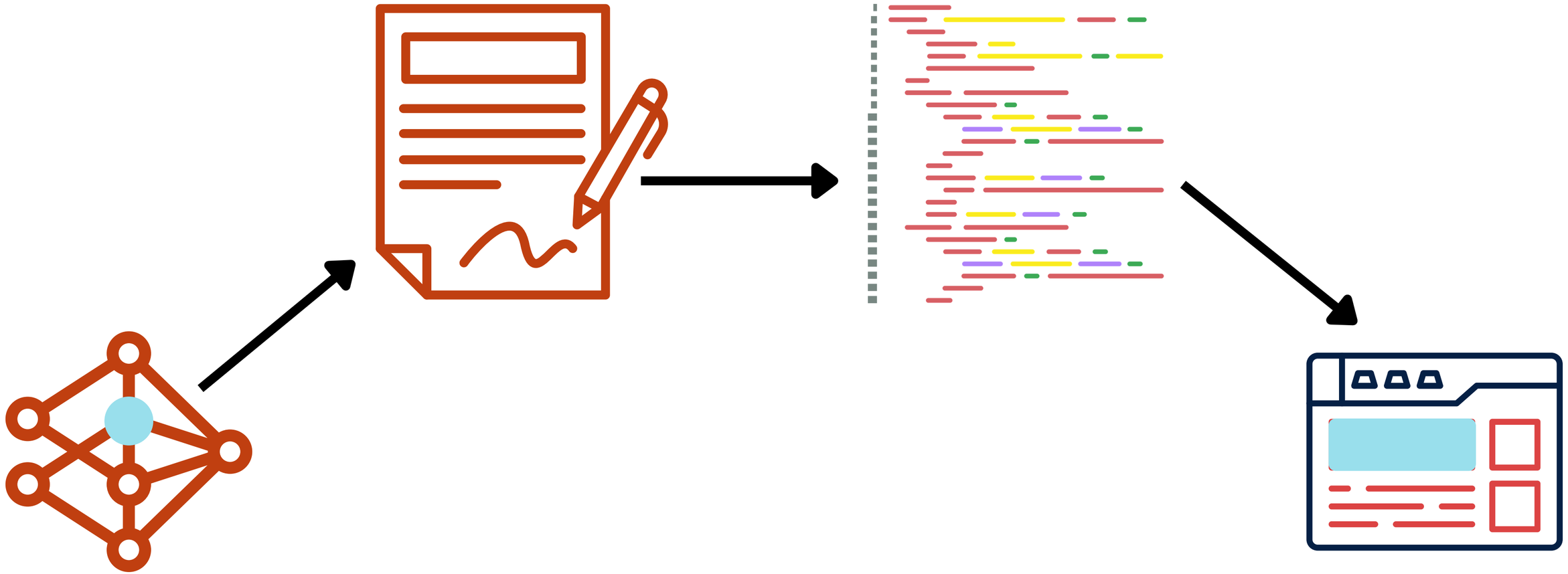
How do you measure correctness?

Tinkering



How do you measure correctness?

Tinkering



How do you measure correctness?

Tinkering

How do you measure correctness?

Tinkering

Verifiability is the Limit.

Posted on 2025-03-09 :: 9 min read :: Tags: software engineering, ai

LLMs have created an enormous turmoil within the software engineering community within the past 5 years, much of it revolving around one central question, **what is the future of our profession?**

Verifiability is the Limit.

Posted on 2025-03-09 :: 9 min read :: Tags: software engineering, ai

LLMs have created an enormous turmoil within the software engineering community within the past 5 years, much of it revolving around one central question, **what is the future of our profession?**

> ...Every piece of software in the world has to be “correct” with respect to some notion of correctness. Before LLMs, our time was split between writing, reading and verifying code. With LLMs, we are able to offload code writing phase to LLMs, but the verification process cannot be offloaded, only pushed up to a different layer...

Verifiability is the Limit.

Posted on 2025-03-09 :: 9 min read :: Tags: software engineering, ai

LLMs have created an enormous turmoil within the software engineering community within the past 5 years, much of it revolving around one central question, **what is the future of our profession?**

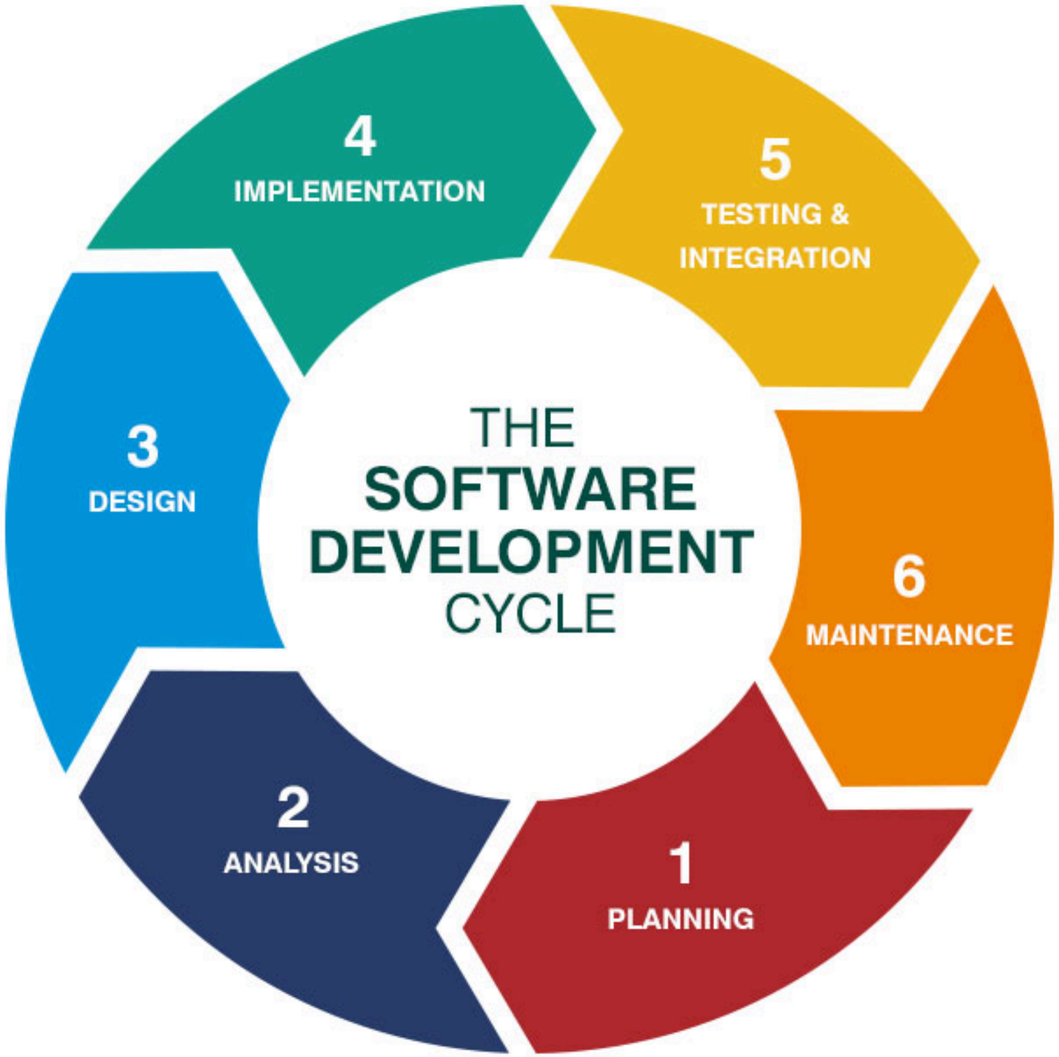
Going back to the title, **verifiability is the limit.**

It is the limit that tells us what we cannot implement via LLMs, and it cannot be solved with agentic approaches. A security agent might in theory add security checks to an application, but such a check is worthless without verifying it as the person who intends to produce the code. A testing agent might add tests to a program, but the tests themselves are meaningless until we check to see they correspond to our intent.

