Testing and Analysis in the AI Era

Cristian Cadar

SOFTWARE RELIABILITY
GROUP

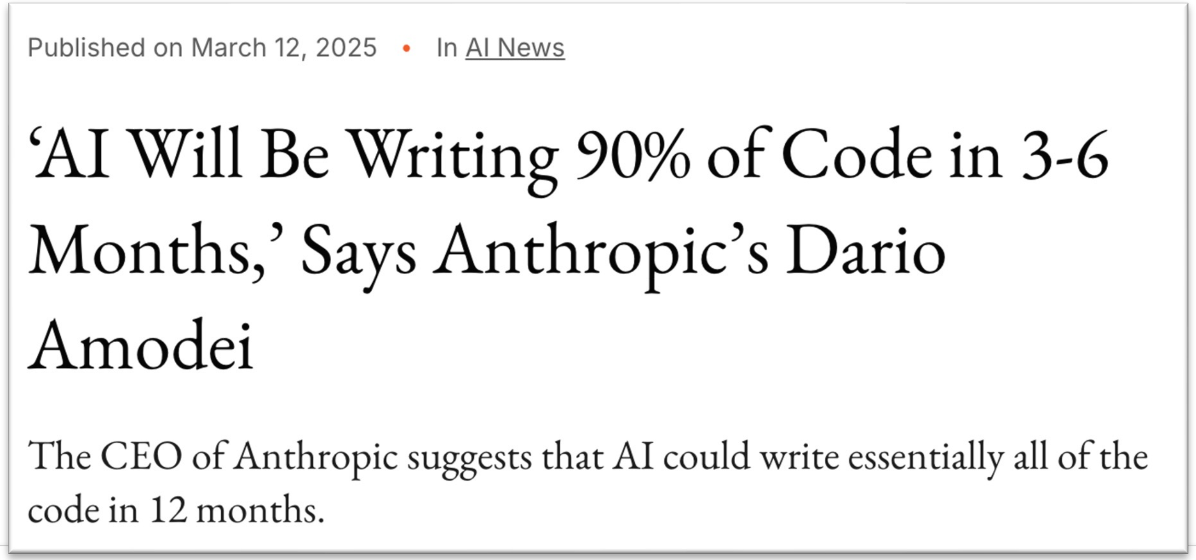
Department of Computing
Imperial College London

Funded by Engineering and Physical Sciences Research Council

erc
European Research Council
Established by the European Commission

Keynote at FM 2026
21 May 2026, Tokyo, Japan

Impact of Generative AI on Software Development



Published on March 12, 2025 • In AI News

‘AI Will Be Writing 90% of Code in 3-6 Months,’ Says Anthropic’s Dario Amodei

The CEO of Anthropic suggests that AI could write essentially all of the code in 12 months.

Over 25% of Google’s code is now written by AI—and CEO Sundar Pichai says it’s just the start

BY **GREG MCKENNA**
NEWS FELLOW
October 30, 2024 at 11:14 AM EDT

Amazon CEO Andy Jassy says the company's AI assistant Amazon Q has dramatically cut the time needed for Java updates.

Mark Zuckerberg wants AI to do half of Meta's coding by 2026

The comments came during a discussion between Zuckerberg and Microsoft CEO Satya Nadella.

By Cecily Mauran April 30, 2025

TECH

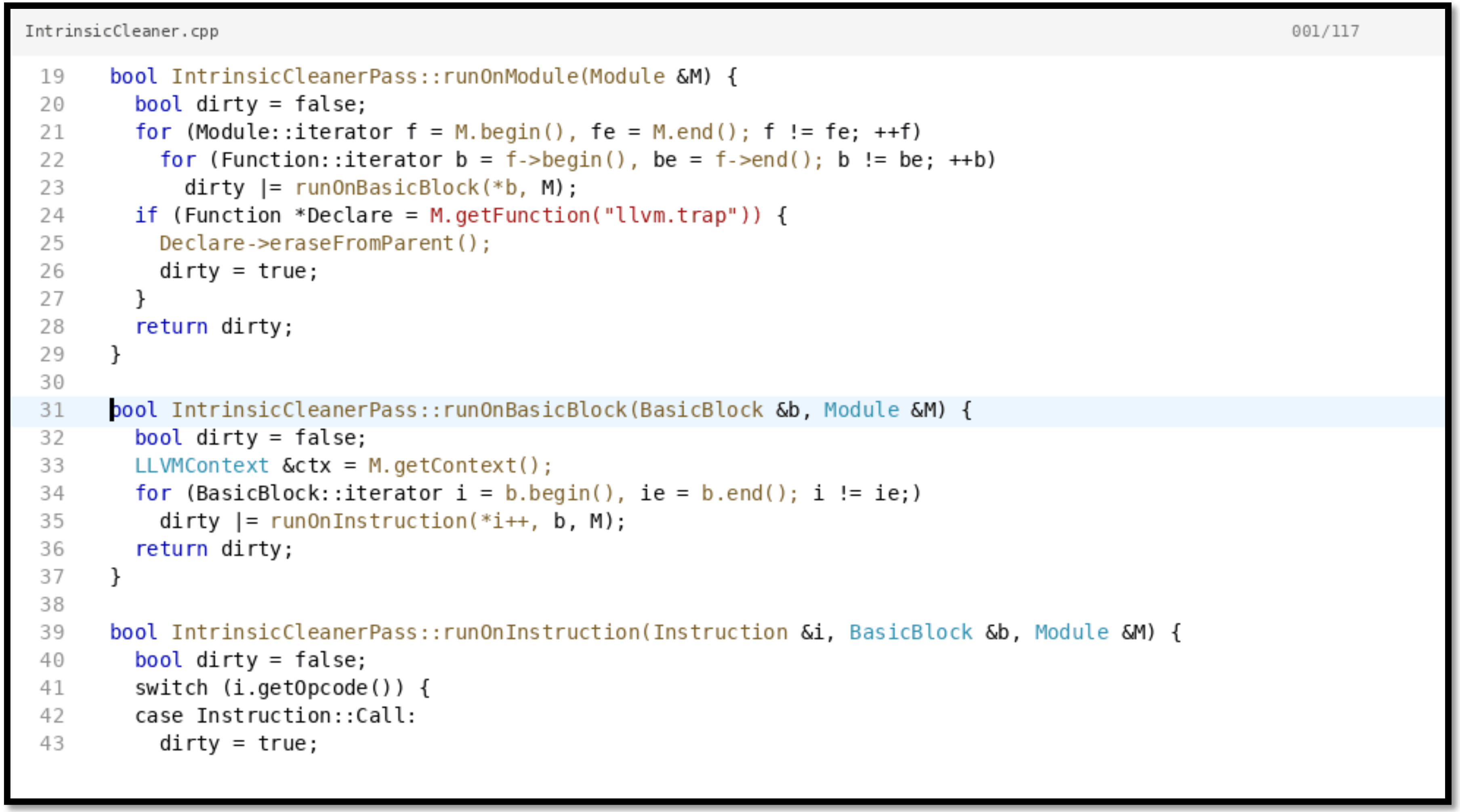
Satya Nadella says as much as 30% of Microsoft code is written by AI

PUBLISHED TUE, APR 29 2025•9:33 PM EDT | UPDATED TUE, APR 29 2025•9:58 PM EDT

Accelerated shift from code writing to code reviewing and validation

Code reviewing is hard, and even more so for the new generation of “AI-native” code developers

AI Further Accelerates Development



```
bool IntrinsicCleanerPass::runOnModule(Module &M) {
  bool dirty = false;
  for (Module::iterator f = M.begin(), fe = M.end(); f != fe; ++f)
    for (Function::iterator b = f->begin(), be = f->end(); b != be; ++b)
      dirty |= runOnBasicBlock(*b, M);
  if (Function *Declare = M.getFunction("llvm.trap")) {
    Declare->eraseFromParent();
    dirty = true;
  }
  return dirty;
}

bool IntrinsicCleanerPass::runOnBasicBlock(BasicBlock &b, Module &M) {
  bool dirty = false;
  LLVMContext &ctx = M.getContext();
  for (BasicBlock::iterator i = b.begin(), ie = b.end(); i != ie;)
    dirty |= runOnInstruction(*i++, b, M);
  return dirty;
}

bool IntrinsicCleanerPass::runOnInstruction(Instruction &i, BasicBlock &b, Module &M) {
  bool dirty = false;
  switch (i.getOpcode()) {
  case Instruction::Call:
    dirty = true;
```

Can Formal Methods, Software Testing and Program Analysis Tools Help?

Clang Static Analyzer

cbmc

{PMD}

AFL++

SpotBugs

Infer

sanitizers

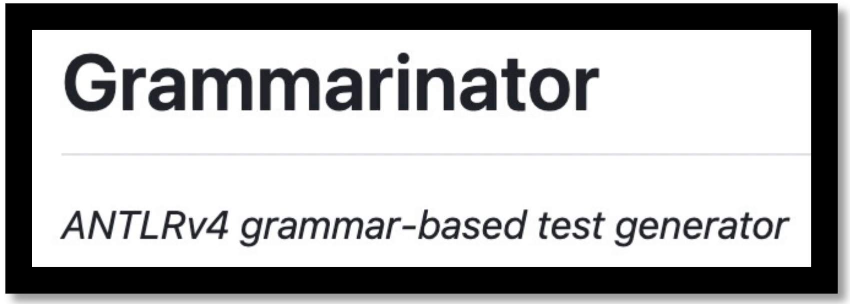
KLEE

FINDING BUGS WITH STYLE

EVOSUITE

Valgrind

ESLint


Grammarinator

ANTLRv4 grammar-based test generator

Designed for specific programming languages and system types!

Designed for whole program analysis

Designed for, or in practice limited to, finding generic bugs

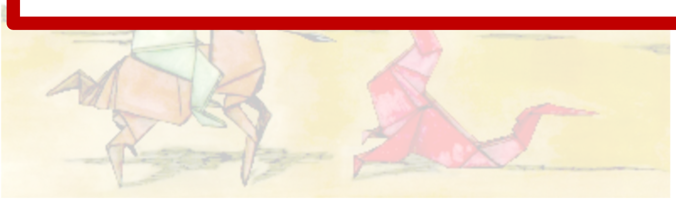
Clang Static Analyzer

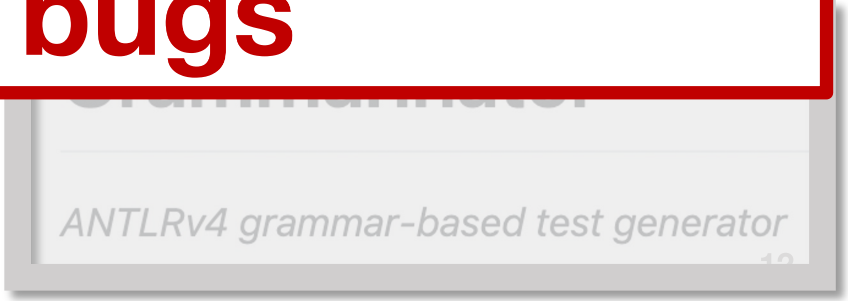
sanitizers

KLEE

Infer

ESLint





ANTLRv4 grammar-based test generator

cbmc

We need support for the full diversity of PLs and system types!

Clang Static Analyzer

Infer

We need fast incremental techniques!

FINDING BUGS WITH STYLE

EVOSUITE

We need to reason about semantics!

Grammarinator

ANTLRv4 grammar-based test generator

ESLint

Dynamic Symbolic Execution (DSE)

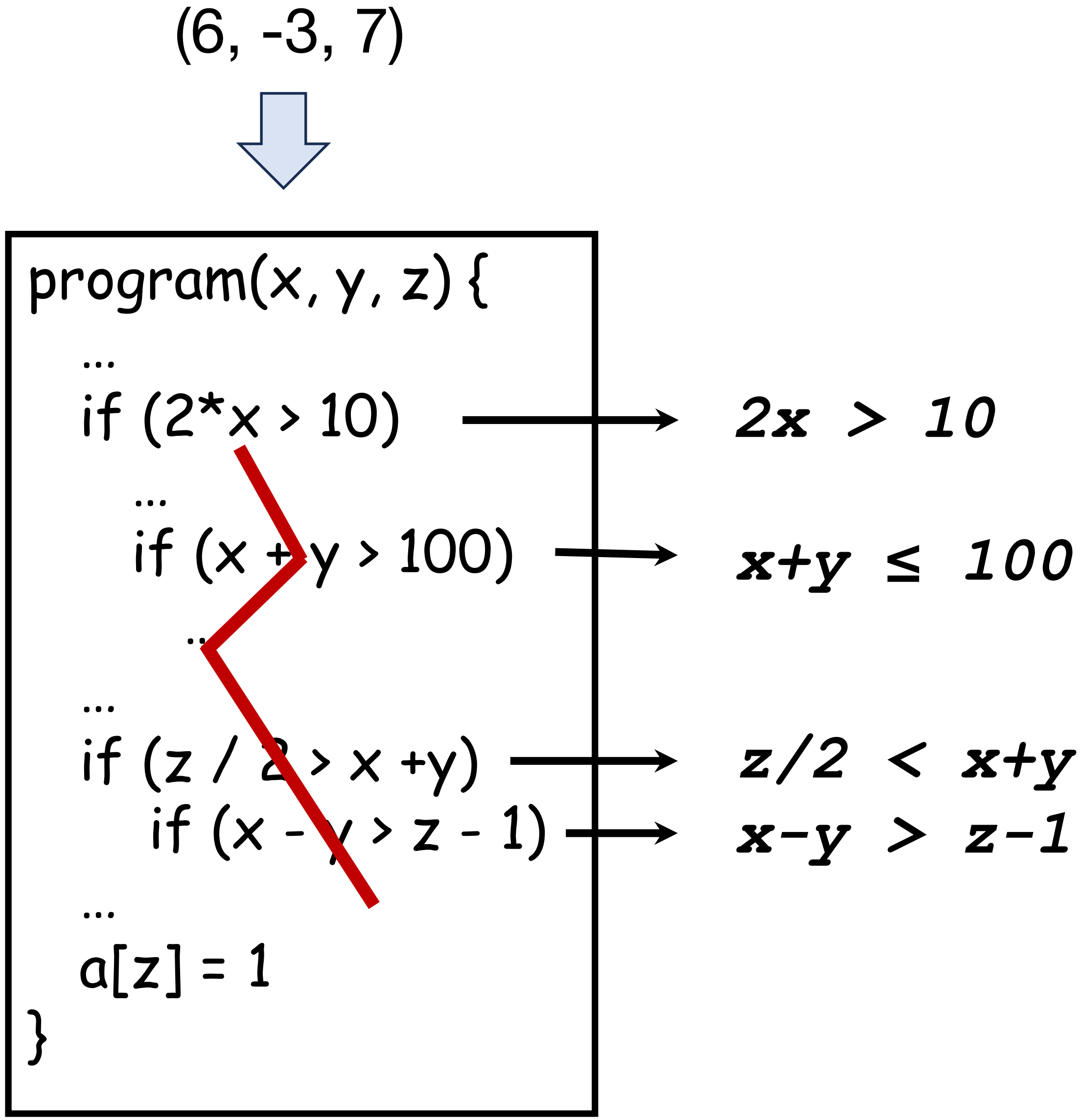
Technique for ***automatically exploring and analysing program paths***

Models program paths as ***mathematical constraints,*** *to be analysed by an* ***SMT solver***

Applications in:

- Bug finding
- Test generation
- Vulnerability detection and exploitation
- Equivalence checking
- Debugging
- Program repair
- Bounded verification
- etc. etc.

Dynamic Symbolic Execution or Concolic Execution



(6, -3, 7)

```
program(x, y, z) {
   ...
   if (2*x > 10)
      ...
      if (x + y > 100)
         ...
   ...
   if (z / 2 > x +y)
      if (x - y > z - 1)
   ...
   a[z] = 1
}
```