So far, I theorized about the LLMs and the way we produced software to make my point. **Now, I shall preach about what I think we should do.**

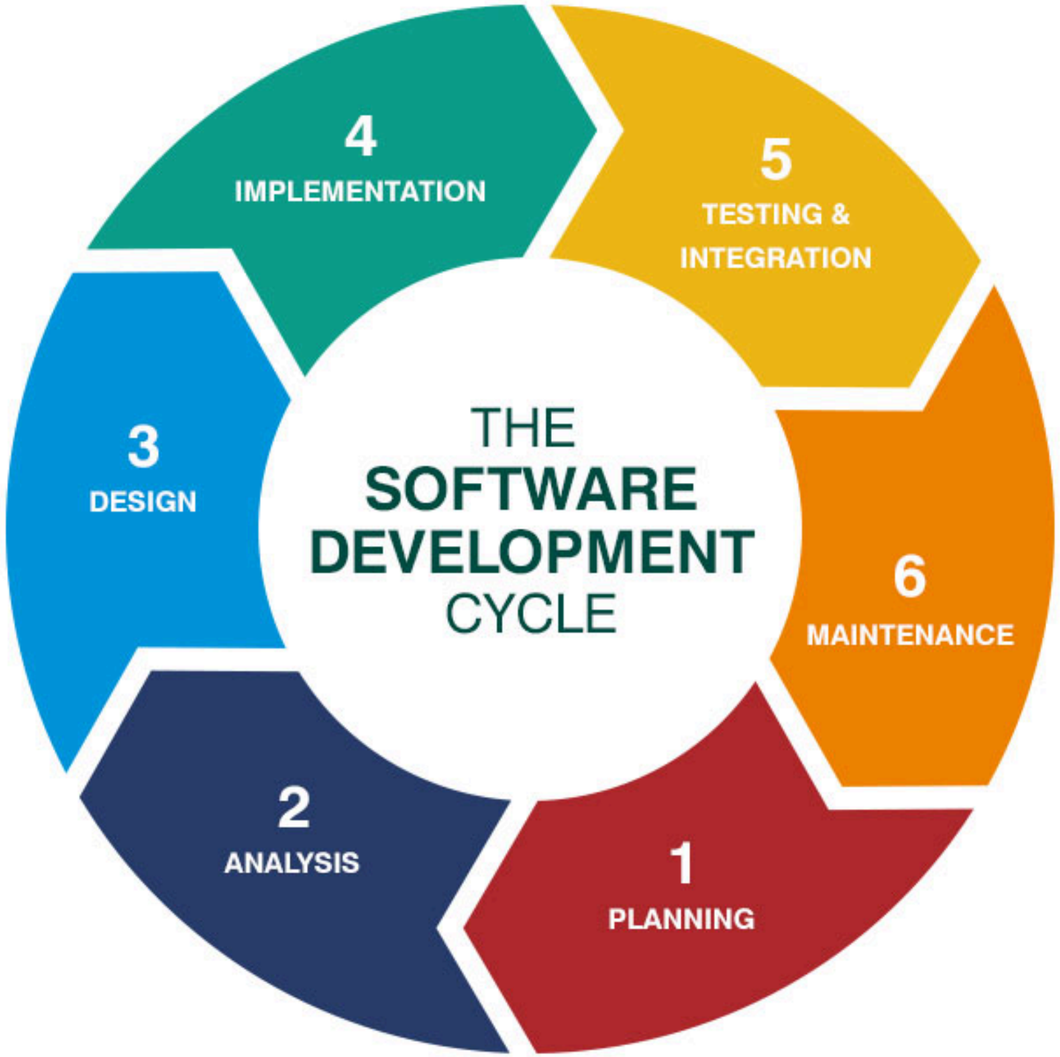
If verifiability is the limit, if it's the bottleneck of using LLMs for programming, the natural question is **how do we raise the limit, how do we make it easier to verify?**

How do you verify software?