$2x > 10$

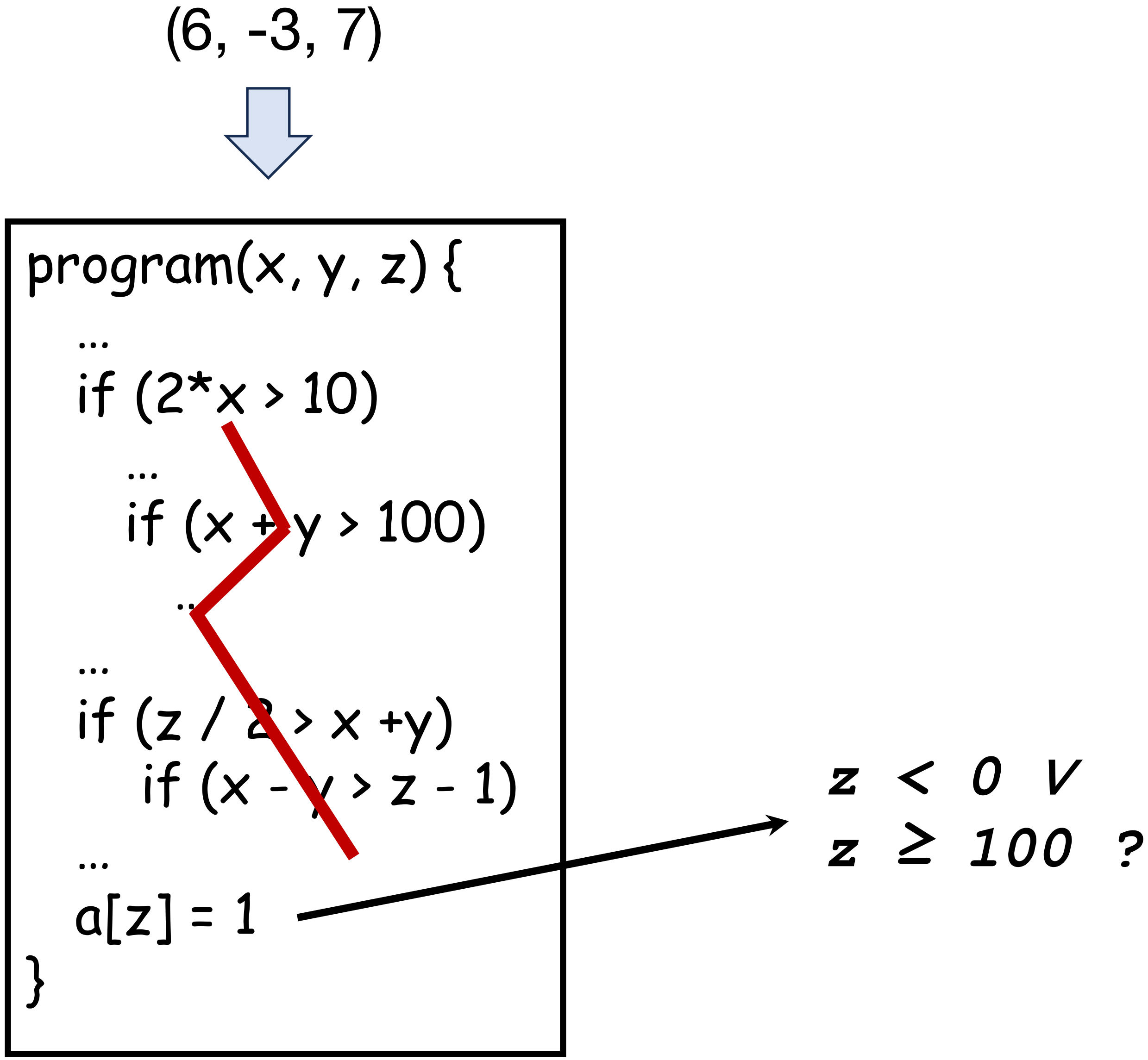
$x+y \le 100$

$z/2 < x+y$

$x-y > z-1$

$2x > 10 \wedge x+y \le 100 \wedge z/2 > x+y \wedge x-y > z-1$

Dynamic Symbolic Execution or Concolic Execution



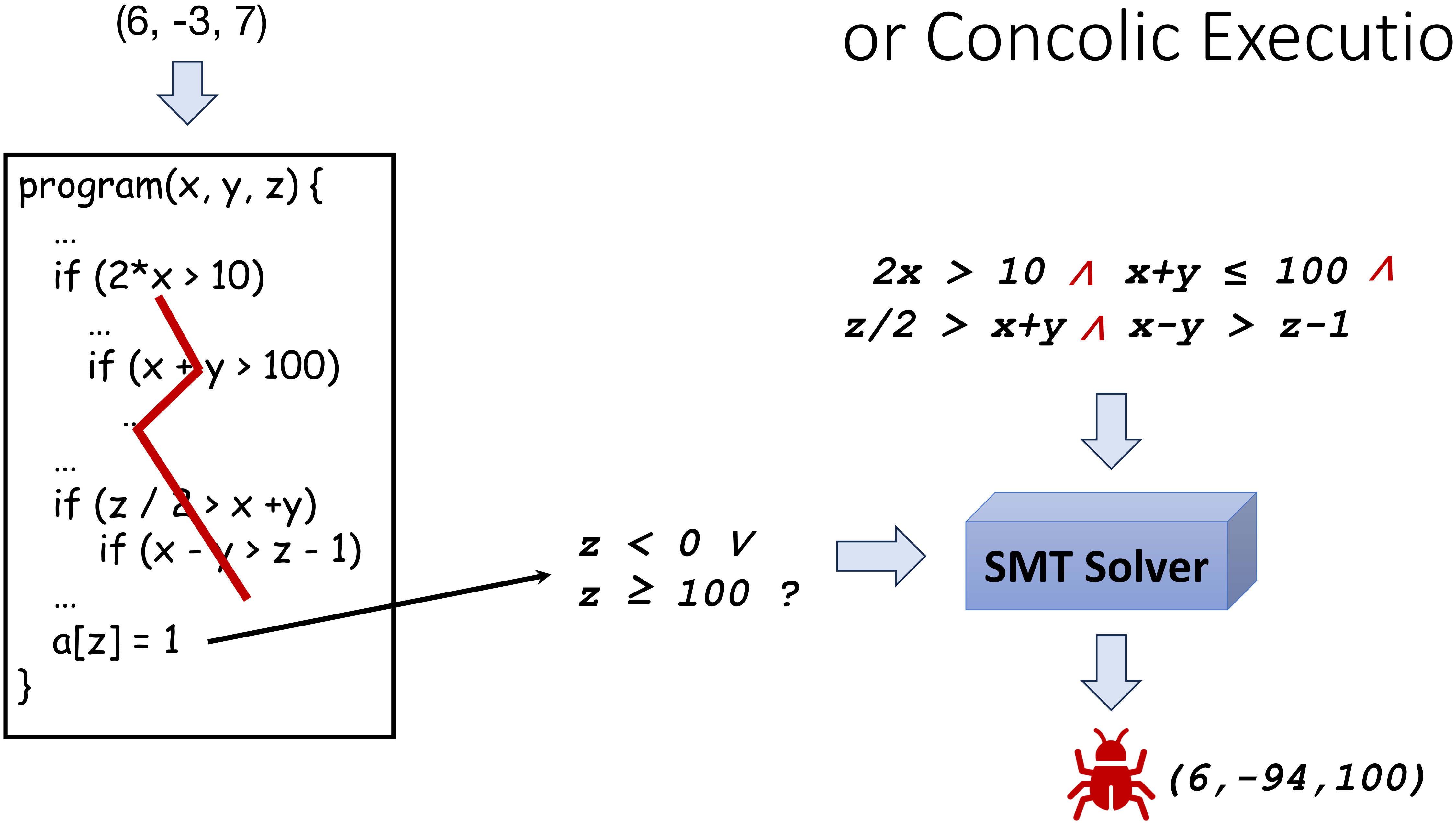
(6, -3, 7)

```
program(x, y, z) {
   …
   if (2*x > 10)
      …
      if (x + y > 100)
         …
   …
   if (z / 2 > x +y)
      if (x - y > z - 1)
   …
   a[z] = 1
}
```

$z < 0 \lor$
$z \geq 100$?

$2x > 10 \land x+y \leq 100 \land$
$z/2 > x+y \land x-y > z-1$

Dynamic Symbolic Execution
or Concolic Execution



(6, -3, 7)

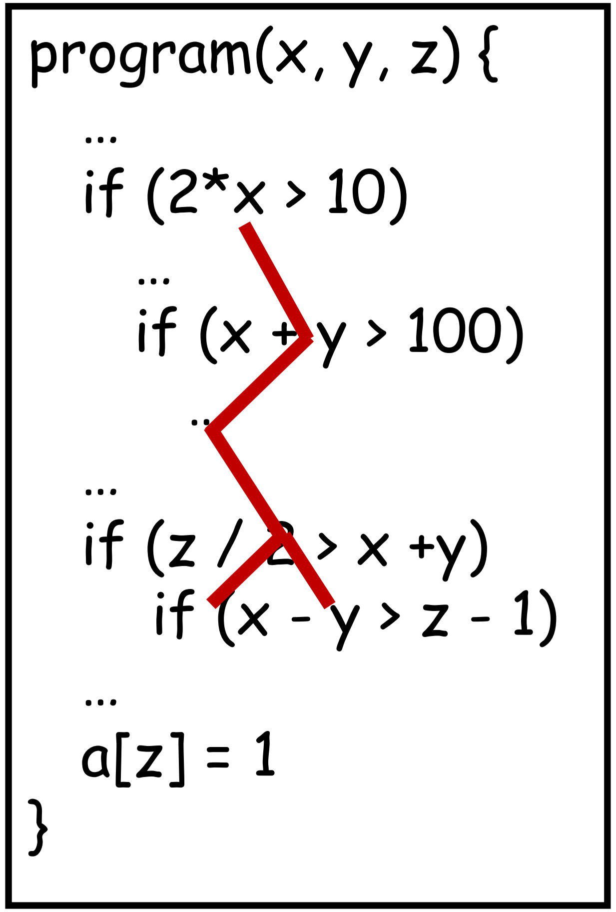
```
program(x, y, z) {
   ...
   if (2*x > 10)
      ...
      if (x + y > 100)
         ...
   ...
   if (z / 2 > x +y)
      if (x - y > z - 1)
   ...
   a[z] = 1
}
```

$2x > 10 \wedge x+y \leq 100 \wedge$
$z/2 > x+y \wedge x-y > z-1$

$z < 0 \vee$
$z \geq 100$?

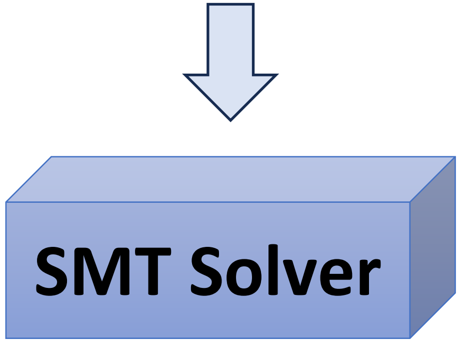
(6,-94,100)

Dynamic Symbolic Execution
or Concolic Execution



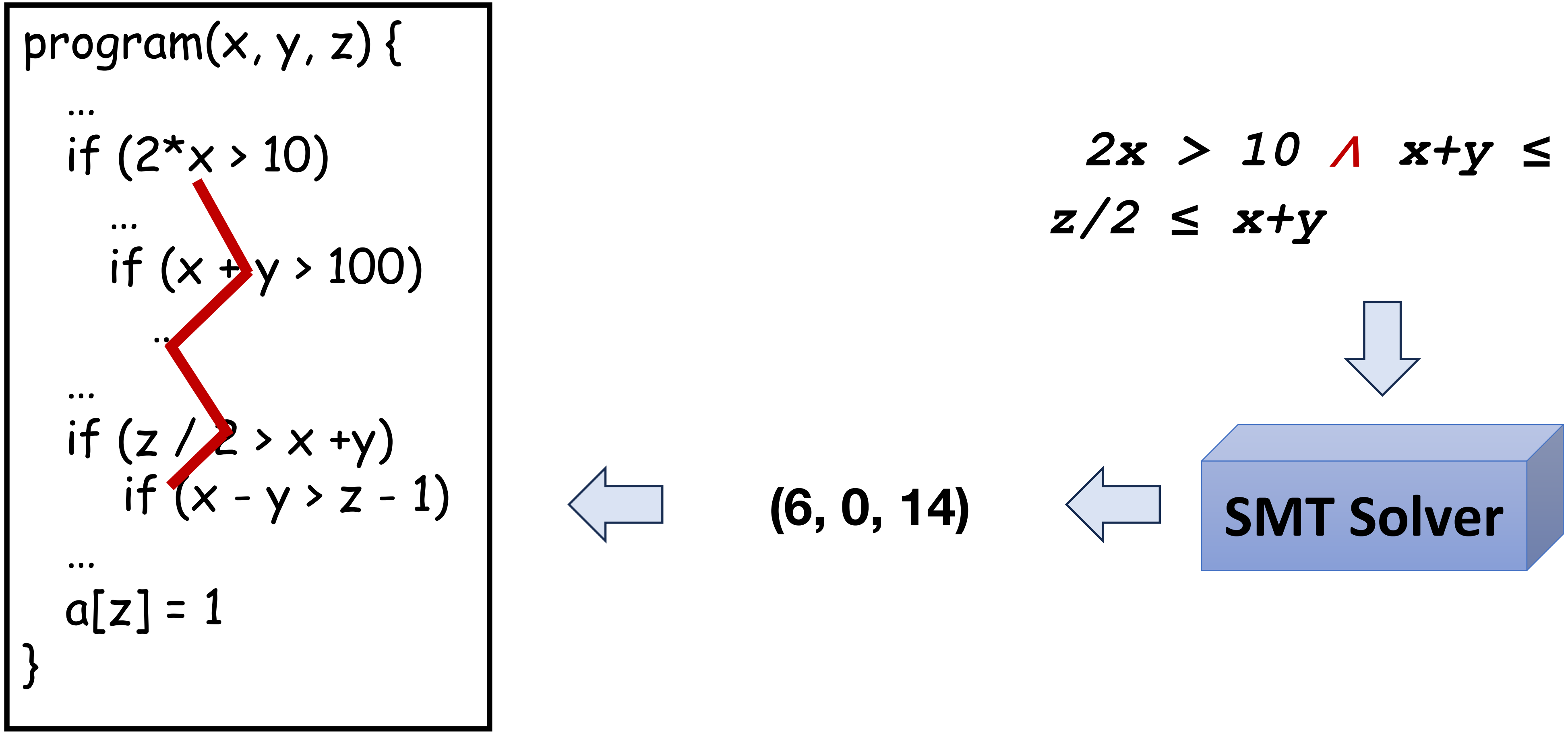
```
program(x, y, z) {
   ...
   if (2*x > 10)
      ...
      if (x + y > 100)
         ...
   ...
   if (z / 2 > x +y)
      if (x - y > z - 1)
   ...
   a[z] = 1
}
```

$2x > 10 \wedge x+y \leq 100 \wedge$
$z/2 \geq x+y$



SMT Solver

Dynamic Symbolic Execution or Concolic Execution



```
program(x, y, z) {
   …
   if (2*x > 10)
      …
      if (x + y > 100)
         …
   …
   if (z / 2 > x +y)
       if (x - y > z - 1)
   …
   a[z] = 1
}
```

$2x > 10 \land x+y \leq 100 \land z/2 \leq x+y$

SMT Solver

(6, 0, 14)

KLEE
FINDING BUGS WITH STYLE

Popular dynamic symbolic executor primarily developed and maintained at Imperial

Works at the LLVM level: C (full support), C++, Rust

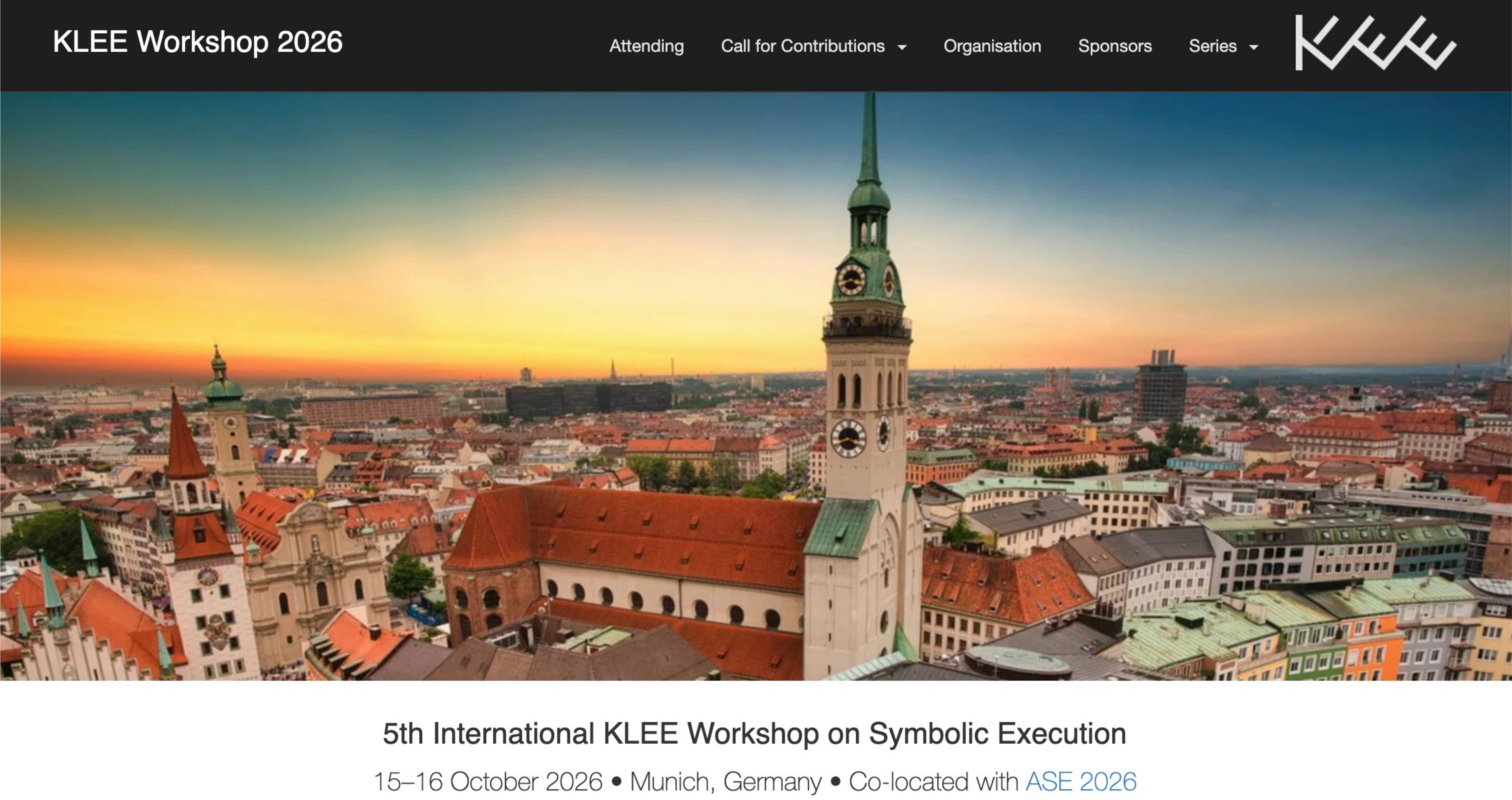
`https://klee-se.org/`
`https://github.com/klee/`

Widely used in academia

- ACM SIGOPS Hall of Fame Award and ACM CCS Test of Time Award
- Hundreds of projects building up on KLEE (`https://klee-se.org/Publications`)
- Over 5k citations to original KLEE paper
  - Many different communities including Software Engineering, Programming Languages, Verification, Security, Architecture, Systems

Growing impact in industry:

- **Baidu**: [KLEE 2018]
- **Fujitsu**: [PPoPP 2012], [CAV 2013], [ICST 2015], [IEEE Software 2017], [KLEE 2018]
- **Intel**: [WOOT 2015]
- **Google**: [2x KLEE 2021]
- **Hitachi**: [CPSNA 2014], [ISPA 2015], [EUC 2016], [KLEE 2021]
- **NASA Ames**: [NFM 2014]
- **Samsung**: 2 x [KLEE 2018], [KLEE 2024]
- **Trail of Bits:** [blog.trailofbits.com/]



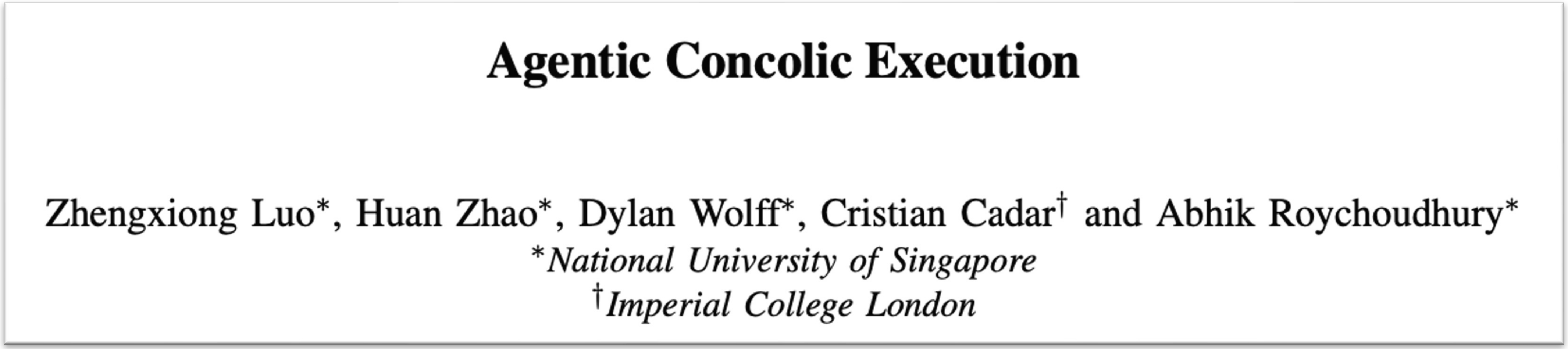
5th International KLEE Workshop on Symbolic Execution

15–16 October 2026 • Munich, Germany • Co-located with ASE 2026

Building Program Reasoning Tools

- Huge effort
  - E.g., KLEE adopted by industry and found bugs and vulnerabilities in hundreds of projects
  - But has taken us >15 years of development time with contributions from dozens of developers
- Important practical limitations
  - E.g., KLEE is limited to LLVM languages (mainly C and C++)
  - Has incomplete support for environmental interactions
  - Has no support for symbolic floating point
  - etc.
- What about other programming languages, DSLs and system types?

Can AI help democratise program reasoning?



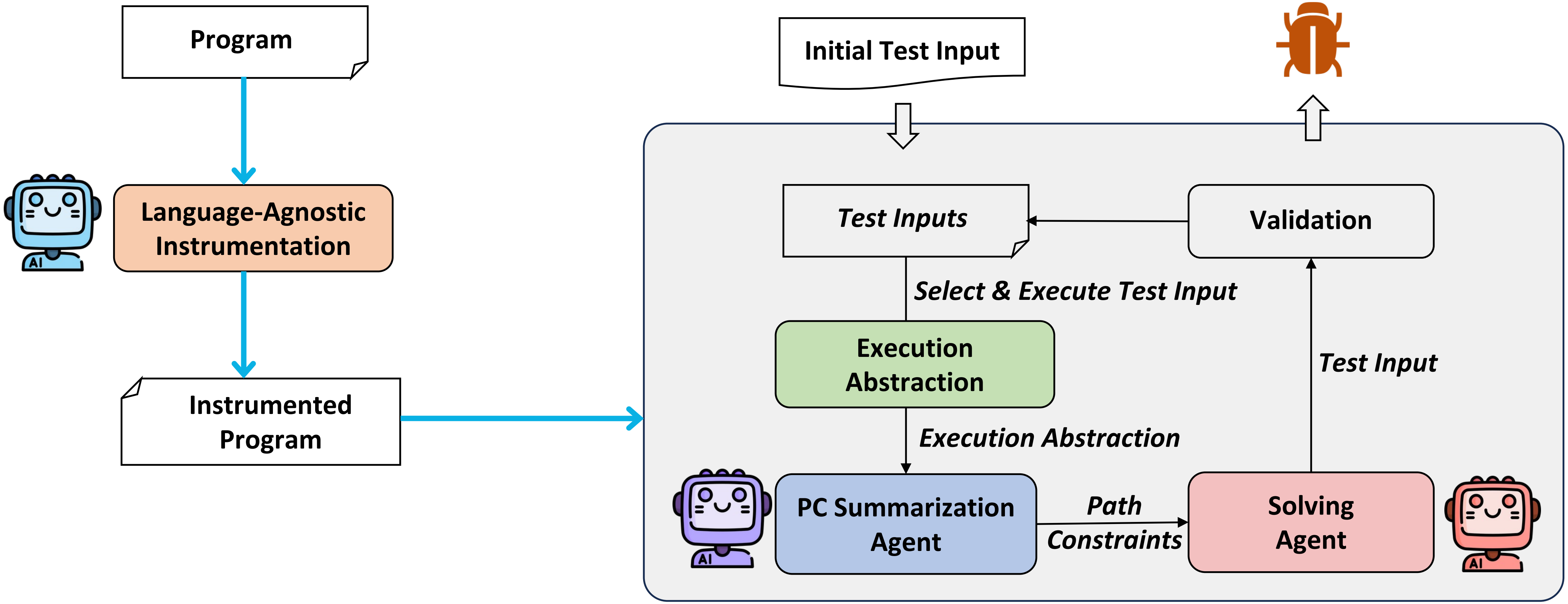
Agentic Concolic Execution

Zhengxiong Luo*, Huan Zhao*, Dylan Wolff*, Cristian Cadar† and Abhik Roychoudhury*
*National University of Singapore
†Imperial College London

- Support for arbitrary languages
  - including support for polyglot code
- Support for a variety of system types
  - e.g., floating-point intensive code, binary inputs, etc.
- Support for complex environmental interactions
  - e.g., memory allocation failures, network communication, etc.

ConcoLLMic

ConcoLLMic


Program

Language-Agnostic Instrumentation

Instrumented Program

Initial Test Input

Test Inputs

Validation

Select & Execute Test Input

Execution Abstraction

Test Input

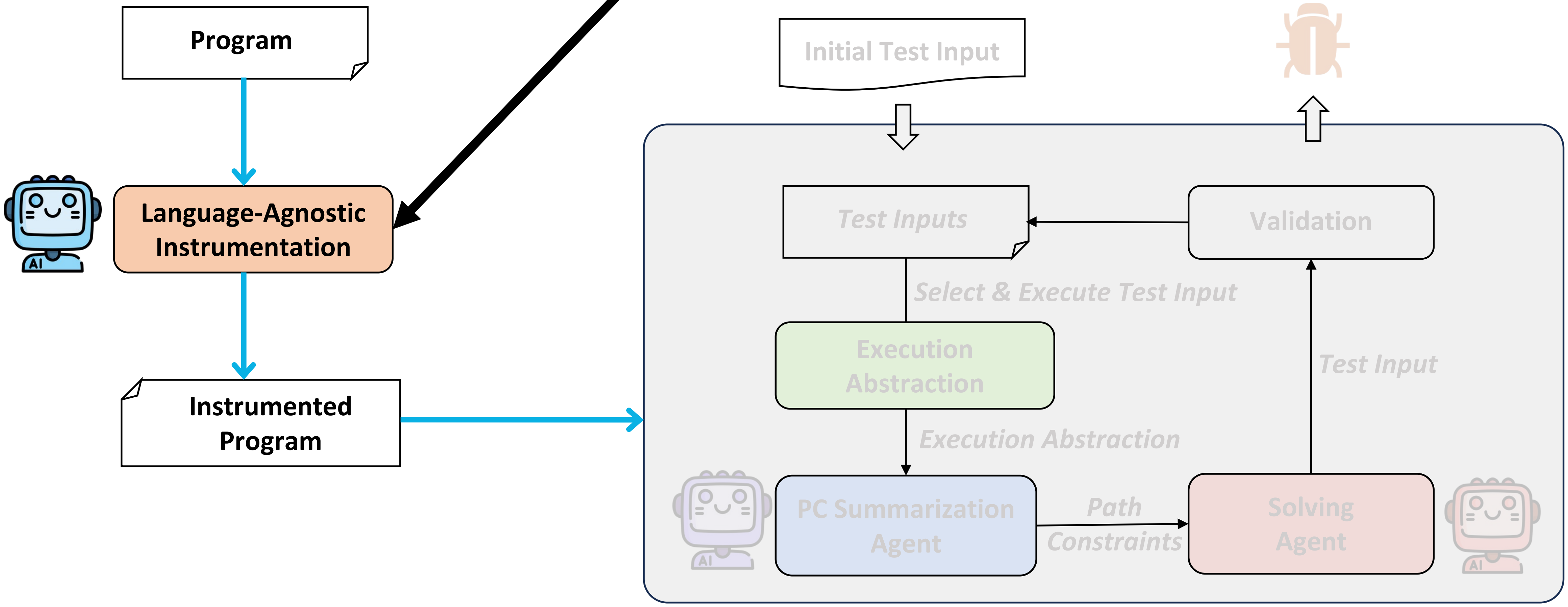
Execution Abstraction

PC Summarization Agent

Path Constraints

Solving Agent

- Adds logging statements around each code block
- Enables recording code coverage
- Language agnostic! Leverages LLM multi-language capabilities



Program

Language-Agnostic Instrumentation

Instrumented Program

Initial Test Input

Test Inputs

Validation

Select & Execute Test Input

Execution Abstraction

Test Input

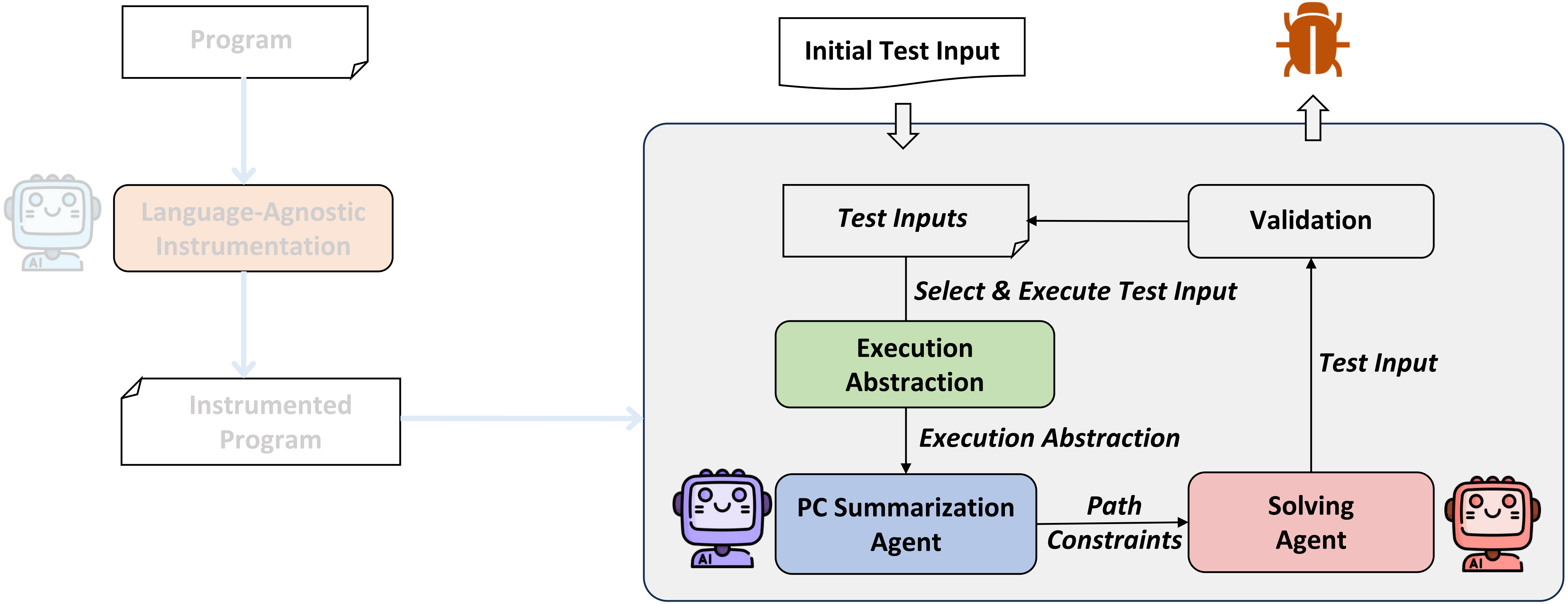
Execution Abstraction

PC Summarization Agent

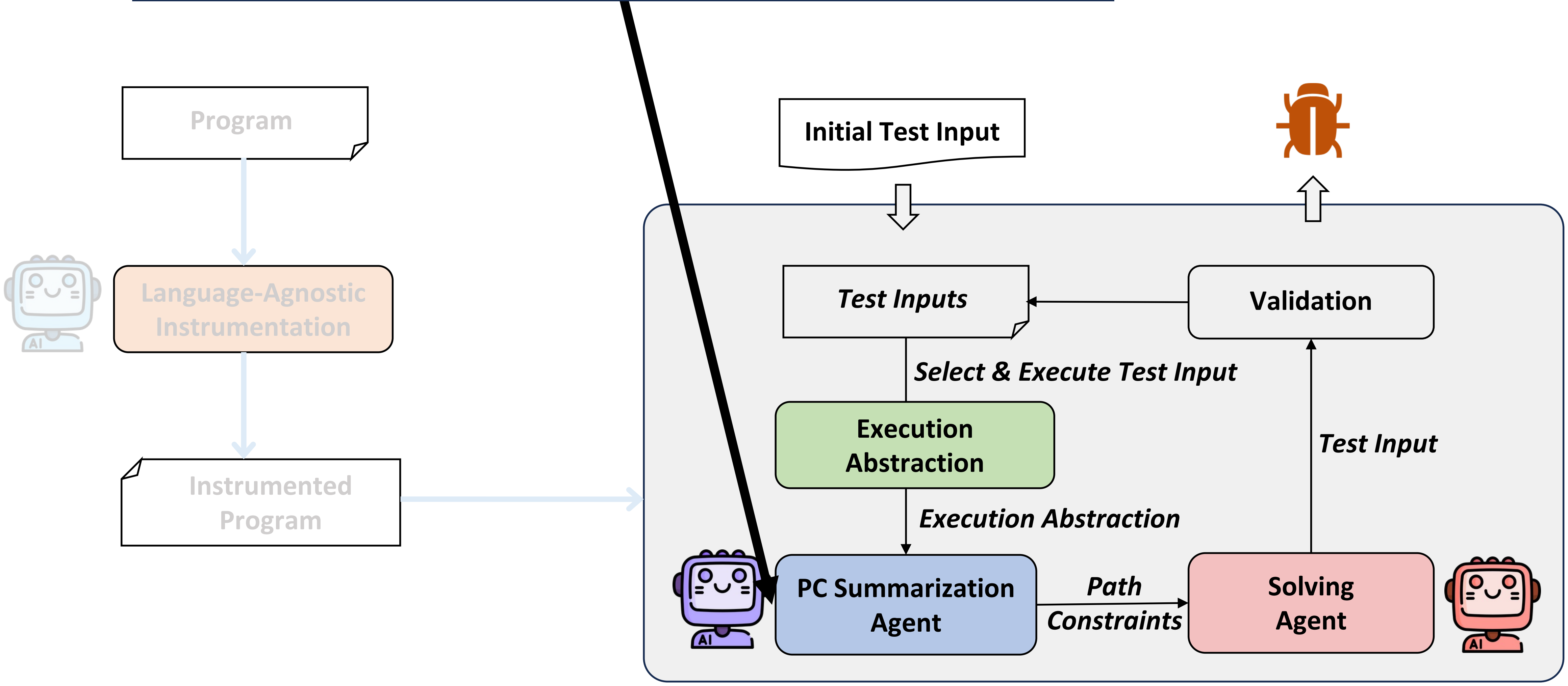
Path Constraints

Solving Agent

ConcoLLMic

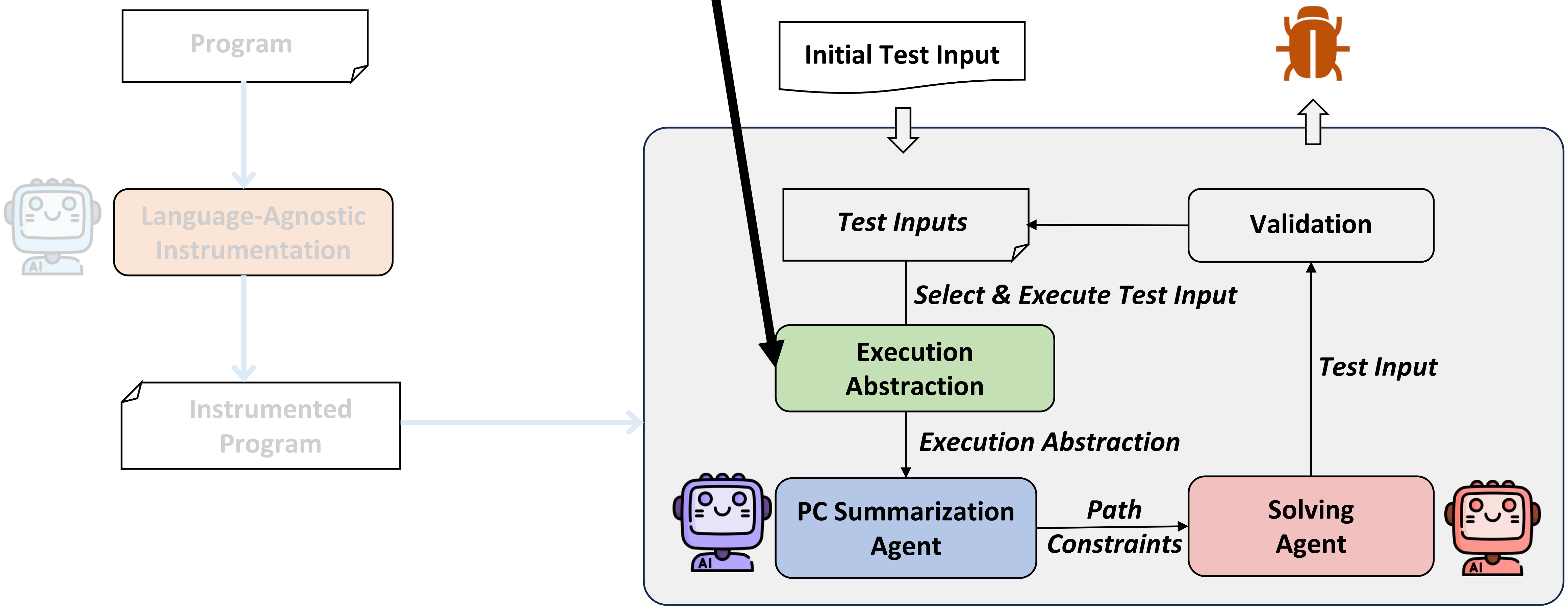

Program
Language-Agnostic Instrumentation
Instrumented Program
Initial Test Input
Test Inputs
Validation
Select & Execute Test Input
Execution Abstraction
Test Input
Execution Abstraction
PC Summarization Agent
Path Constraints
Solving Agent
AI
AI
AI

- Select a branch to switch
- Summarise the new path constraints PC’ in a choice of: SMT constraints, natural language, or code

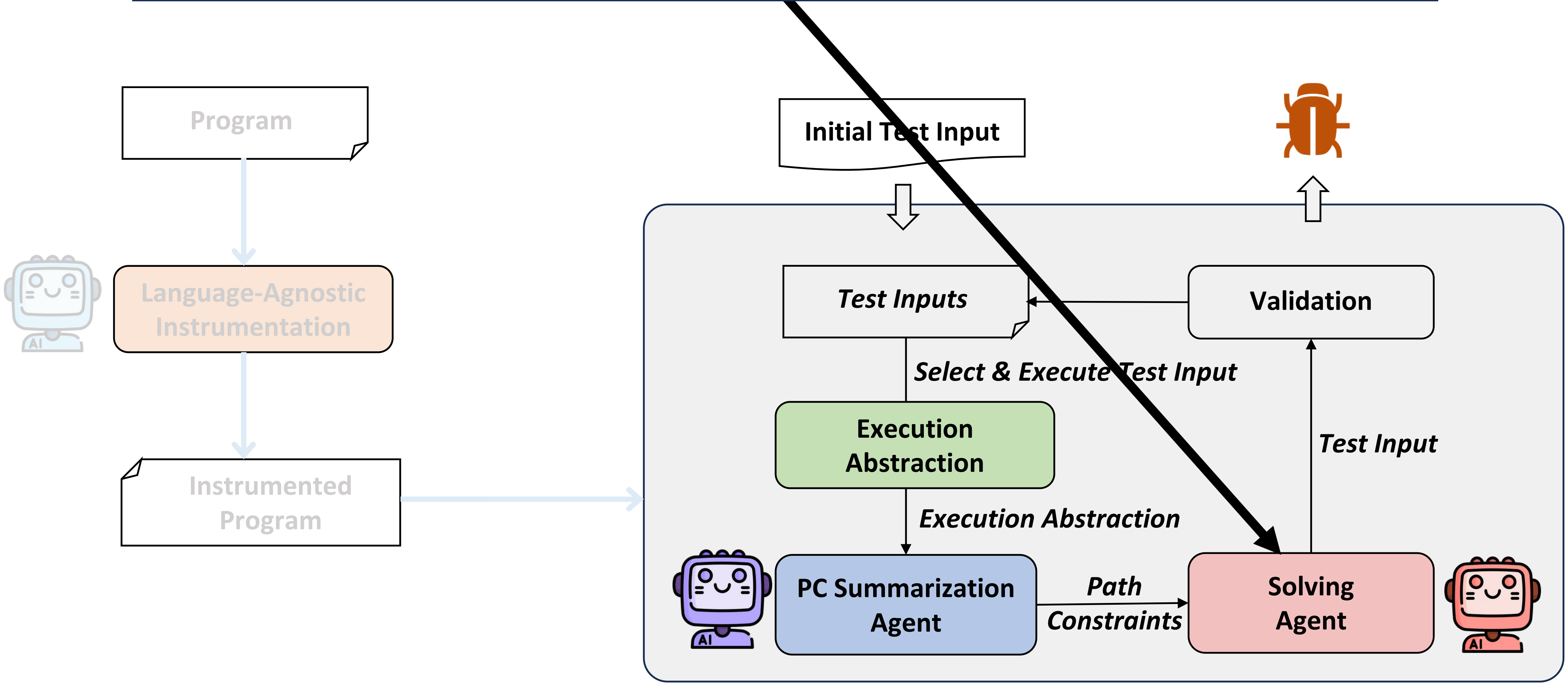


Minimise amount of info passed to PC Summarization Agent

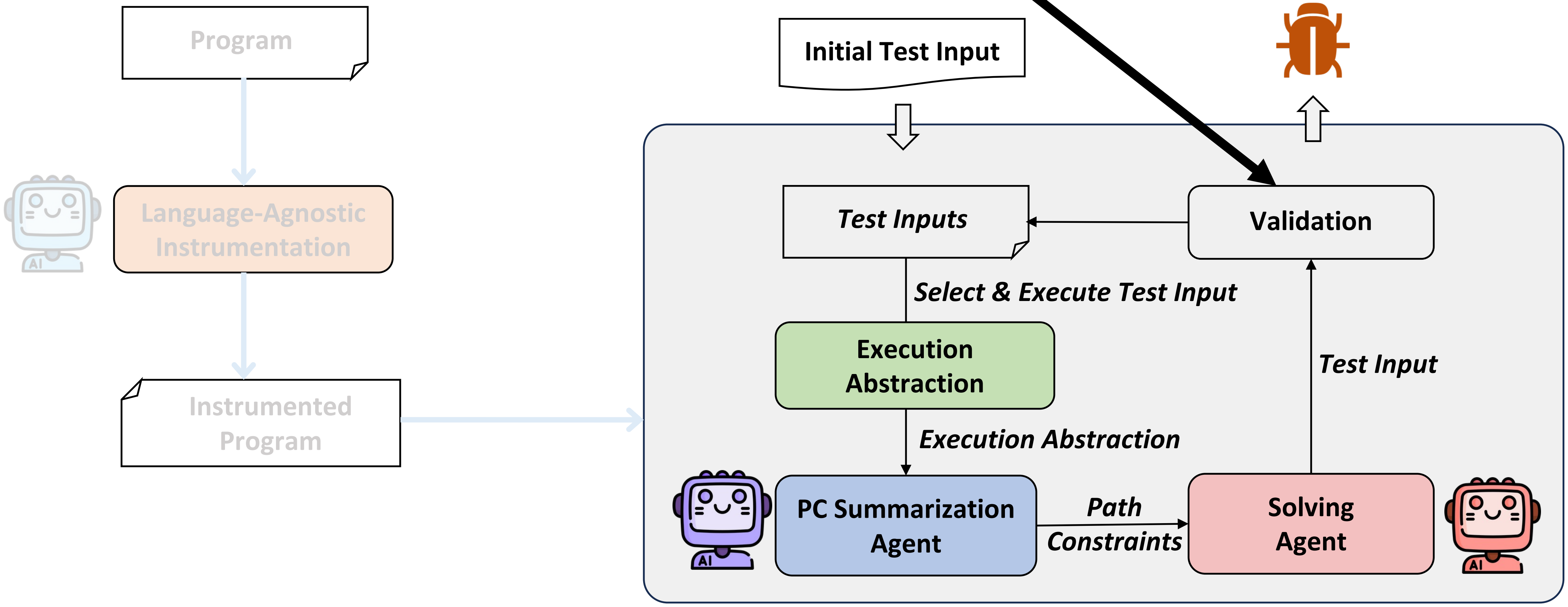
1. Function call chain info
2. Provide covered code and a coverage summary for uncovered code