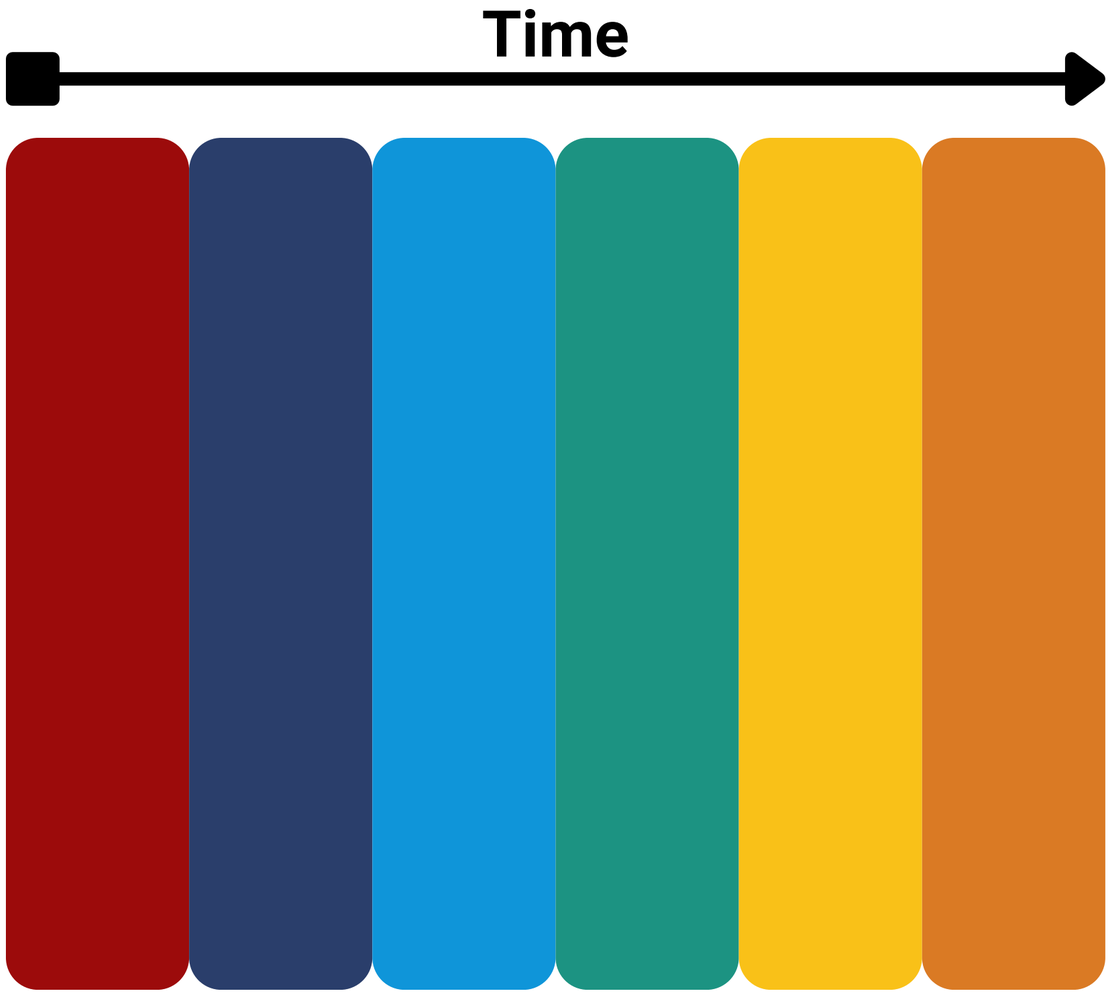


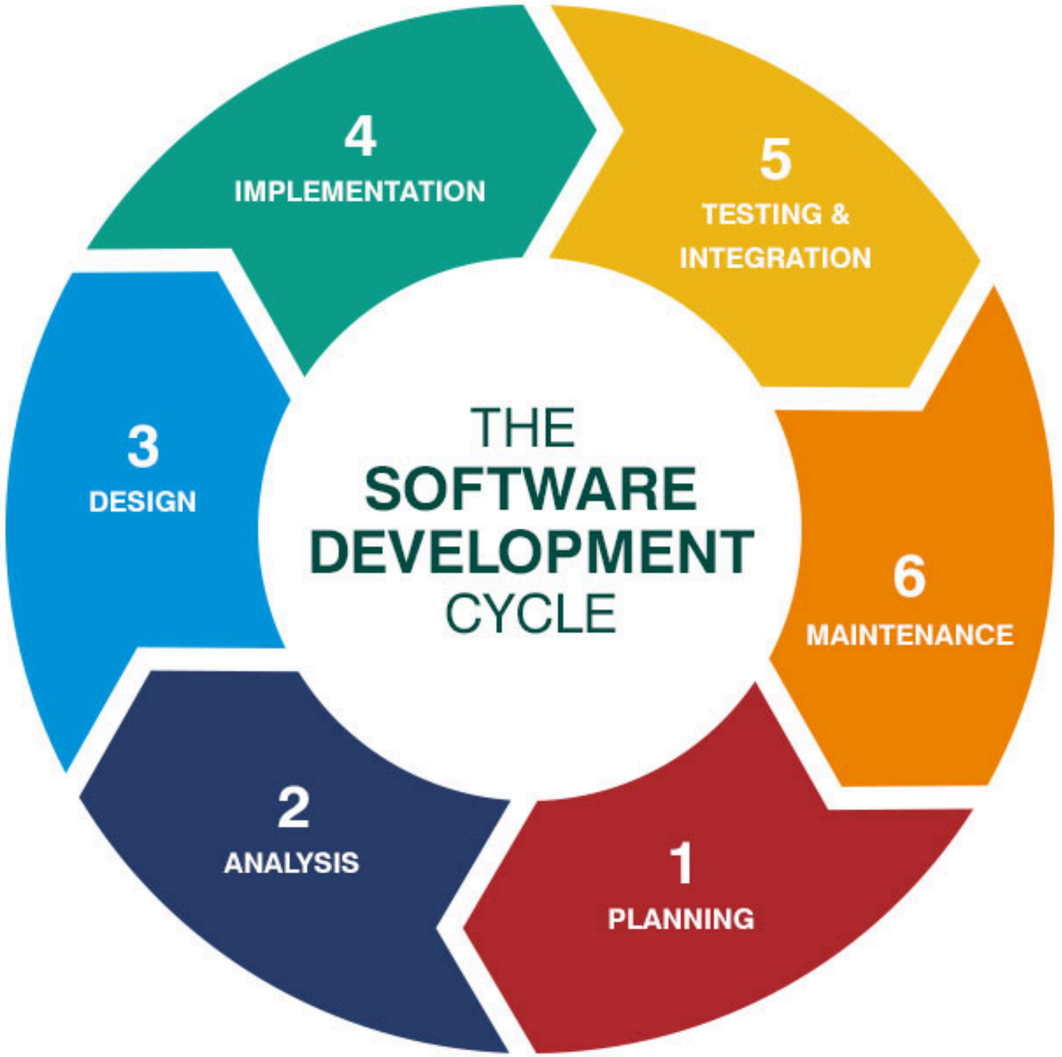
How do you verify software?