- Solve the constraints received from PC Summarisation Agent
- Can reason in natural language, via code execution or using an SMT solver



Check if the solution is correct and/or leads to new code



Actions Available to Agents

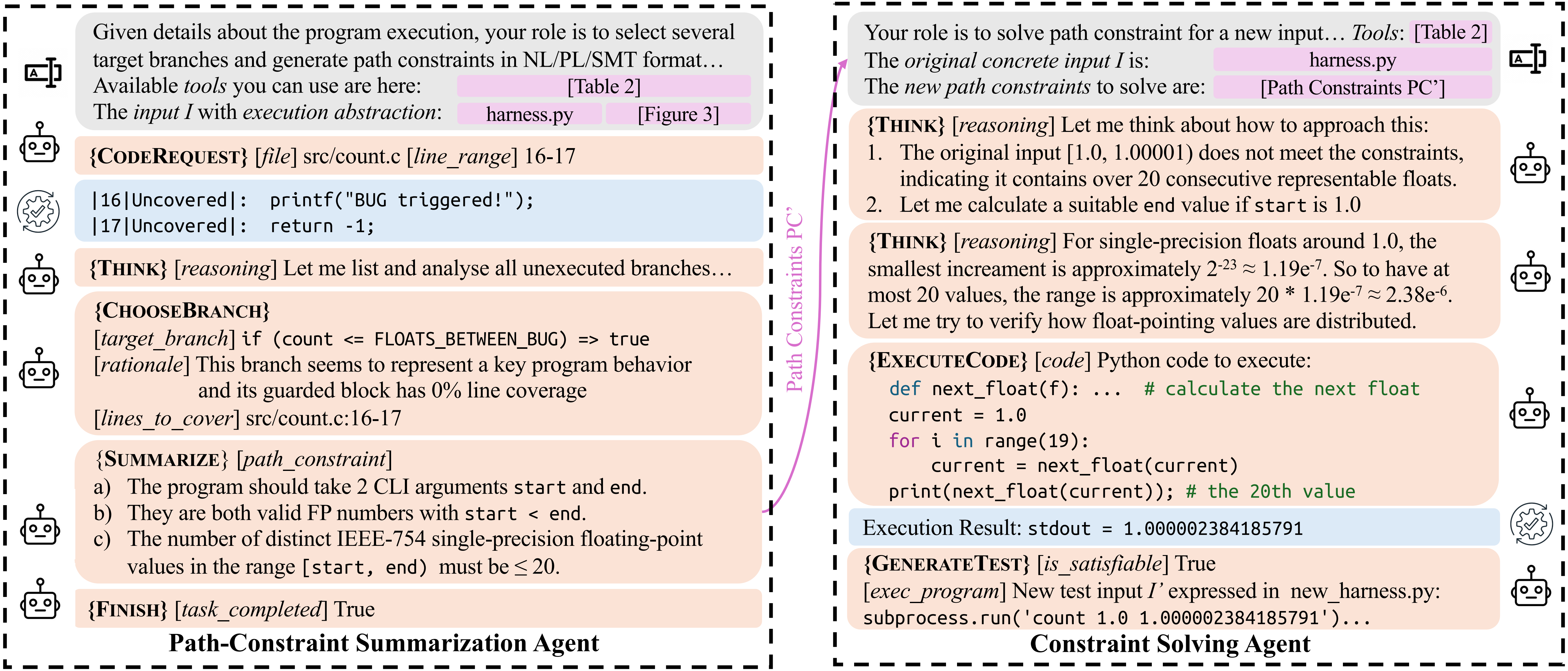
PC Summarization Agent

- **Think**: Record reasoning and planning
- **CodeRequest**: View source code and coverage status
- **ChooseBranch**: Select a branch to flip
- **Summarise**: Generate PC to reach the target branch (in NL/PL/SMT)

Constraint Solving Agent

- **Think**: Record reasoning and planning
- **ExecuteCode**: Create & execute Python code and collect its output
- **QuerySMT**: Solve formula with SMT solver
- **GenerateTest**: Output an input for the given constraints, if satisfiable

ConcoLLMic in Action



Given details about the program execution, your role is to select several target branches and generate path constraints in NL/PL/SMT format…
Available *tools* you can use here: [Table 2]
The *input I* with *execution abstraction*: harness.py [Figure 3]

{CodeRequest} [*file*] src/count.c [*line_range*] 16-17

```
|16|Uncovered|:  printf("BUG triggered!");
|17|Uncovered|:  return -1;
```

{Think} [*reasoning*] Let me list and analyse all unexecuted branches…

{ChooseBranch}
[*target_branch*] `if (count <= FLOATS_BETWEEN_BUG) => true`
[*rationale*] This branch seems to represent a key program behavior and its guarded block has 0% line coverage
[*lines_to_cover*] src/count.c:16-17

{Summarize} [*path_constraint*]
a) The program should take 2 CLI arguments `start` and `end`.
b) They are both valid FP numbers with `start < end`.
c) The number of distinct IEEE-754 single-precision floating-point values in the range `[start, end)` must be ≤ 20.

{Finish} [*task_completed*] True

Path-Constraint Summarization Agent

Your role is to solve path constraint for a new input… *Tools*: [Table 2]
The *original concrete input I* is: harness.py
The *new path constraints* to solve are: [Path Constraints PC’]

{Think} [*reasoning*] Let me think about how to approach this:
1. The original input [1.0, 1.00001) does not meet the constraints, indicating it contains over 20 consecutive representable floats.
2. Let me calculate a suitable `end` value if `start` is 1.0

{Think} [*reasoning*] For single-precision floats around 1.0, the smallest increament is approximately $2^{-23} \approx 1.19e^{-7}$. So to have at most 20 values, the range is approximately $20 * 1.19e^{-7} \approx 2.38e^{-6}$. Let me try to verify how float-pointing values are distributed.

{ExecuteCode} [*code*] Python code to execute:

```
def next_float(f): ...  # calculate the next float
current = 1.0
for i in range(19):
    current = next_float(current)
print(next_float(current)); # the 20th value
```

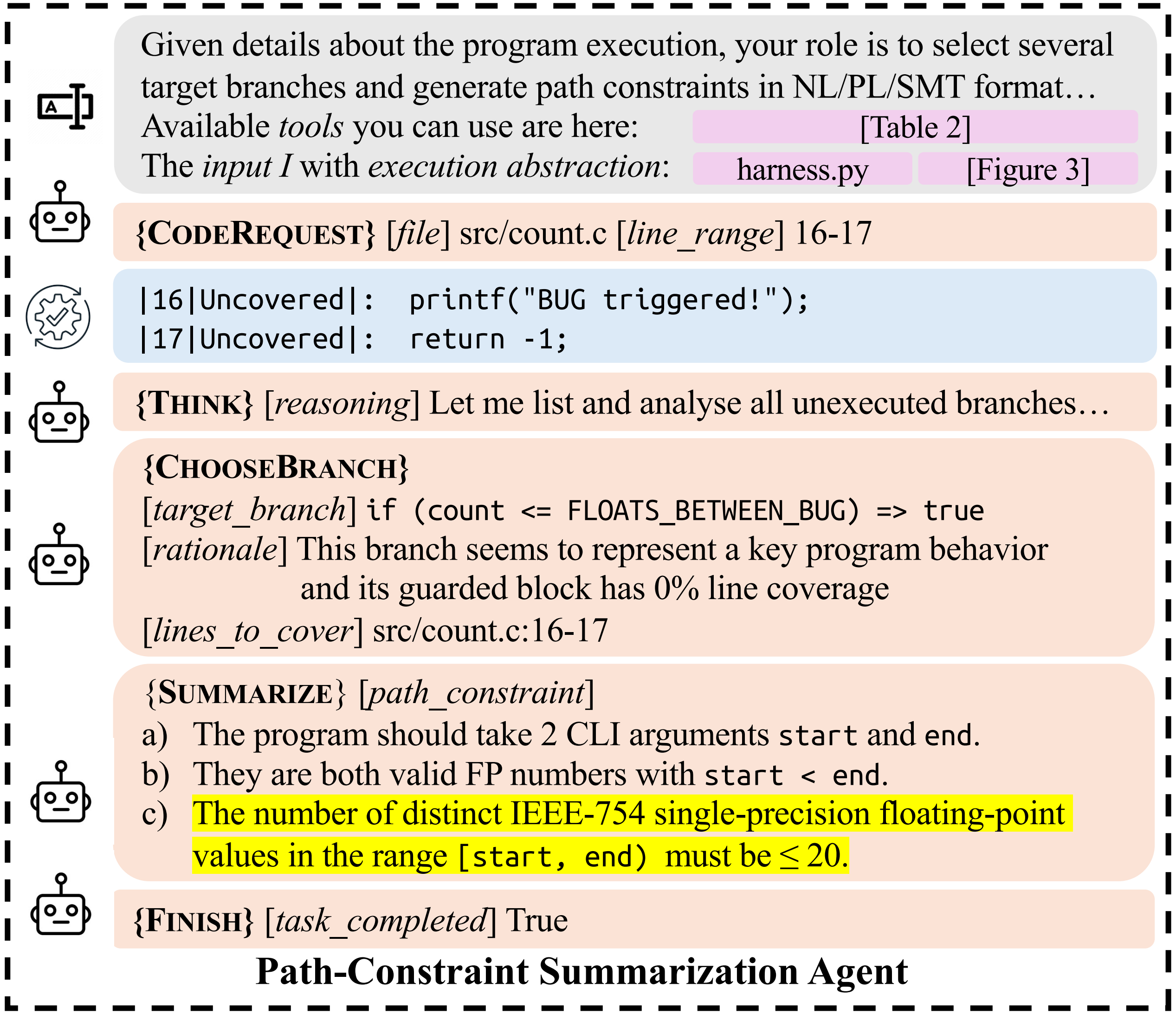
Execution Result: `stdout = 1.000002384185791`

{GenerateTest} [*is_satisfiable*] True
[*exec_program*] New test input *I’* expressed in new_harness.py:
`subprocess.run('count 1.0 1.000002384185791')...`

Constraint Solving Agent

ConcoLLMic with Claude 3.7 Sonnet

ConcoLLMic in Action



Given details about the program execution, your role is to select several target branches and generate path constraints in NL/PL/SMT format…
Available *tools* you can use are here: [Table 2]
The *input I* with *execution abstraction*: harness.py [Figure 3]

{CodeRequest} [*file*] src/count.c [*line_range*] 16-17

```
|16|Uncovered|:  printf("BUG triggered!");
|17|Uncovered|:  return -1;
```

{Think} [*reasoning*] Let me list and analyse all unexecuted branches…

{ChooseBranch}
[*target_branch*] `if (count <= FLOATS_BETWEEN_BUG) => true`
[*rationale*] This branch seems to represent a key program behavior and its guarded block has 0% line coverage
[*lines_to_cover*] src/count.c:16-17

{Summarize} [*path_constraint*]

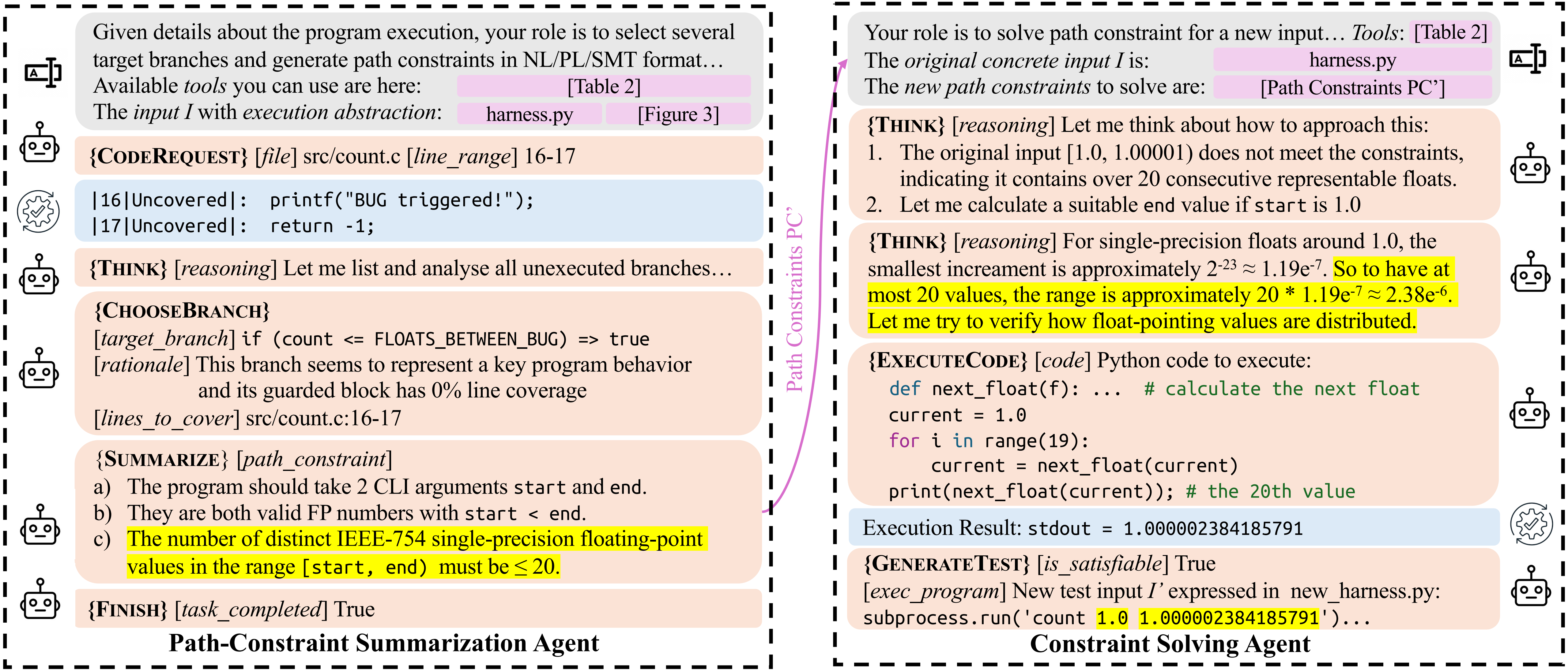
a) The program should take 2 CLI arguments `start` and `end`.
b) They are both valid FP numbers with `start < end`.
c) The number of distinct IEEE-754 single-precision floating-point values in the range `[start, end)` must be ≤ 20.

{Finish} [*task_completed*] True

Path-Constraint Summarization Agent

ConcoLLMic with Claude 3.7 Sonnet

ConcoLLMic in Action



Given details about the program execution, your role is to select several target branches and generate path constraints in NL/PL/SMT format…
Available *tools* you can use are here: [Table 2]
The *input I* with *execution abstraction*: harness.py [Figure 3]

{CodeRequest} [*file*] src/count.c [*line_range*] 16-17

```
|16|Uncovered|:  printf("BUG triggered!");
|17|Uncovered|:  return -1;
```

{Think} [*reasoning*] Let me list and analyse all unexecuted branches…

{ChooseBranch}
[*target_branch*] `if (count <= FLOATS_BETWEEN_BUG) => true`
[*rationale*] This branch seems to represent a key program behavior and its guarded block has 0% line coverage
[*lines_to_cover*] src/count.c:16-17

{Summarize} [*path_constraint*]
a) The program should take 2 CLI arguments `start` and `end`.
b) They are both valid FP numbers with `start < end`.
c) The number of distinct IEEE-754 single-precision floating-point values in the range `[start, end)` must be ≤ 20.

{Finish} [*task_completed*] True

Path-Constraint Summarization Agent

Your role is to solve path constraint for a new input… *Tools*: [Table 2]
The *original concrete input I* is: harness.py
The *new path constraints* to solve are: [Path Constraints PC']

{Think} [*reasoning*] Let me think about how to approach this:
1. The original input [1.0, 1.00001) does not meet the constraints, indicating it contains over 20 consecutive representable floats.
2. Let me calculate a suitable `end` value if `start` is 1.0

{Think} [*reasoning*] For single-precision floats around 1.0, the smallest increament is approximately $2^{-23} \approx 1.19e^{-7}$. So to have at most 20 values, the range is approximately $20 * 1.19e^{-7} \approx 2.38e^{-6}$. Let me try to verify how float-pointing values are distributed.