- Code review
- Software testing
- Observability

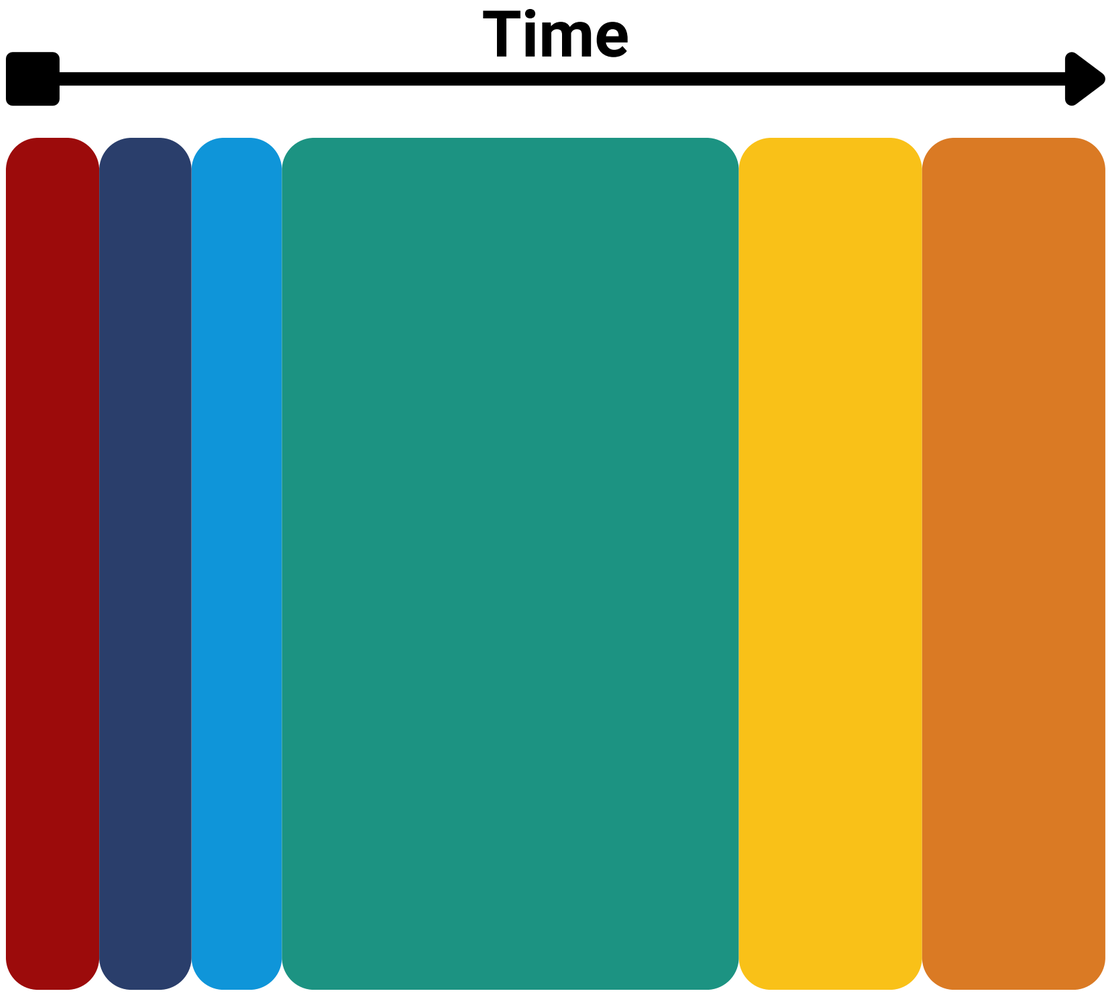


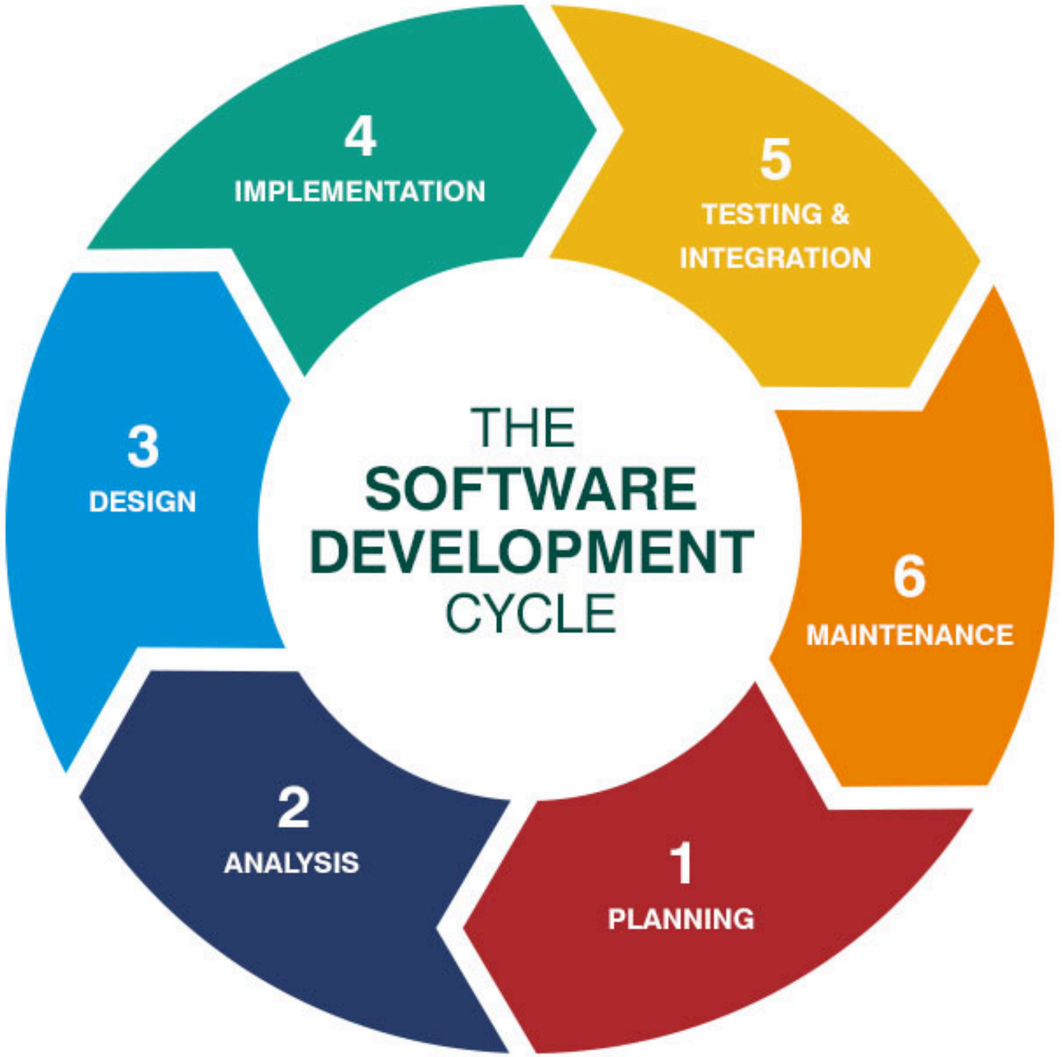
How do you verify software?



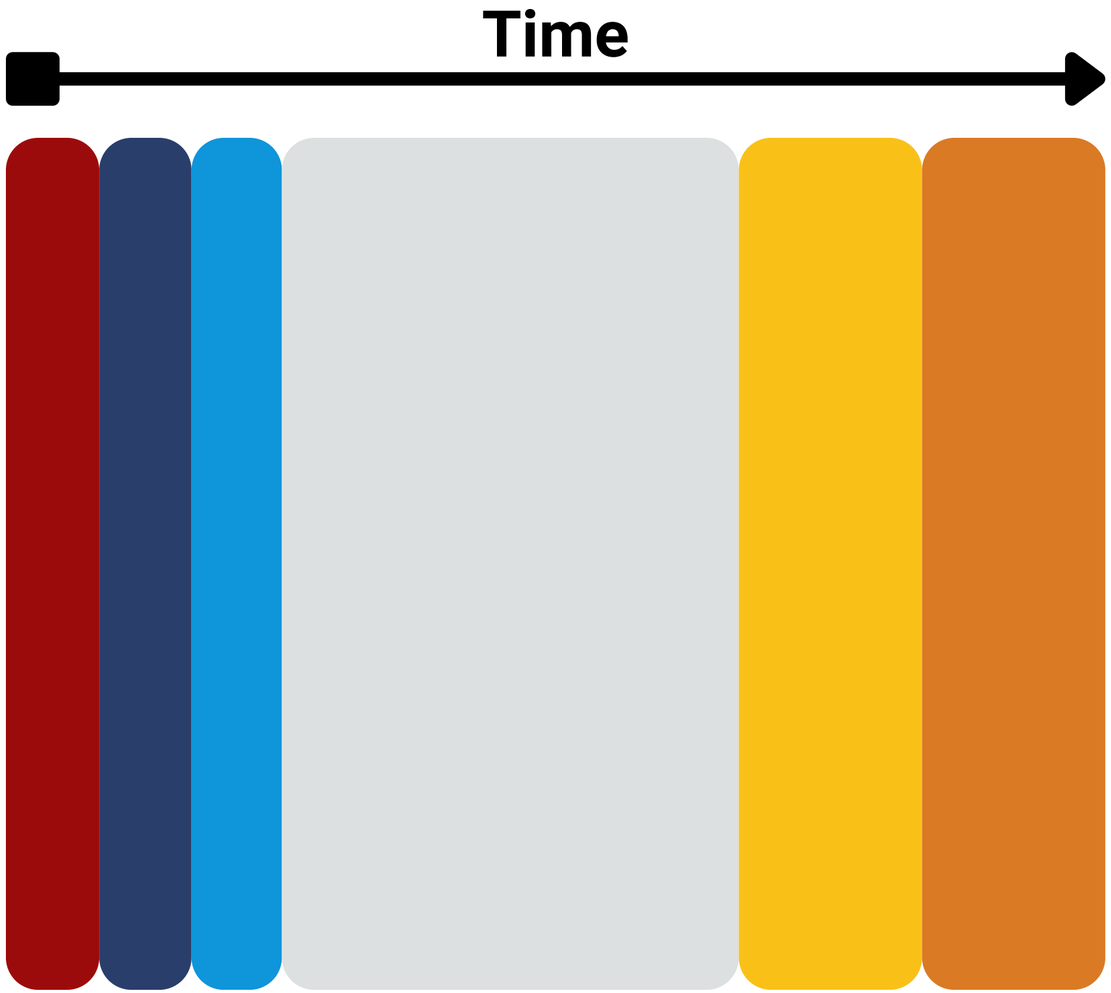


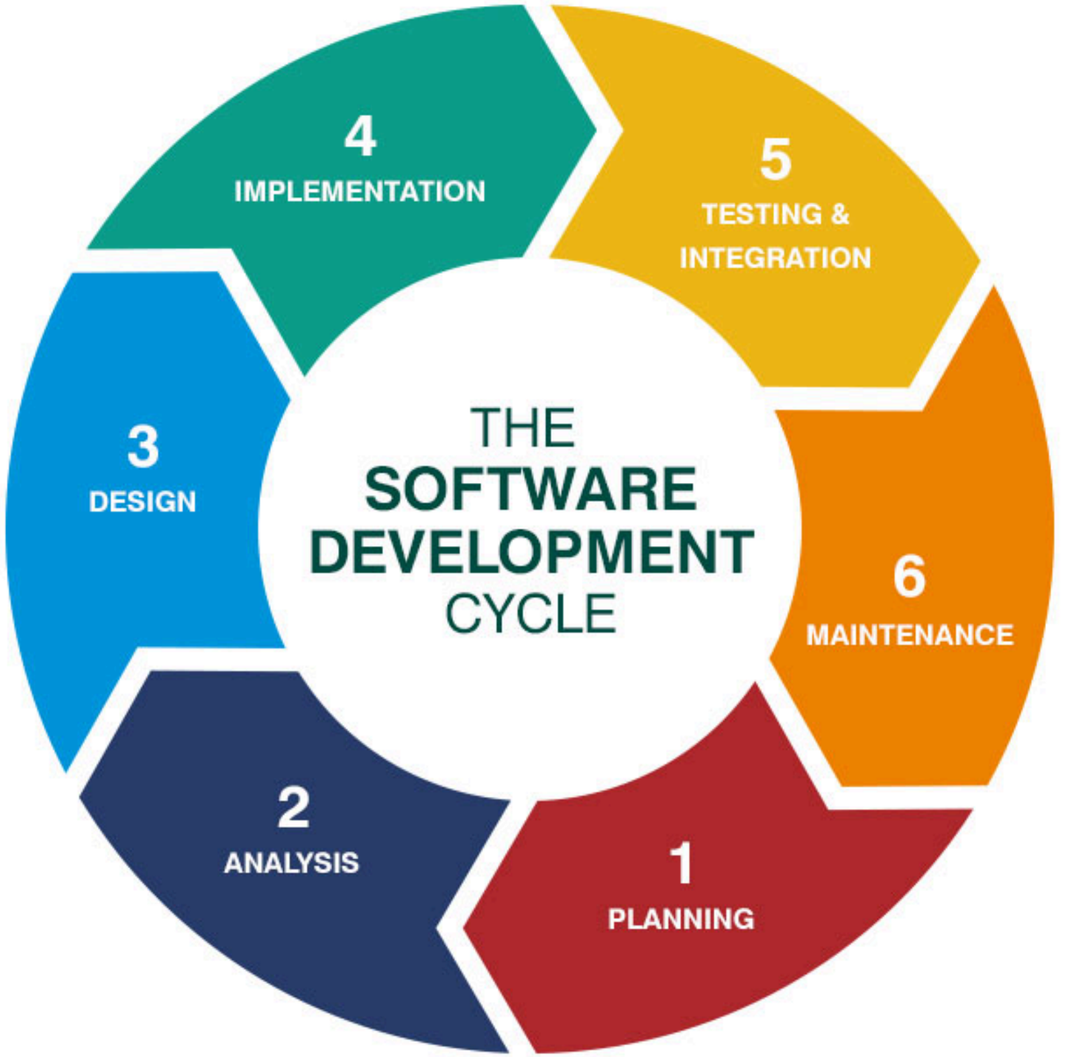
How do you verify software?



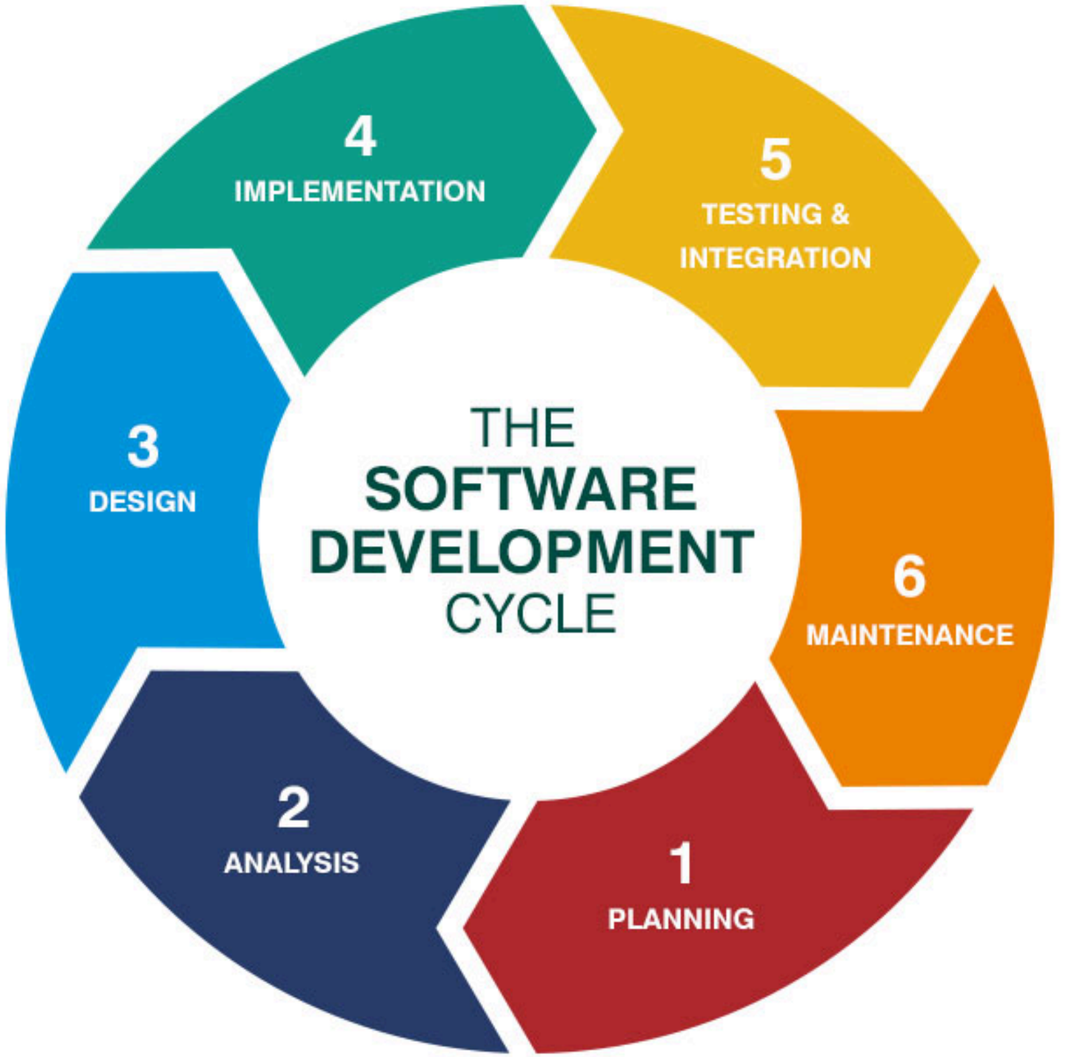


How do you verify software?