{ExecuteCode} [*code*] Python code to execute:

```
def next_float(f): ...   # calculate the next float
current = 1.0
for i in range(19):
    current = next_float(current)
print(next_float(current)); # the 20th value
```

Execution Result: `stdout = 1.000002384185791`

{GenerateTest} [*is_satisfiable*] True
[*exec_program*] New test input *I'* expressed in new_harness.py:
`subprocess.run('count 1.0 1.000002384185791')...`

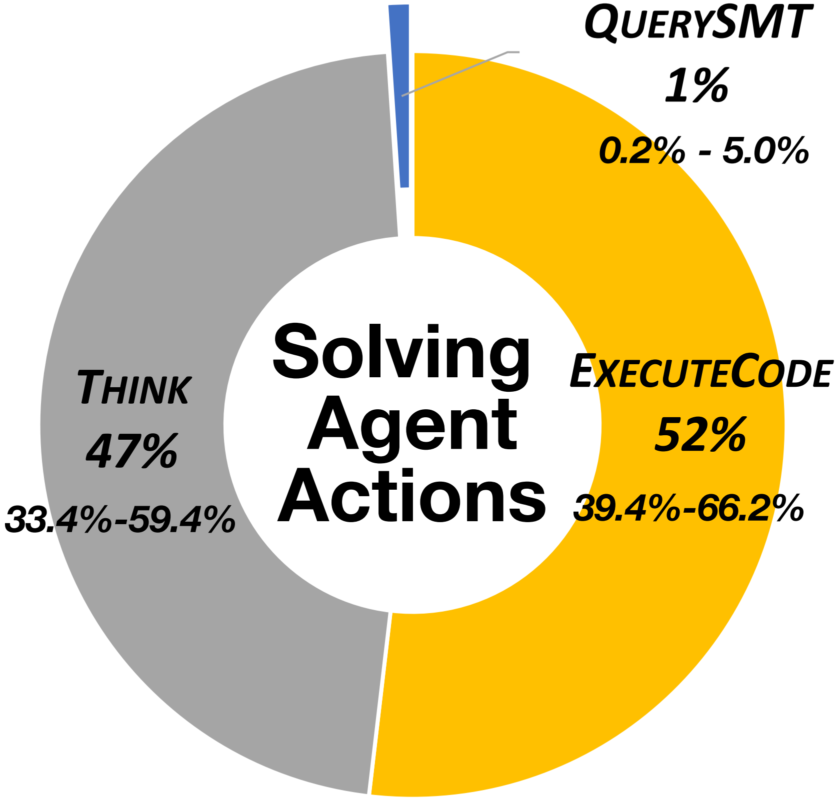
Constraint Solving Agent

ConcoLLMic with Claude 3.7 Sonnet

Reasoning Beyond Formal Solvers

- In the Solving Agent, QUERYSMT is used *surprisingly rarely (0.2% - 5%)*

- *Think – Execute – Validate* pattern:
    - THINK for analysis and hypothesis
    - EXECUTECODE to compute or validate solution
    - Back to THINK to refine reasoning iteratively



Environment Reasoning

Case Study: `malloc` failure scenario

1. Create a custom `malloc` wrapper
2. Compile into a shared library `malloc.so`
3. Employ `LD_PRELOAD` inject into program's execution environment
4. Successfully covers the failure scenario ✓

```
void * bc_malloc (size_t size) {
  void *ptr = (void *) malloc (size);
  if (ptr == NULL) {
    // Unexecuted line 723 removed. Line cov: 0/1
  }
  return ptr;
}
```

```
# Create a malloc wrapper that will fail after a
     certain number of calls
malloc_wrapper_c = """
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>

static void* (*real_malloc)(size_t) = NULL;
static int malloc_count = 0;
static int fail_after = 30;  // Fail after this many
      calls

void* malloc(size_t size) {
// Initialize real_malloc on first call
if (real_malloc == NULL) {
  real_malloc = (void* (*)(size_t))dlsym(RTLD_NEXT,
       "malloc");
  if (real_malloc == NULL) {
    fprintf(stderr, "Error finding real malloc\\n");
    return NULL;
  }
}

malloc_count++;

// Fail after a certain number of calls
if (malloc_count >= fail_after) {
  return NULL;  // Simulate allocation failure
}

return real_malloc(size);
}
"""
```

Evaluation Setup

8 medium-sized
C/C++ applications

4 medium-sized
polyglot applications

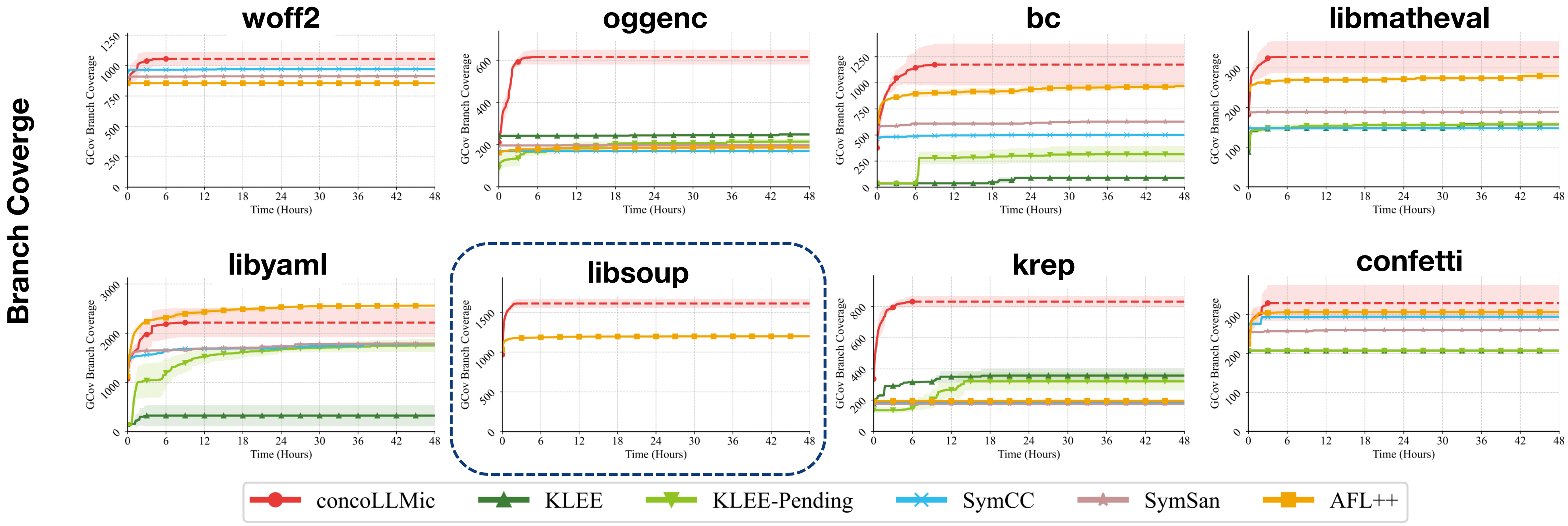
26 hand-crafted
FP programs

Three variants:
Mainline, Pending, Float

SymCC **SymSan**

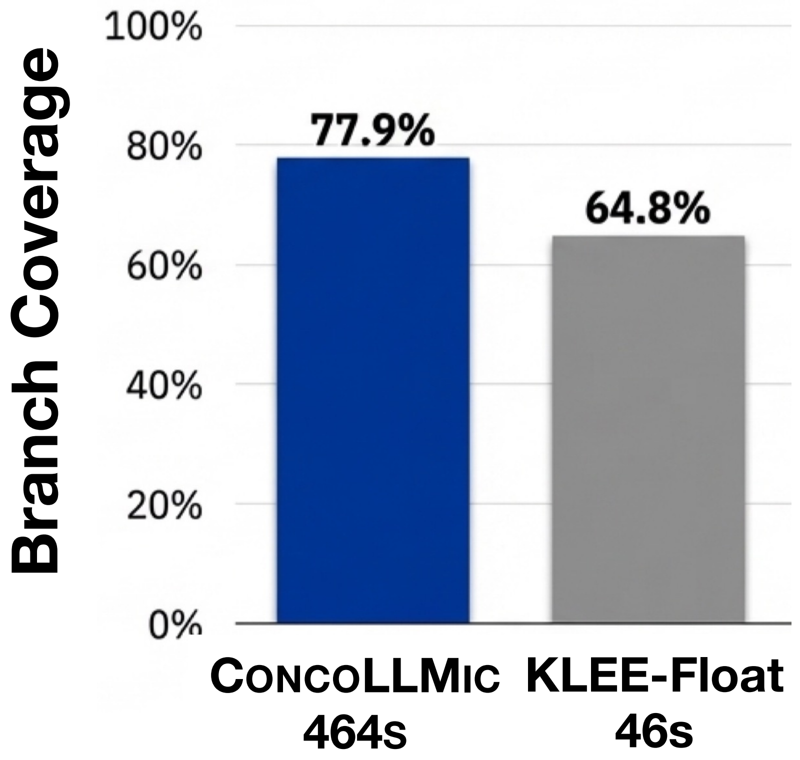
Code Coverage: C/C++ (48h)

- ConcoLLMic significantly outperforms leading DSEs & Fuzzers
  - +135% vs KLEE-Pending, +233% vs KLEE, +130% vs SymCC, +115% vs SymSan, +81% vs AFL++
- Handles network applications out-of-the-box, while other DSEs lack support



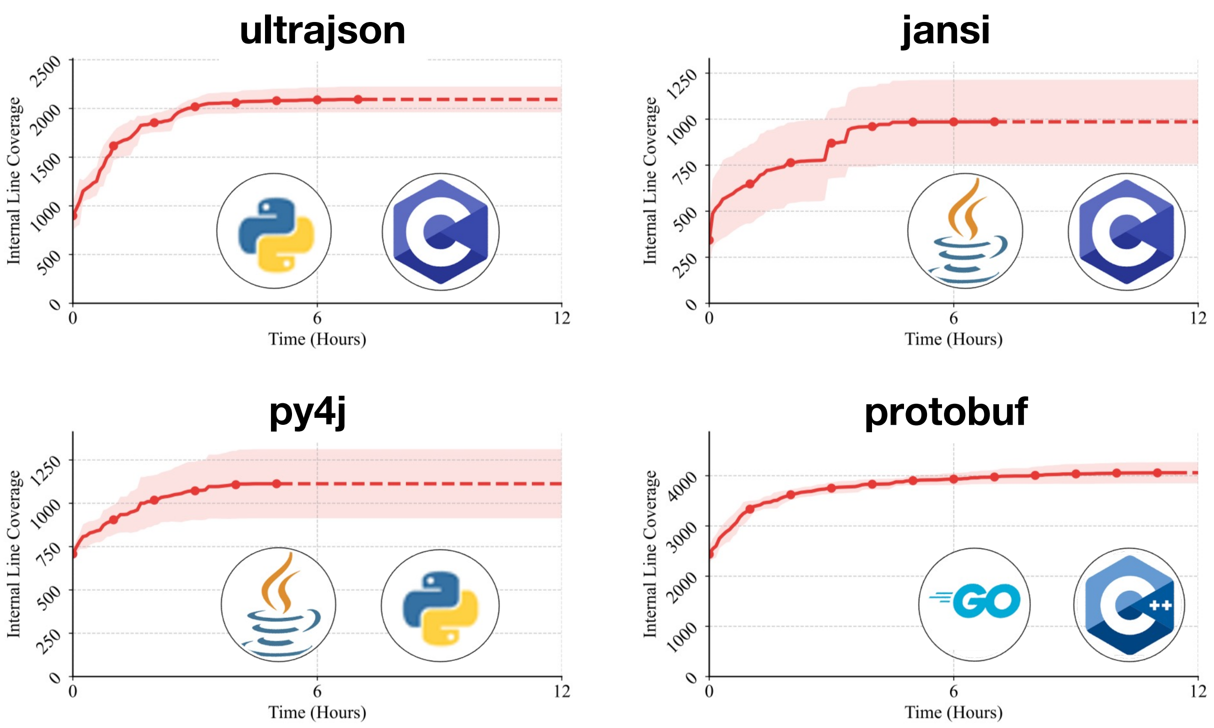
Code Coverage: Floating Point (FP-Bench)

- Floating-point constraints are notoriously difficult for DSE solvers
- KLEE-Float
  - A specialized, hand-crafted engine for floating-point
- CONCOLLMIC
  - 13% more coverage (*vs* KLEE-Float) without additional manual effort



Code Coverage: Polyglot Code (12h)

- Instrumentation & reasoning are universal across languages
- Successfully tested polyglot systems
  - Growing coverage + bugs



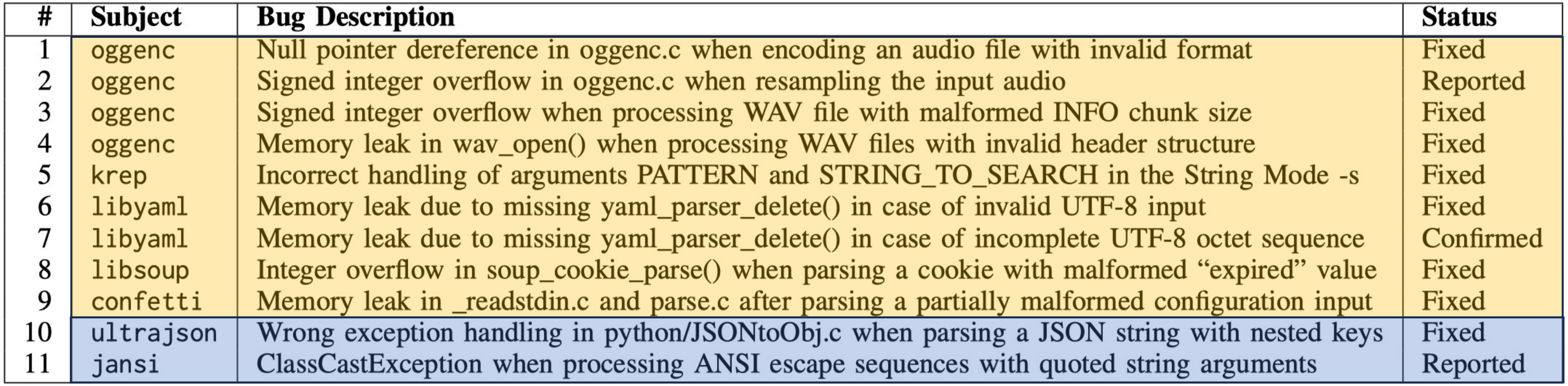
Line coverage / time

Bug Detection

- 11 previously-unknown bugs, 8 already fixed
- 6 of them missed by all other baseline tools
- Only KLEE-Pending found a bug not found by ConcoLLMic

C/C++ Subjects

Polyglot Subjects



#	Subject	Bug Description	Status
1	`oggenc`	Null pointer dereference in oggenc.c when encoding an audio file with invalid format	Fixed
2	`oggenc`	Signed integer overflow in oggenc.c when resampling the input audio	Reported
3	`oggenc`	Signed integer overflow when processing WAV file with malformed INFO chunk size	Fixed
4	`oggenc`	Memory leak in wav_open() when processing WAV files with invalid header structure	Fixed
5	`krep`	Incorrect handling of arguments PATTERN and STRING_TO_SEARCH in the String Mode -s	Fixed
6	`libyaml`	Memory leak due to missing yaml_parser_delete() in case of invalid UTF-8 input	Fixed
7	`libyaml`	Memory leak due to missing yaml_parser_delete() in case of incomplete UTF-8 octet sequence	Confirmed
8	`libsoup`	Integer overflow in soup_cookie_parse() when parsing a cookie with malformed "expired" value	Fixed
9	`confetti`	Memory leak in _readstdin.c and parse.c after parsing a partially malformed configuration input	Fixed
10	`ultrajson`	Wrong exception handling in python/JSONtoObj.c when parsing a JSON string with nested keys	Fixed
11	`jansi`	ClassCastException when processing ANSI escape sequences with quoted string arguments	Reported

We need support for the full diversity of PLs and system types!

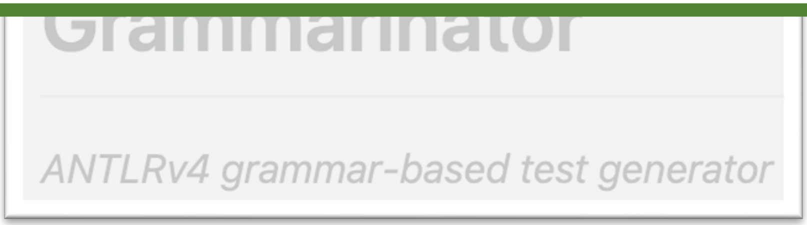
Clang Static Analyzer

AFL++

We need fast incremental techniques!

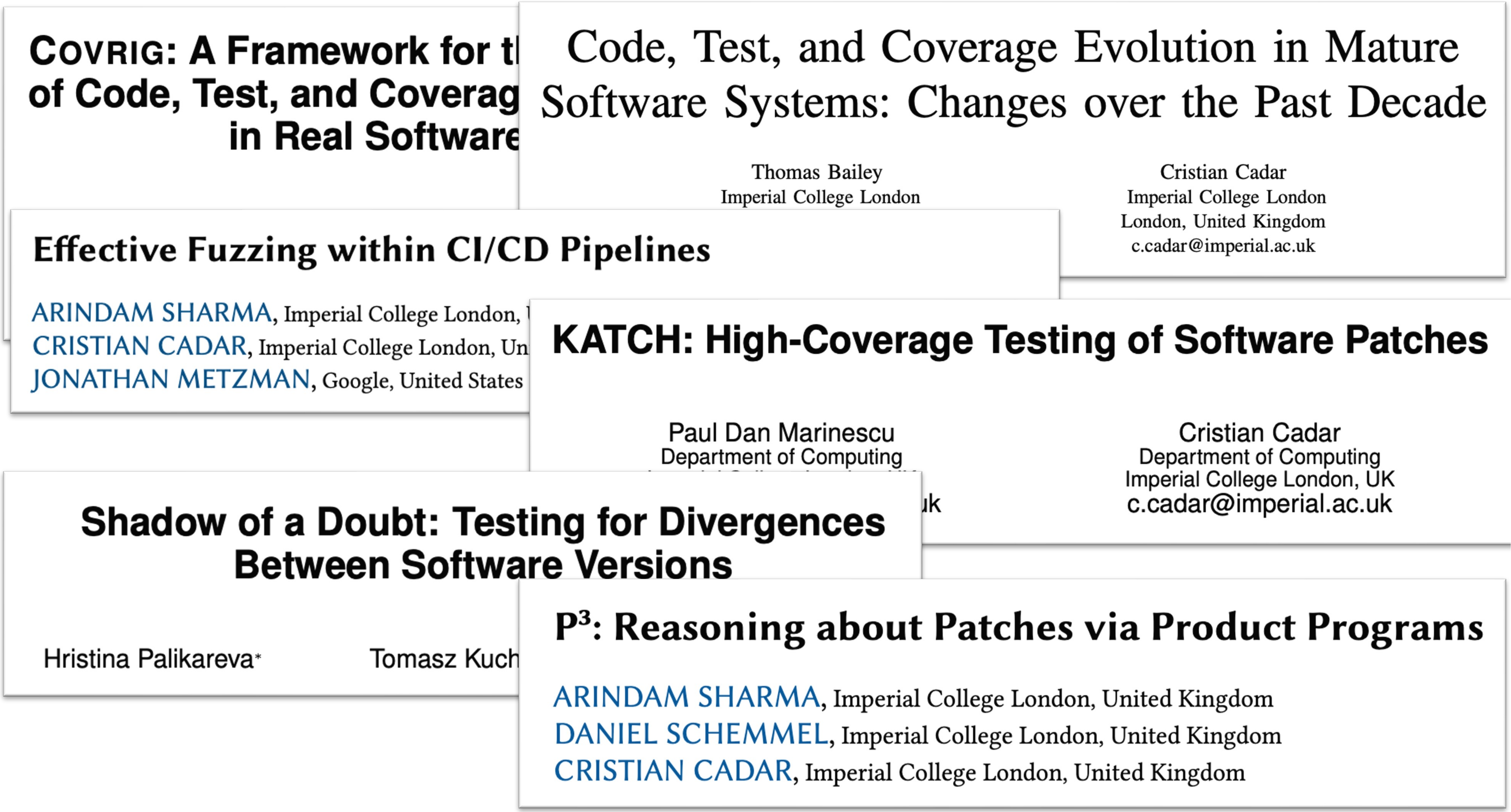
EVOSUITE

We need to reason about semantics!