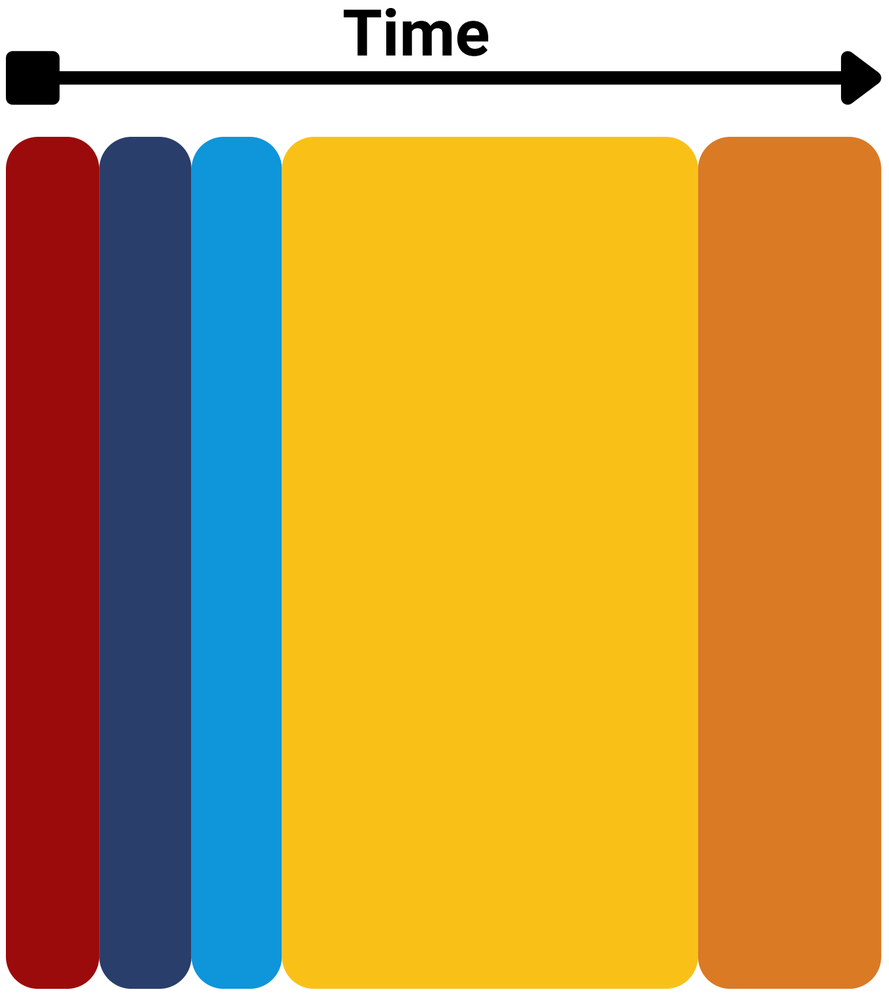


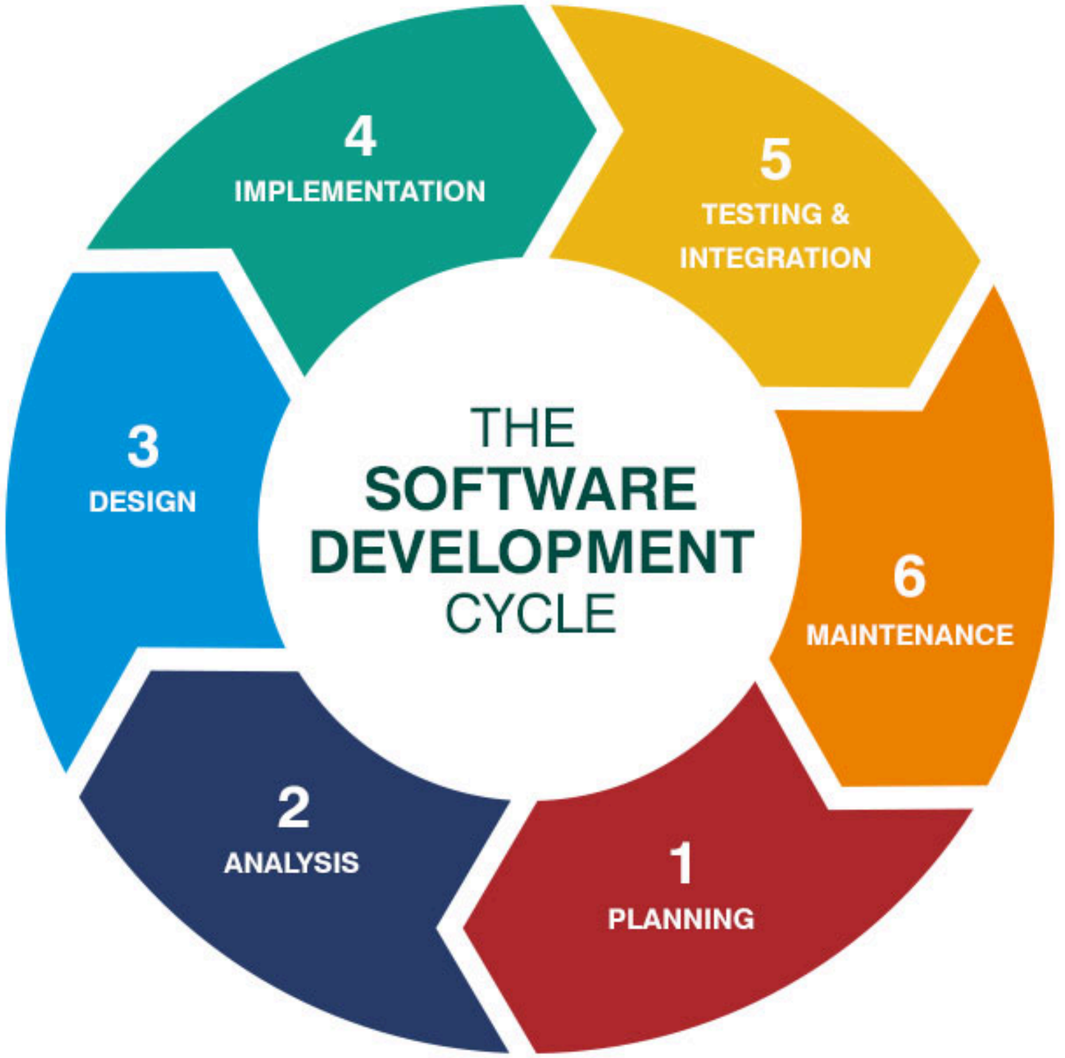


How do you verify software?



How do you verify software?

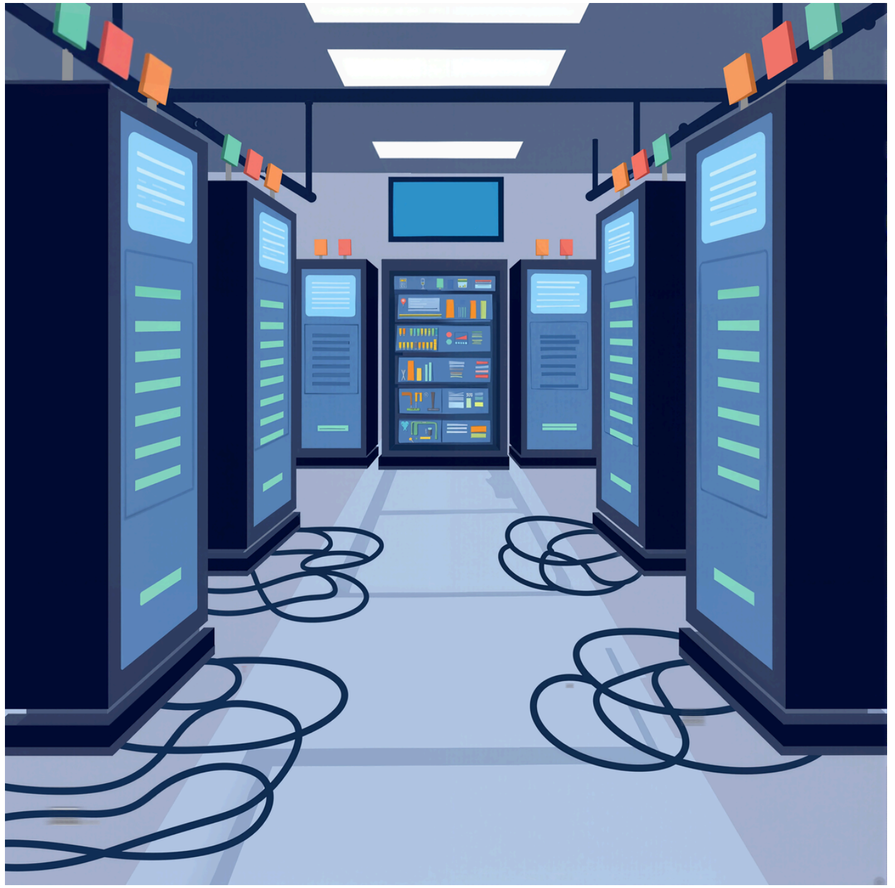




How do you verify software?

Code

Behavior



How do you specify software?

Tinkering:

f(x) = y,

f(x + 1) = y + 1

f(x + 10) = y + 10

f(x - 3) = y - 3

f(x - -3) = y + 3

How do you specify software?

Tinkering:

f(x) = y,

f(x + 1) = y + 1

f(x + 10) = y + 10

f(x - 3) = y - 3

f(x - -3) = y + 3

Examples:

f(x) = y,

t(f(x)) < 5ms,

sql(q(x)) = Error

How do you specify software?

Tinkering:

f(x) = y,

f(x + 1) = y + 1

f(x + 10) = y + 10

f(x - 3) = y - 3

f(x - -3) = y + 3

Examples:

f(3) = 3

f(4 + 2) = 6

f(0 + 0) = 0...

Properties:

∀ x. p(f(x)).

∀ x, y. f(x + y) = f(x) + y

Property-Based Testing

$\forall x.\ p(f(x))$.

Idempotency

```
sort(list) = sort(sort(list))
```

Monotonicity

```
len(list) < len(append(list, x))
```

Roundtrip

```
decompress(compress(data)) = data
```

Model

```
reference_fn(x) = fn(x)
```

american fuzzy lop 0.47b (readpng)

- VLC
- Sqlite
- Vim
- Pure-FTPd
- Bftpd
- Tcpdump
- ProFTPd
- Gifsicle
- FFmpeg
- Glibc
- FreeRDP
- GNOME
- QEMU
- GNU coreutils
- PostgreSQL
- Node.js
- libjxl
- Perl
- zlog

SQLANCER

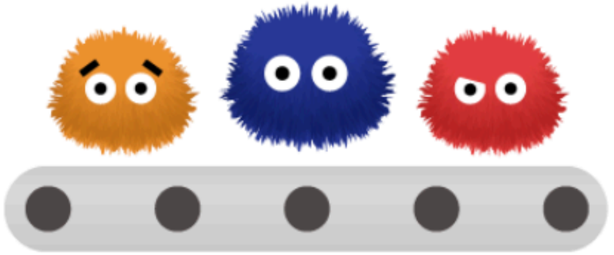
- SQLite (179)
- MySQL (19)
- PostgreSQL (13)
- MariaDB (1)
- CockroachDB (56)
- TiDB (55)
- DuckDB (73)

Csmith

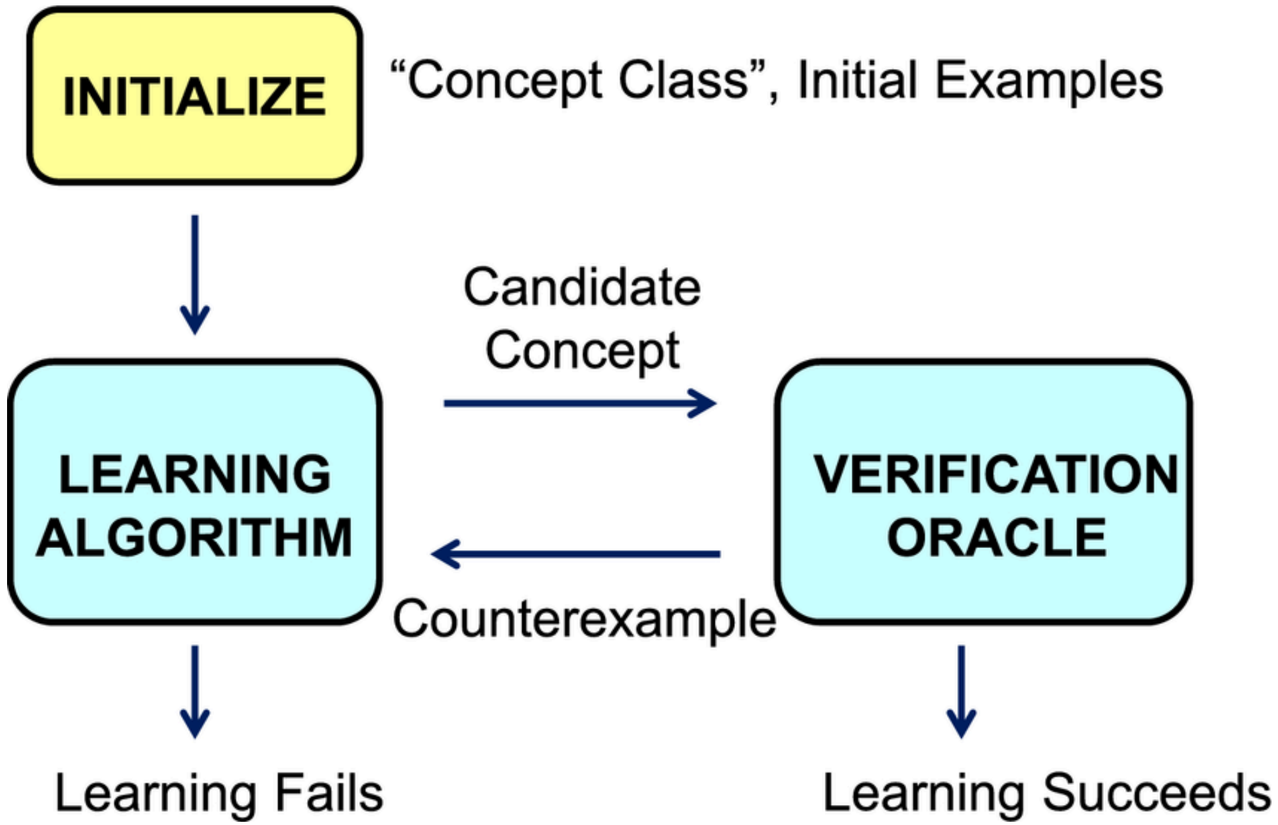
	GCC	LLVM
Front end	0	10
Middle end	49	75
Back end	17	74
Unclassified	13	43
Total	79	202

Geneva: **Evolving Censorship Evasion Strategies**

Fray *test*



Counterexample-Guided Program Synthesis



Scaling long-running autonomous coding

Jan 14, 2026 by Wilson Lin

Andreas Kling — Mon, 23 Feb 2026

Ladybird adopts Rust, with help from AI

Techno-Optimism · Tech

AI Just Ported SimCity in 4 Days Without Reading the Code

Building a C compiler with a team of parallel Claudes

How AI helps break the cost barrier to COBOL modernization

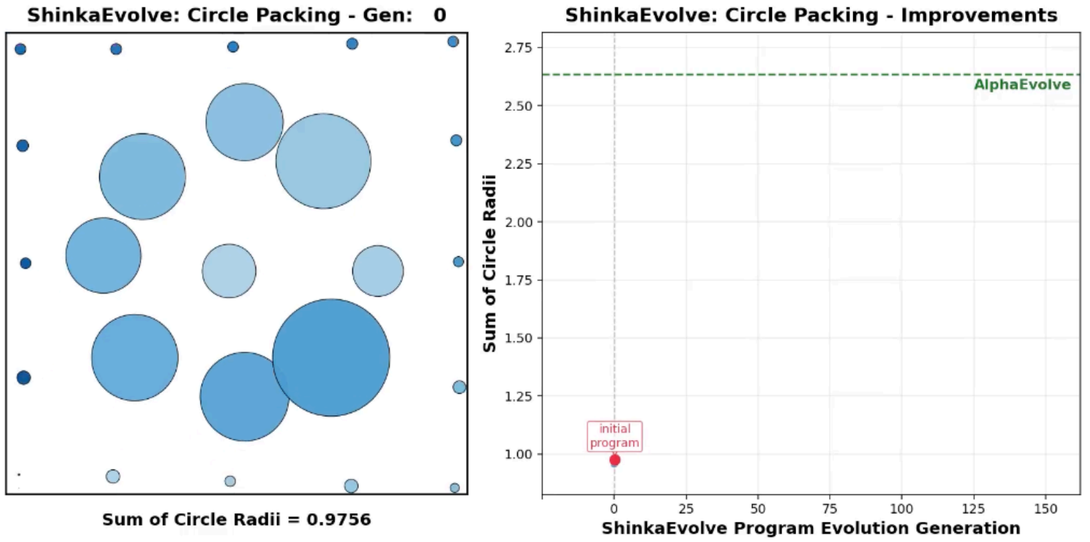
ENGINEERING

From hand-tuned Go to self-optimizing code: Building BitsEvolve

CODE-PROFILING AI PERFORMANCE

ShinkaEvolve: Evolving New Algorithms with LLMs, Orders of Magnitude More Efficiently