Grammarinator

ANTLRv4 grammar-based test generator



COVRIG: A Framework for t
of Code, Test, and Coverag
in Real Software

Code, Test, and Coverage Evolution in Mature Software Systems: Changes over the Past Decade

Thomas Bailey
Imperial College London

Cristian Cadar
Imperial College London
London, United Kingdom
c.cadar@imperial.ac.uk

Effective Fuzzing within CI/CD Pipelines

ARINDAM SHARMA, Imperial College London,
CRISTIAN CADAR, Imperial College London, Un
JONATHAN METZMAN, Google, United States

KATCH: High-Coverage Testing of Software Patches

Paul Dan Marinescu
Department of Computing
uk

Cristian Cadar
Department of Computing
Imperial College London, UK
c.cadar@imperial.ac.uk

Shadow of a Doubt: Testing for Divergences Between Software Versions

Hristina Palikareva*

Tomasz Kuch

P^3: Reasoning about Patches via Product Programs

ARINDAM SHARMA, Imperial College London, United Kingdom
DANIEL SCHEMMEL, Imperial College London, United Kingdom
CRISTIAN CADAR, Imperial College London, United Kingdom

Case Study: Cross-Patch Divergences in `cut`

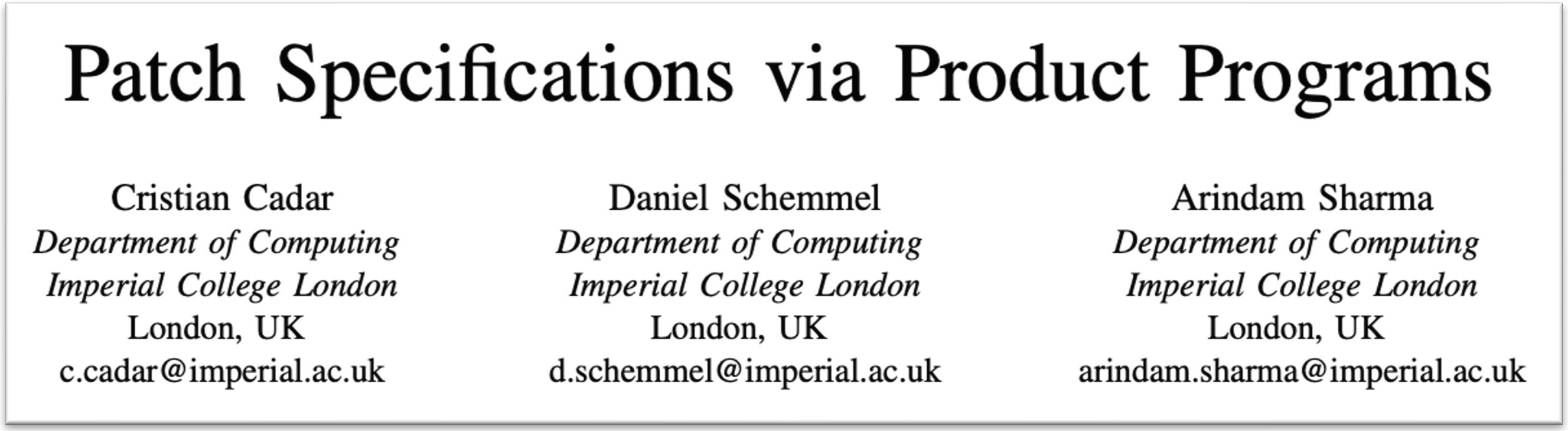
Input	Pre-patch version	Post-patch version
`cut -c1-3,8- -output-d=: file` (file is “**abcdefg**”)	**`abc`**	***Buffer overflow***
`cut -b0-2,2- --output-d=: file` (file contains “**abc**”)	**`abc`**	***Signal abort***
`cut -s -d: -f0- file` (file is “**:::\n:1**”)	**`:::\n:1`**	**`\n\n`**
`cut -d: -f1,0- file` (file is “**a:b:c**”)	**`a:b:c`**	**`a`**

Case Study: Cross-Patch Divergences in `cut`

- Expected or unexpected changes?
- Patch intent or bug?
- Need specifications to decide!

`cut -s -d: -f0- file` `(file is “:::`**`\n:1`**`”)`	**`:::\n:1`**	**`\n\n`**
`cut -d: -f1,0- file` `(file is “`**`a:b:c`**`”)`	**`a:b:c`**	**`a`**

We need to reason about semantics!



Patch Specifications via Product Programs

Cristian Cadar
Department of Computing
Imperial College London
London, UK
c.cadar@imperial.ac.uk

Daniel Schemmel
Department of Computing
Imperial College London
London, UK
d.schemmel@imperial.ac.uk

Arindam Sharma
Department of Computing
Imperial College London
London, UK
arindam.sharma@imperial.ac.uk

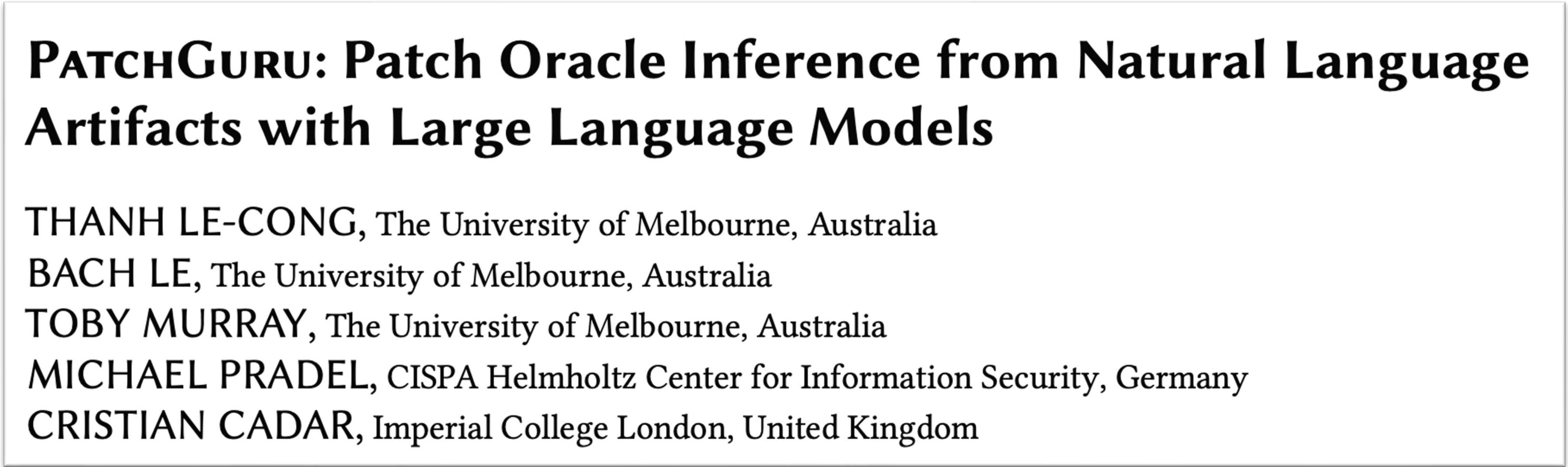
Specifications encoding
cross-patch properties

```
- z = x >> 2
+ z = x >> 3

assert z_post == 2 * z_pre
```

- Manually wrote patch specs for several complex patches
- Found semantic bugs that escaped human reviewing

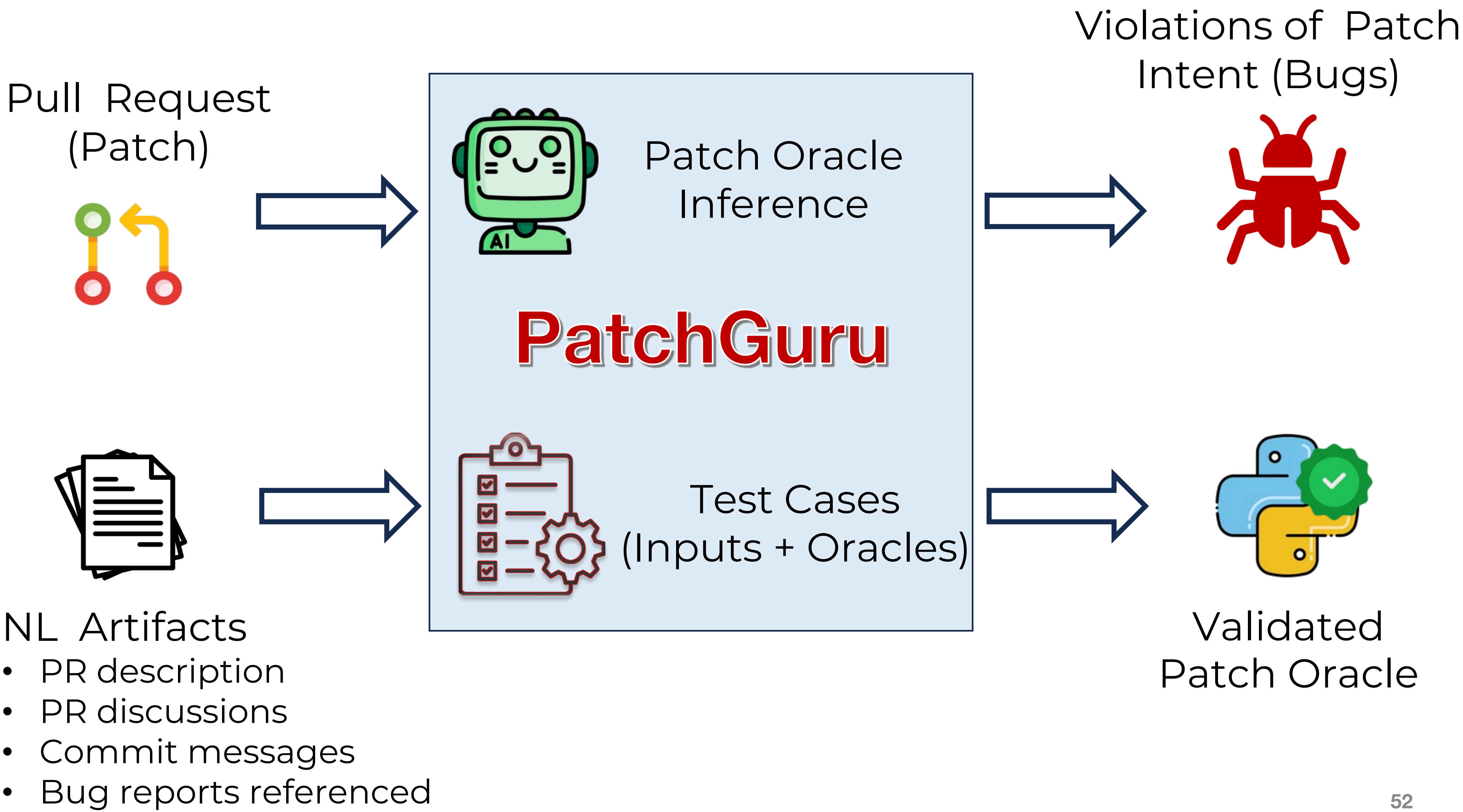
Can LLMs Infer Patch Specs?



PatchGuru: Patch Oracle Inference from Natural Language Artifacts with Large Language Models

THANH LE-CONG, The University of Melbourne, Australia
BACH LE, The University of Melbourne, Australia
TOBY MURRAY, The University of Melbourne, Australia
MICHAEL PRADEL, CISPA Helmholtz Center for Information Security, Germany
CRISTIAN CADAR, Imperial College London, United Kingdom

Patch oracle = weaker form of patch spec that holds on some given inputs



Pull Request
(Patch)

NL Artifacts

- PR description
- PR discussions
- Commit messages
- Bug reports referenced

PatchGuru

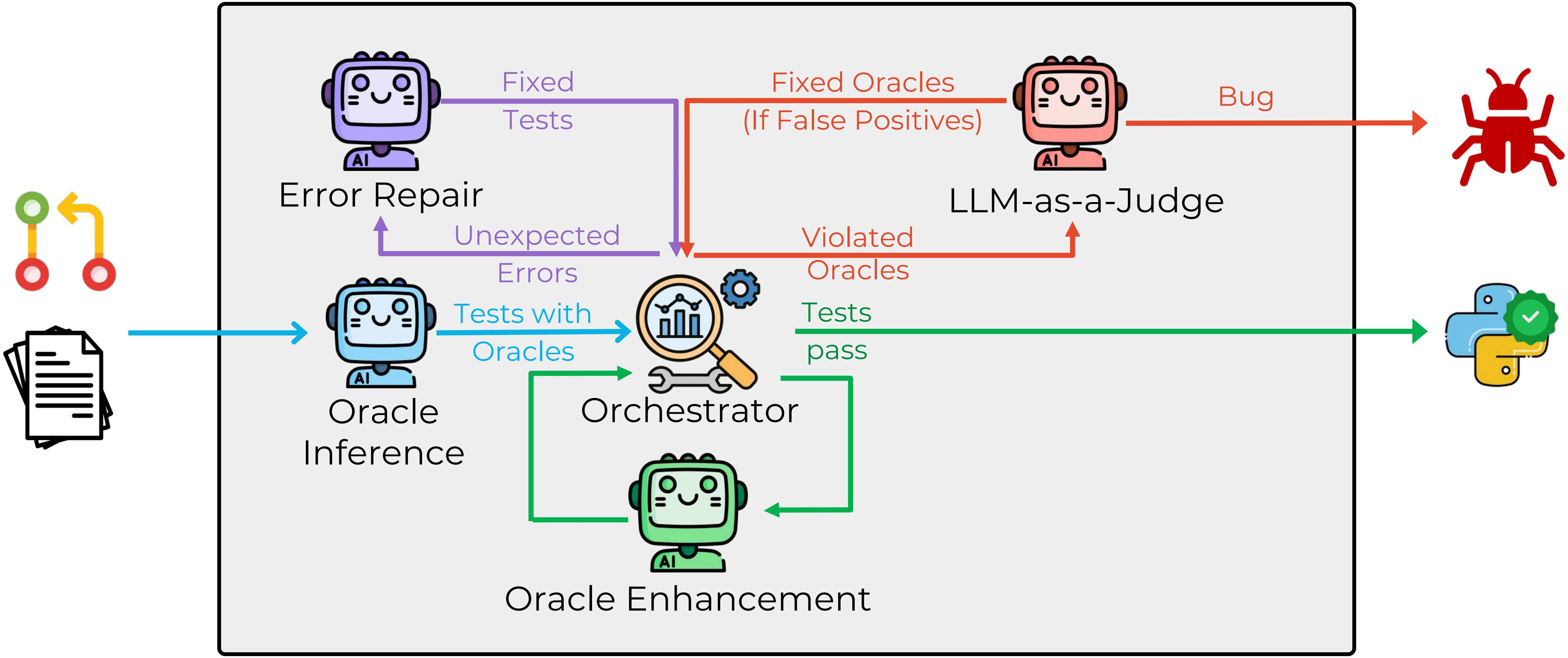
Patch Oracle
Inference

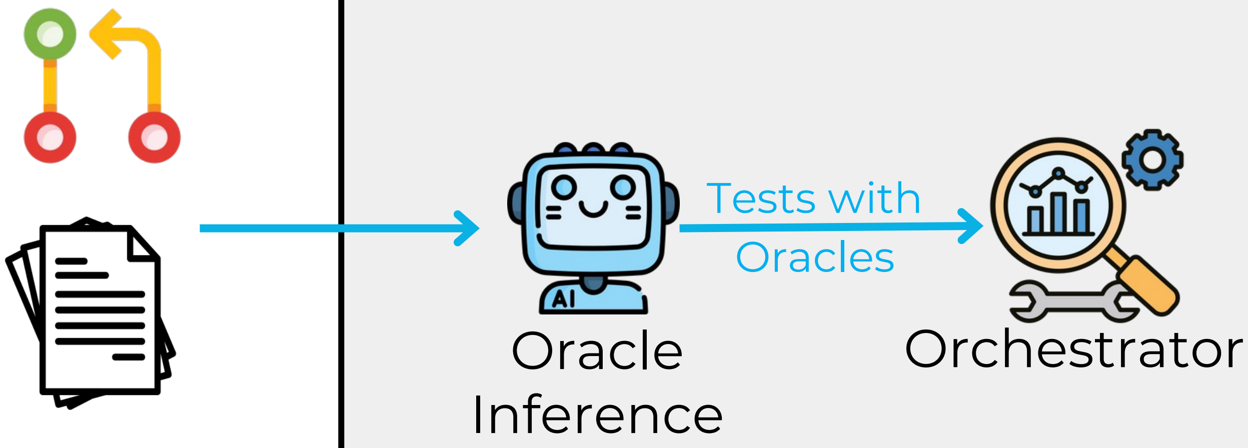
Test Cases
(Inputs + Oracles)

Violations of Patch
Intent (Bugs)

Validated
Patch Oracle

PatchGuru


Fixed
Tests
Fixed Oracles
(If False Positives)
Bug
Error Repair
LLM-as-a-Judge
Unexpected
Errors
Violated
Oracles
Tests with
Oracles
Tests
pass
Oracle
Inference
Orchestrator
Oracle Enhancement
AI
AI
AI
AI



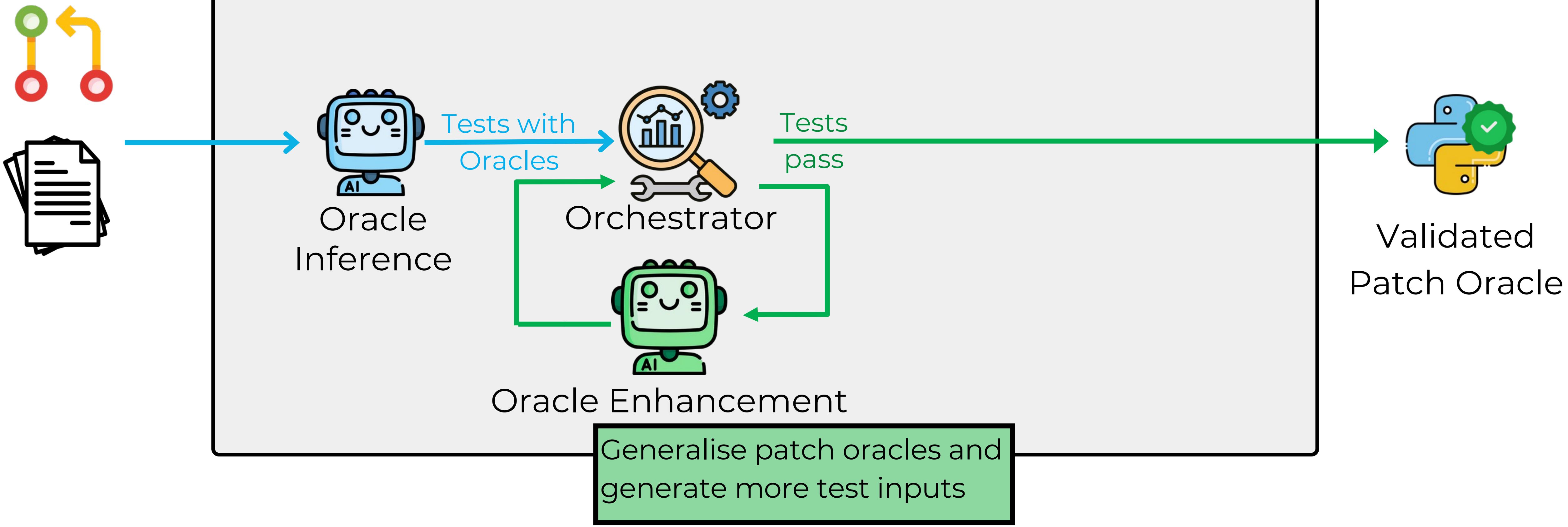
Tests with Oracles

Oracle Inference

Orchestrator

Infer patch oracles and generate test cases for:

- Preserved behaviours
- Changed behaviours
- New behaviours



Tests with Oracles

Tests pass

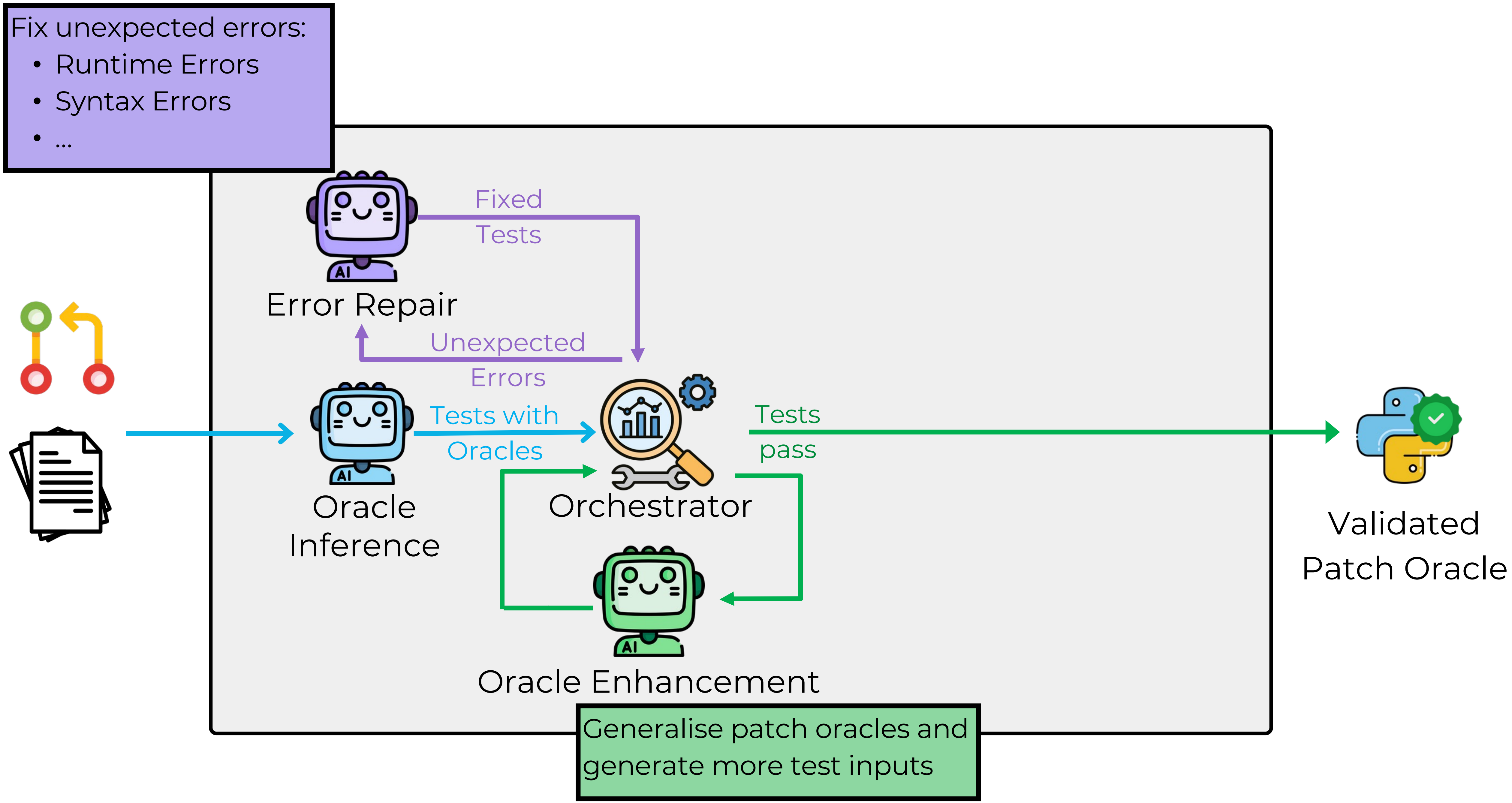
Oracle Inference

Orchestrator

Oracle Enhancement

Validated Patch Oracle

Generalise patch oracles and generate more test inputs



Fix unexpected errors:
- Runtime Errors
- Syntax Errors
- ...

Error Repair

Fixed Tests

Unexpected Errors

Oracle Inference

Tests with Oracles

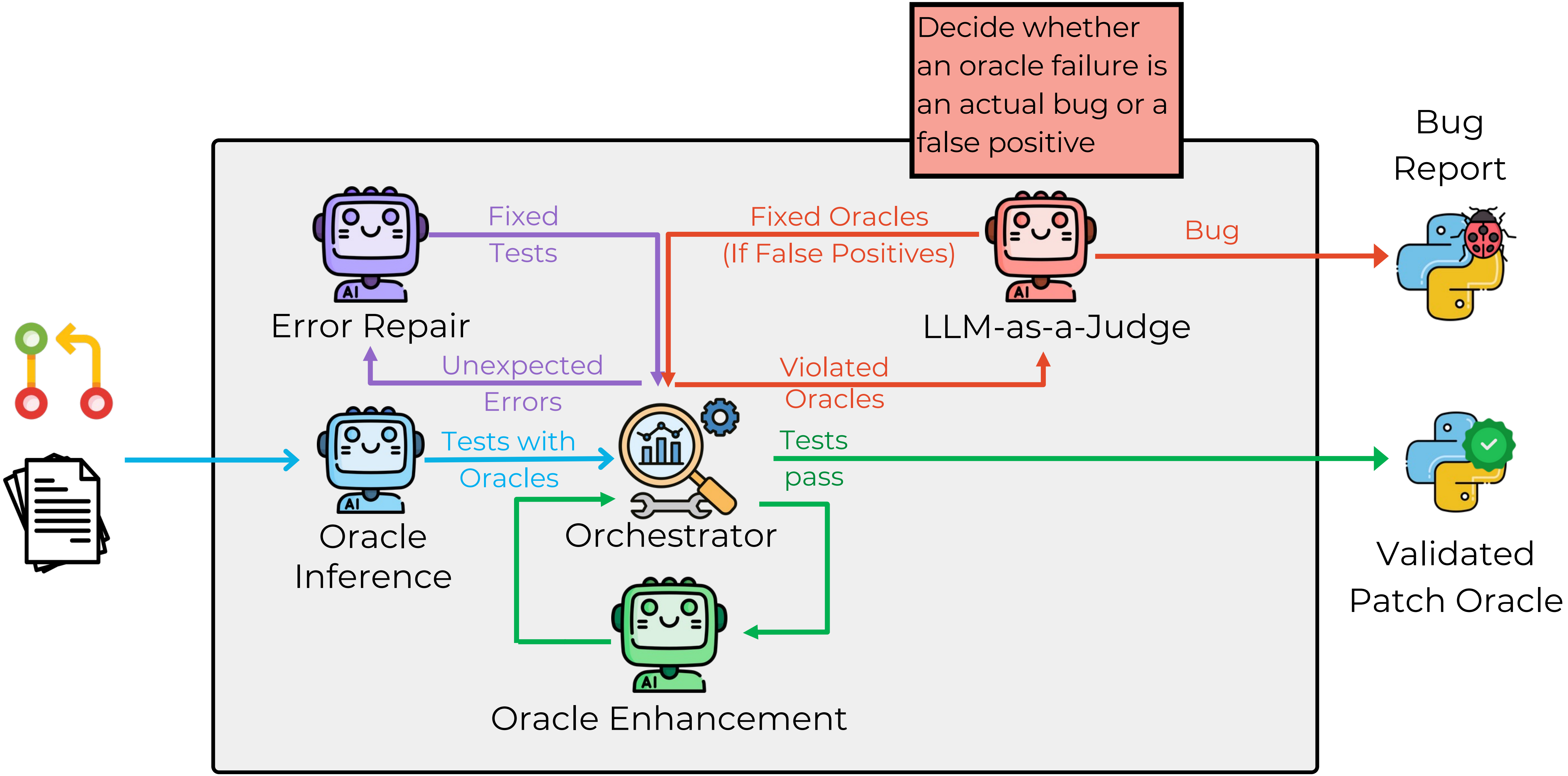
Orchestrator

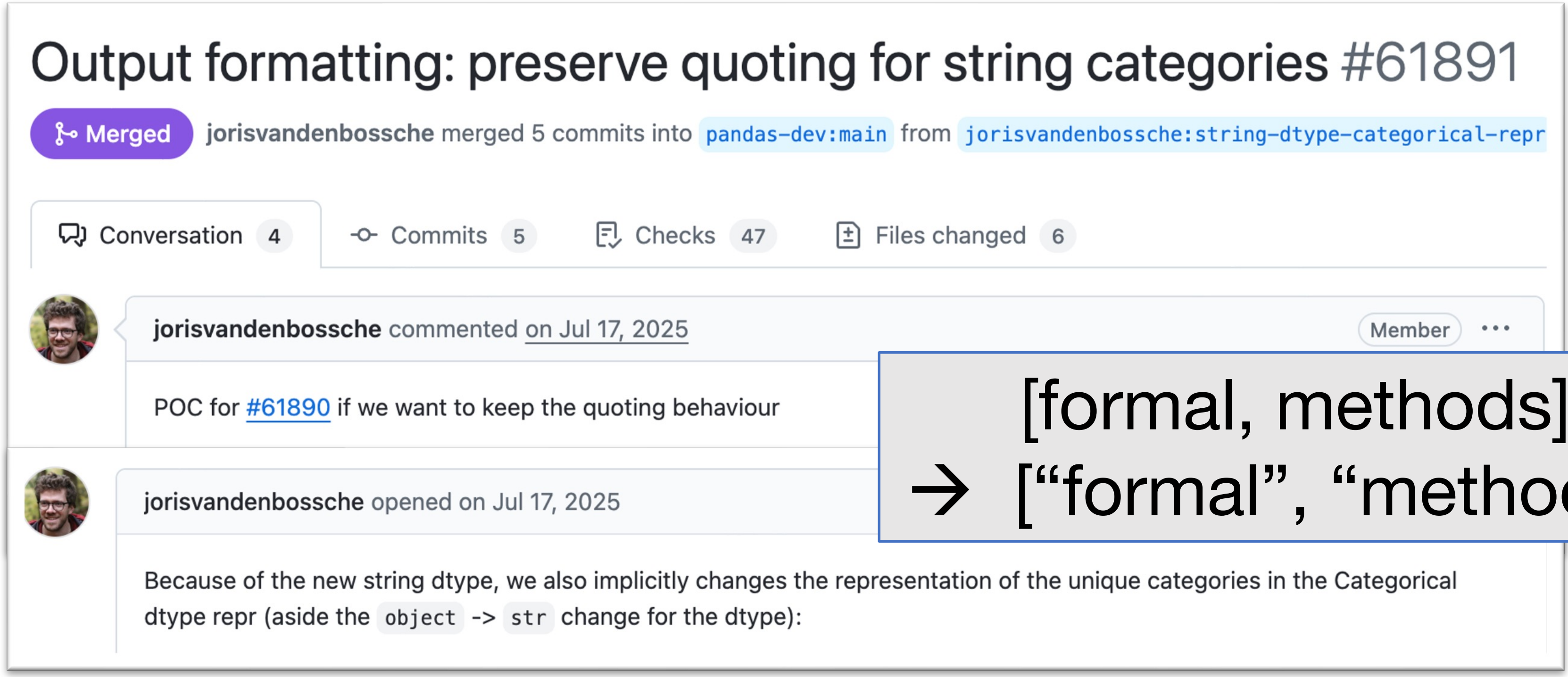
Tests pass

Oracle Enhancement

Validated Patch Oracle

Generalise patch oracles and generate more test inputs


Decide whether an oracle failure is an actual bug or a false positive
Bug Report
Fixed Tests
Fixed Oracles (If False Positives)
Bug
Error Repair
LLM-as-a-Judge
Unexpected Errors
Violated Oracles
Tests with Oracles
Tests pass
Oracle Inference
Orchestrator
Validated Patch Oracle
Oracle Enhancement



Output formatting: preserve quoting for string categories #61891

jorisvandenbossche merged 5 commits into `pandas-dev:main` from `jorisvandenbossche:string-dtype-categorical-repr`

jorisvandenbossche commented on Jul 17, 2025

POC for #61890 if we want to keep the quoting behaviour

jorisvandenbossche opened on Jul 17, 2025

Because of the new string dtype, we also implicitly changes the representation of the unique categories in the Categorical dtype repr (aside the `object` -> `str` change for the dtype):

[formal, methods]
→ [“formal”, “methods”]

```
...
# for each item in out_post:
if (item_post.startswith('"') and item_post.endswith('"')) or
   (item_post.startswith("'") and item_post.endswith("'")):
   assert item_post[1:-1] == item_pre
```

Output formatting: preserve quoting for string categories #61891

jorisvandenbossche merged 5 commits into pandas-dev:main from jorisvandenbossche:string-dtype-categorical-repr

jorisvandenbossche commented on Jul 17, 2025

POC for #61890 if we want to keep the quoting behaviour

jorisvandenbossche opened on Jul 17, 2025

Because of the new string dtype, we also implicitly changes the representation of the unique categories in the Categorical dtype repr (aside the `object` -> `str` change for the dtype):

[formal, methods]
→ ["formal", "methods"]

```
...
# for each item in out_post:
if (item_post.startswith('"') and item_post.endswith('"')) or
   (item_post.startswith("'") and item_post.endswith("'")):
   assert item_post[1:-1] == item_pre
```

Output formatting: preserve quoting for string categories #61891

Merged jorisvandenbossche merged 5 commits into `pandas-dev:main` from `jorisvandenbossche:string-dtype-categorical-repr`

jorisvandenbossche commented on Jul 17, 2025

POC for #61890 if we want to keep the quoting behaviour

jorisvandenbossche opened on Jul 17, 2025

Because of the new string dtype, we also implicitly changes the representation of the unique categories in the Categorical dtype repr (aside the `object` -> `str` change for the dtype):

[formal, methods]
→ ["formal", "methods"]

```
...
# for each item in out_post:
if (item_post.startswith('"') and item_post.endswith('"')) or
   (item_post.startswith("'") and item_post.endswith("'")):
   assert item_post[1:-1] == item_pre
```

400 PRs: Pandas, Keras, SciPy, Marshmallows

Bugs: 24 = 8 + 5 + 7 + 4

17 code

7 doc

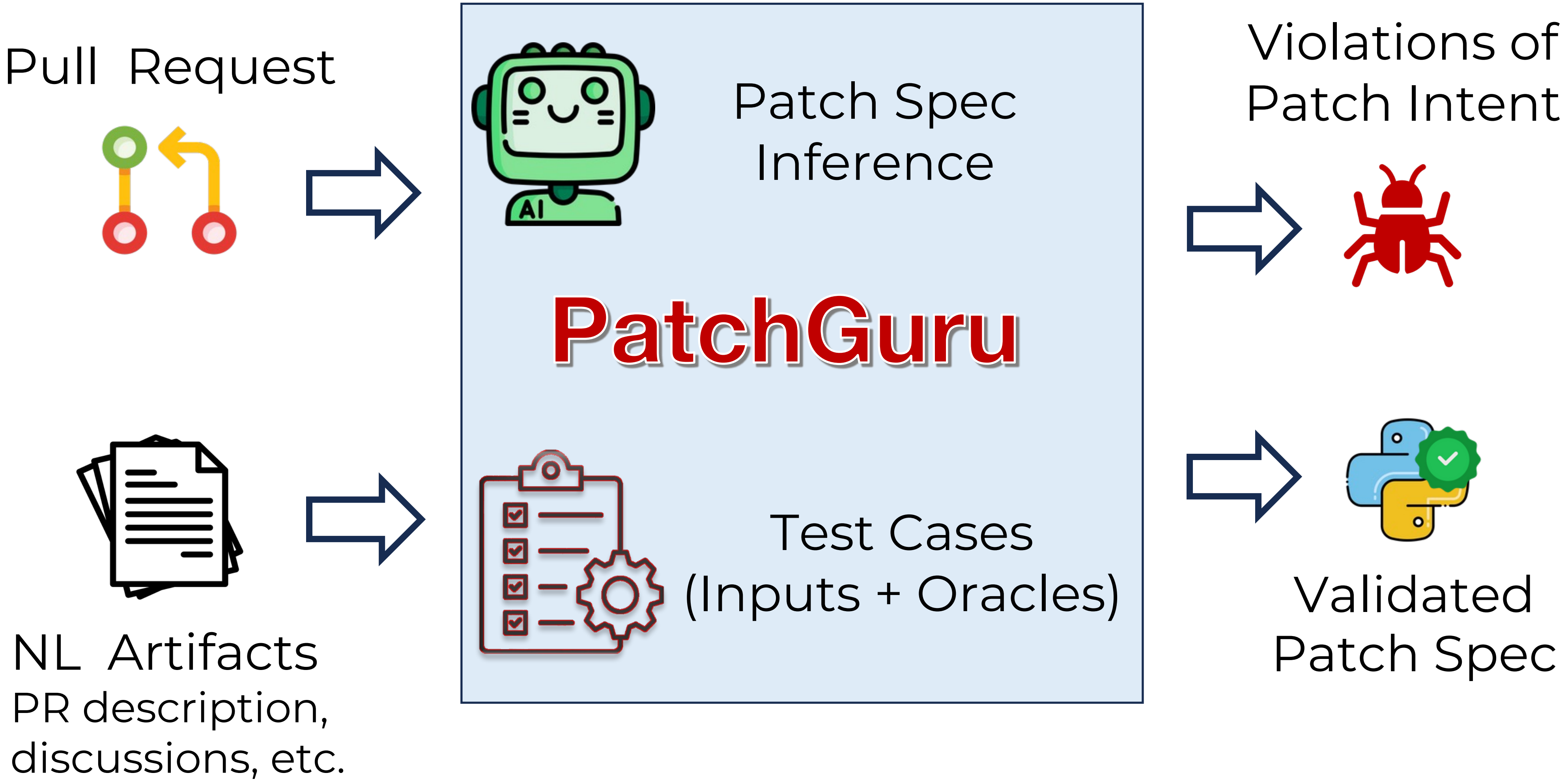
12 new

11 w/ fixes

Violations of patch intent!

15 FPs

8: Unrealistic mocking and violated preconditions
3: Misunderstanding of developer intent
2: Incorrect assumptions about environment
2: Unclear (conservatively classified as FPs)



Pull Request

NL Artifacts

PR description, discussions, etc.