September 25, 2025



[Submitted on 7 Oct 2025 (v1), last revised 10 Oct 2025 (this version, v3)]

Barbarians at the Gate: How AI is Upending Systems Research

Audrey Cheng, Shu Liu, Melissa Pan, Zhifei Li, Bowen Wang, Alex Krentsel, Tian Xia, Mert Cemri, Jongseok Park, Shuo Yang, Jeff Chen, Lakshya Agrawal, Aditya Desai, Jiarong Xing, Koushik Sen, Matei Zaharia, Ion Stoica

[Submitted on 31 Oct 2025 (v1), last revised 17 Nov 2025 (this version, v3)]

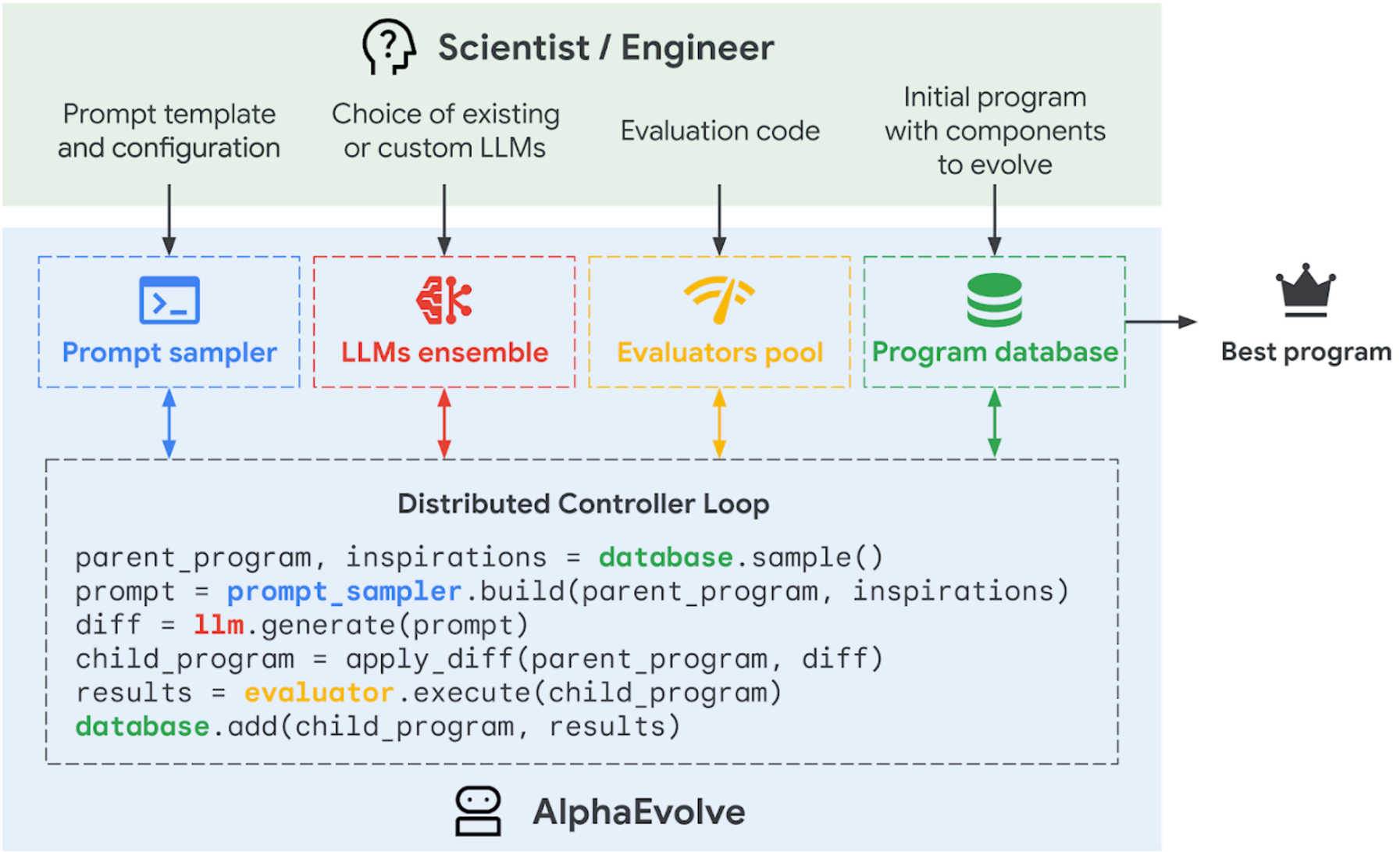
Glia: A Human-Inspired AI for Automated Systems Design and Optimization

Pouya Hamadanian, Pantea Karimi, Arash Nasr-Esfahany, Kimia Noorbakhsh, Joseph Chandler, Ali ParandehGheibi, Mohammad Alizadeh, Hari Balakrishnan

[Submitted on 19 Jul 2025 (v1), last revised 24 Oct 2025 (this version, v4)]

AlgoTune: Can Language Models Speed Up General-Purpose Numerical Programs?

Ori Press, Brandon Amos, Haoyu Zhao, Yikai Wu, Samuel K. Ainsworth, Dominik Krupke, Patrick Kidger, Touqir Sajed, Bartolomeo Stellato, Jisun Park, Nathanael Bosch, Eli Meril, Albert Steppi, Arman Zharmagambetov, Fangzhao Zhang, David Perez-Pineiro, Alberto Mercurio, Ni Zhan, Talor Abramovich, Kilian Lieret, Hanlin Zhang, Shirley Huang, Matthias Bethge, Ofir Press


Scientist / Engineer
Prompt template and configuration
Choice of existing or custom LLMs
Evaluation code
Initial program with components to evolve
Prompt sampler
LLMs ensemble
Evaluators pool
Program database
Best program
Distributed Controller Loop
parent_program, inspirations = database.sample()
prompt = prompt_sampler.build(parent_program, inspirations)
diff = llm.generate(prompt)
child_program = apply_diff(parent_program, diff)
results = evaluator.execute(child_program)
database.add(child_program, results)
AlphaEvolve

Well I already gave away the spoiler, **why are we fixating so much on prompts when we could just use programs instead?**

People already use LLMs for translating code between languages, but they still rely on the LLM to correctly do the translation even though the translation has one of the most popular correctness properties of all the time, equivalence that we can programmatically test for. This problem is traditionally called translation validation, and is far from being solved, we definitely do not have the capability of proving two programs will behave exactly the same way for every single input (it's undecidable, so we will never have that capability). We, however, have the capability of doing differential testing, which has given rise to some of the greatest testing successes of the industry by finding bugs in C compilers and SQL engines, as well as being a central piece of modern practices for developing high assurance software.

I posit that some of the most interesting use cases of AI will emerge from using programs as sources instead of focusing on prompts.

- Translating entire libraries and programs across ecosystems through a continuous feedback loop that is based on counterexample guided program synthesis.
- Optimizing programs for cache friendliness or memory consumption, securing them against side channel attacks, automatically switching between equivalence programs with different characteristics of readability or density, auto-parallelization of single-threaded programs.
- Breaking existing abstractions at lower levels by switching them with verifiable translations, instead of accepting the bytecode or assembly emitted by the compiler as a given, using the instrumentation and observations from the program for doing machine-level verifiable optimizations similar to what a JIT compiler would do.

Verifiability is the Limit

Breaking Verifiable Abstractions

Verification is Not the Silver Bullet

Test, don't (just) verify

LLMs could be, but shouldn't be compilers

The Mechanics of Autonomous Software Translation

Specifiability is the Leverage

alperenkeles.com/posts

Demo