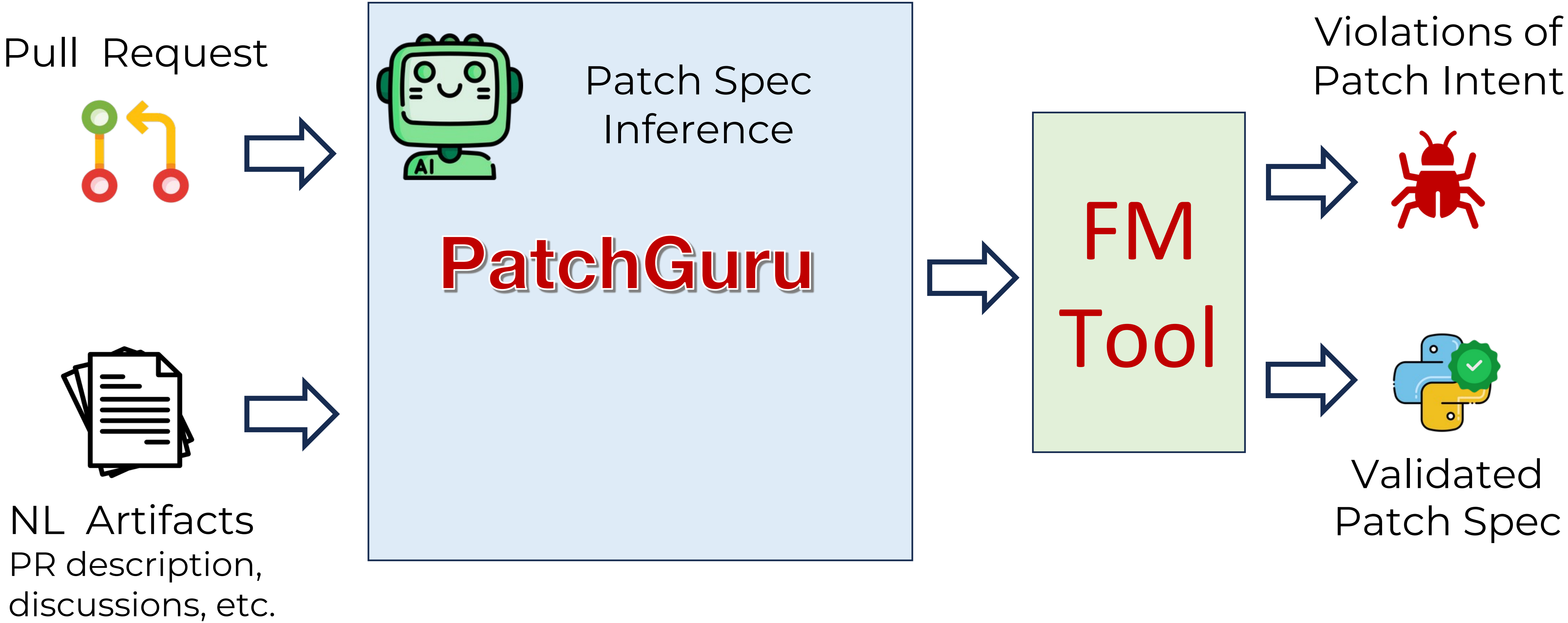
Patch Spec Inference

PatchGuru

Test Cases (Inputs + Oracles)

Violations of Patch Intent

Validated Patch Spec



Pull Request

NL Artifacts
PR description,
discussions, etc.

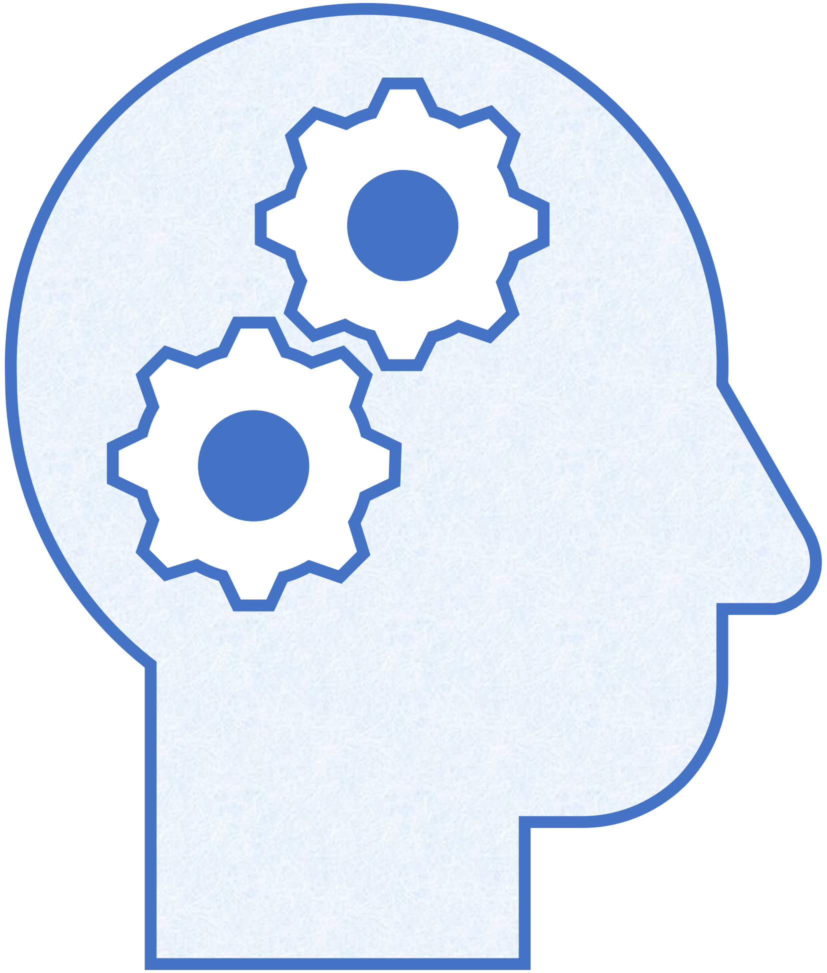
Patch Spec
Inference

PatchGuru

FM
Tool

Violations of
Patch Intent

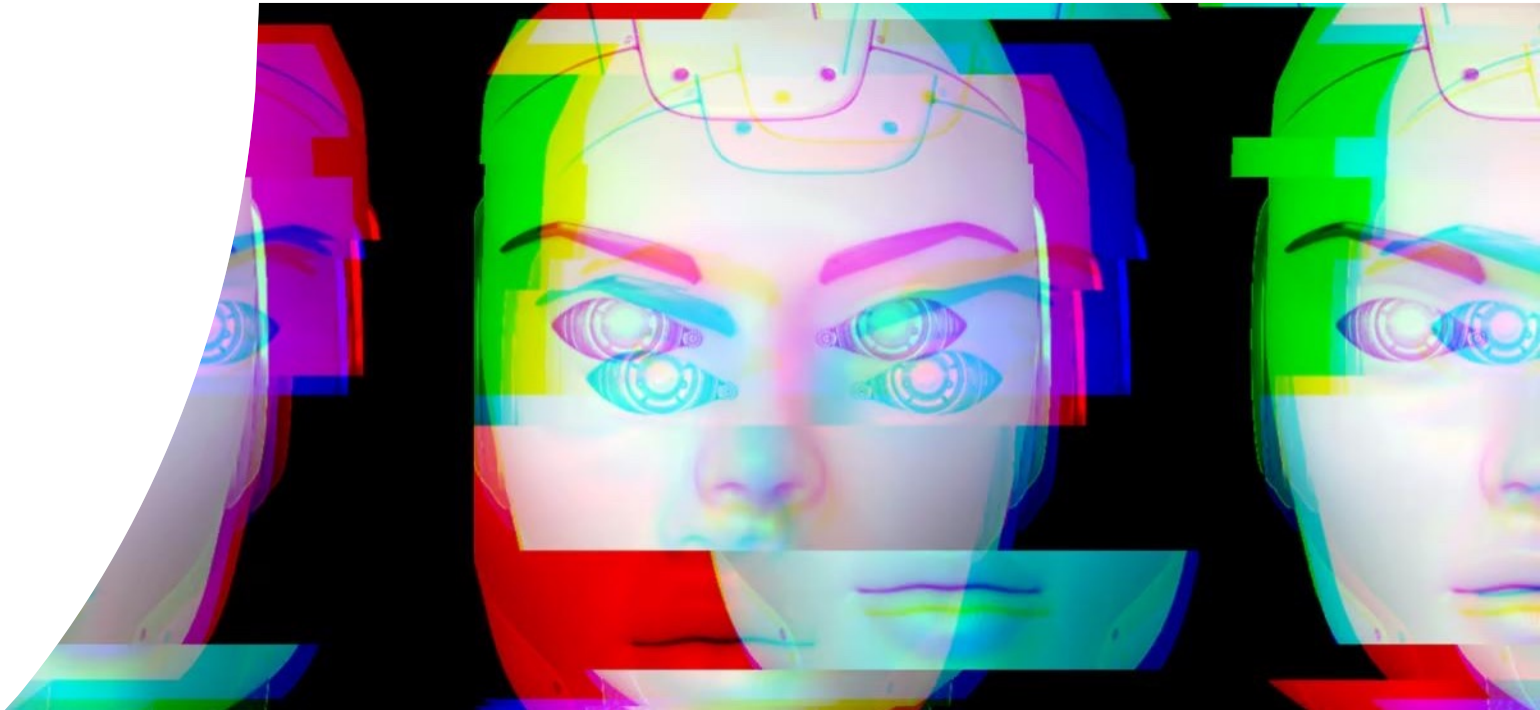
Validated
Patch Spec



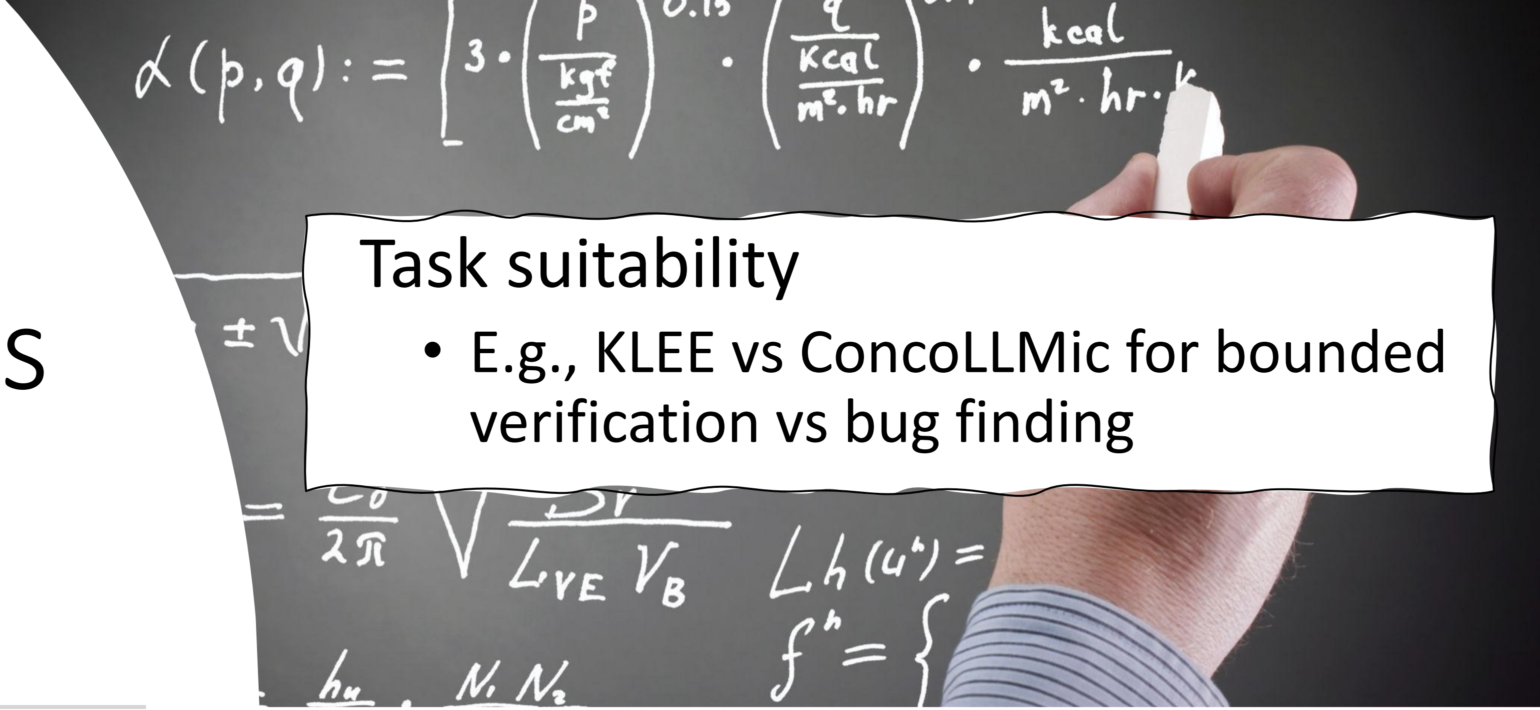
Traditional vs AI-based Program Analysis

Formal guarantees

Hard-to-diagnose hallucinations



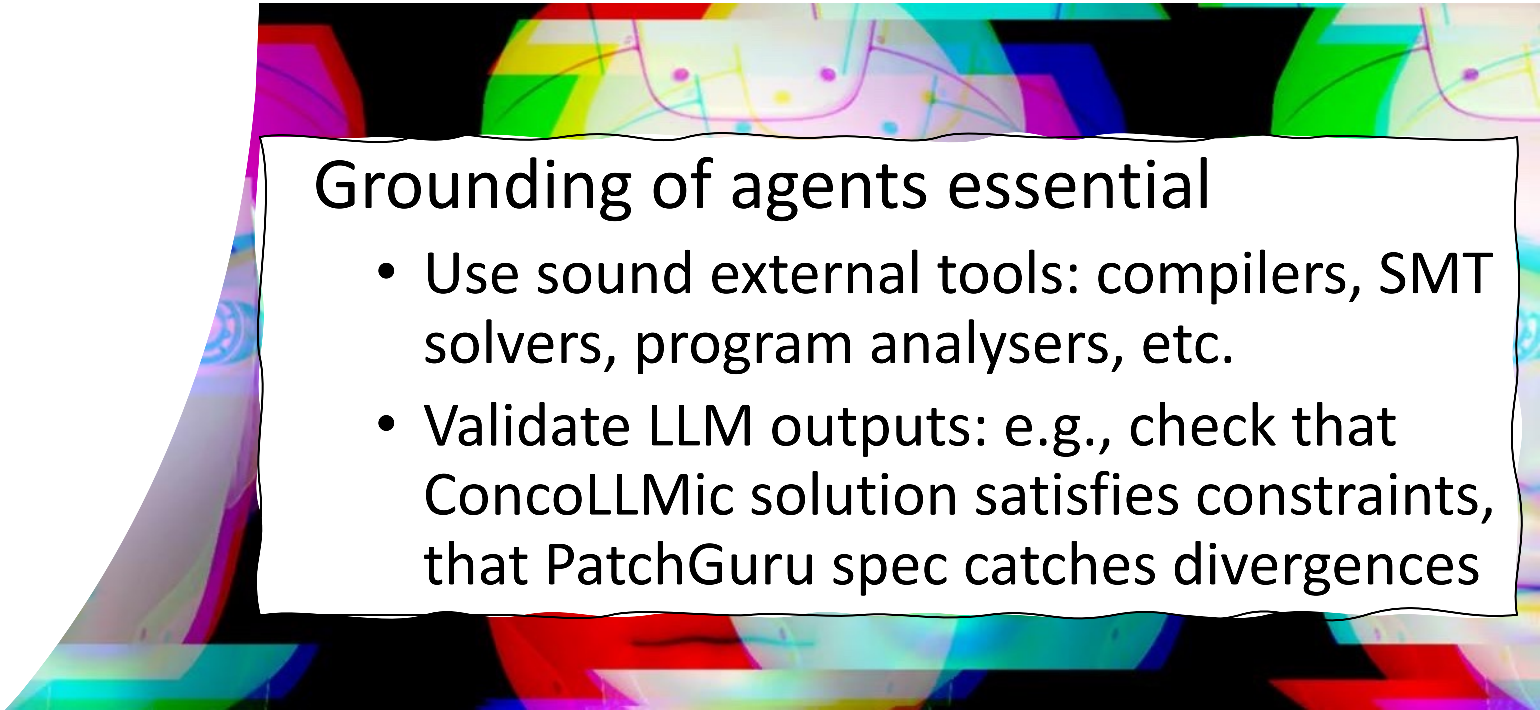
Formal guarantees



Task suitability

- E.g., KLEE vs ConcoLLMic for bounded verification vs bug finding

Hard-to-diagnose hallucinations



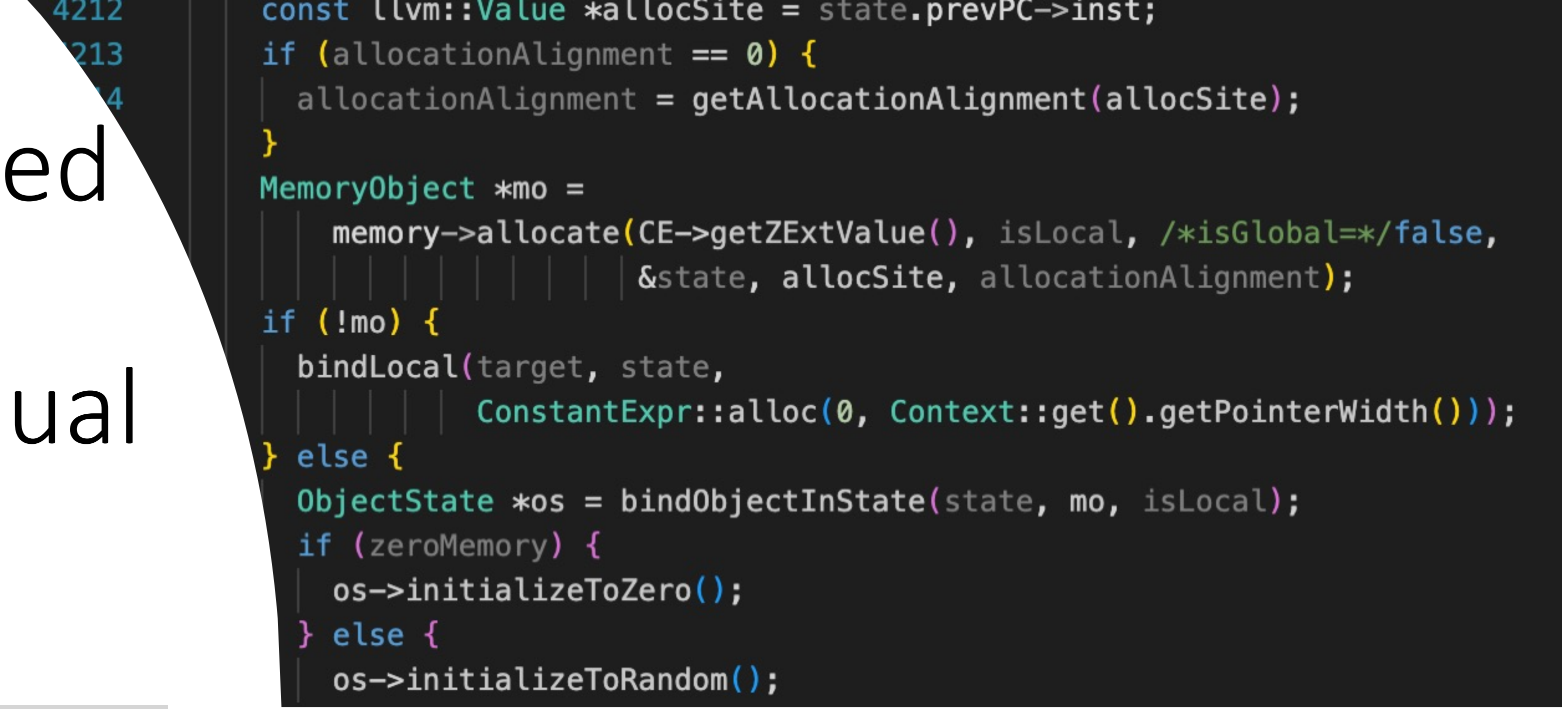
Grounding of agents essential

- Use sound external tools: compilers, SMT solvers, program analysers, etc.
- Validate LLM outputs: e.g., check that ConcoLLMic solution satisfies constraints, that PatchGuru spec catches divergences

Limited to certain programming languages and system types

Generalise across languages and system types, including polyglot code

Limited to code-based analysis (potentially enhanced with manual specifications)



```
const llvm::Value *allocSite = state.prevPC->inst;
if (allocationAlignment == 0) {
  allocationAlignment = getAllocationAlignment(allocSite);
}
MemoryObject *mo =
    memory->allocate(CE->getZExtValue(), isLocal, /*isGlobal=*/false,
                     &state, allocSite, allocationAlignment);
if (!mo) {
  bindLocal(target, state,
            ConstantExpr::alloc(0, Context::get().getPointerWidth()));
} else {
  ObjectState *os = bindObjectInState(state, mo, isLocal);
  if (zeroMemory) {
    os->initializeToZero();
  } else {
    os->initializeToRandom();
```

Can leverage NL artifacts such as documentation, previously inaccessible

Scalability challenges: gets bogged down in low-level details

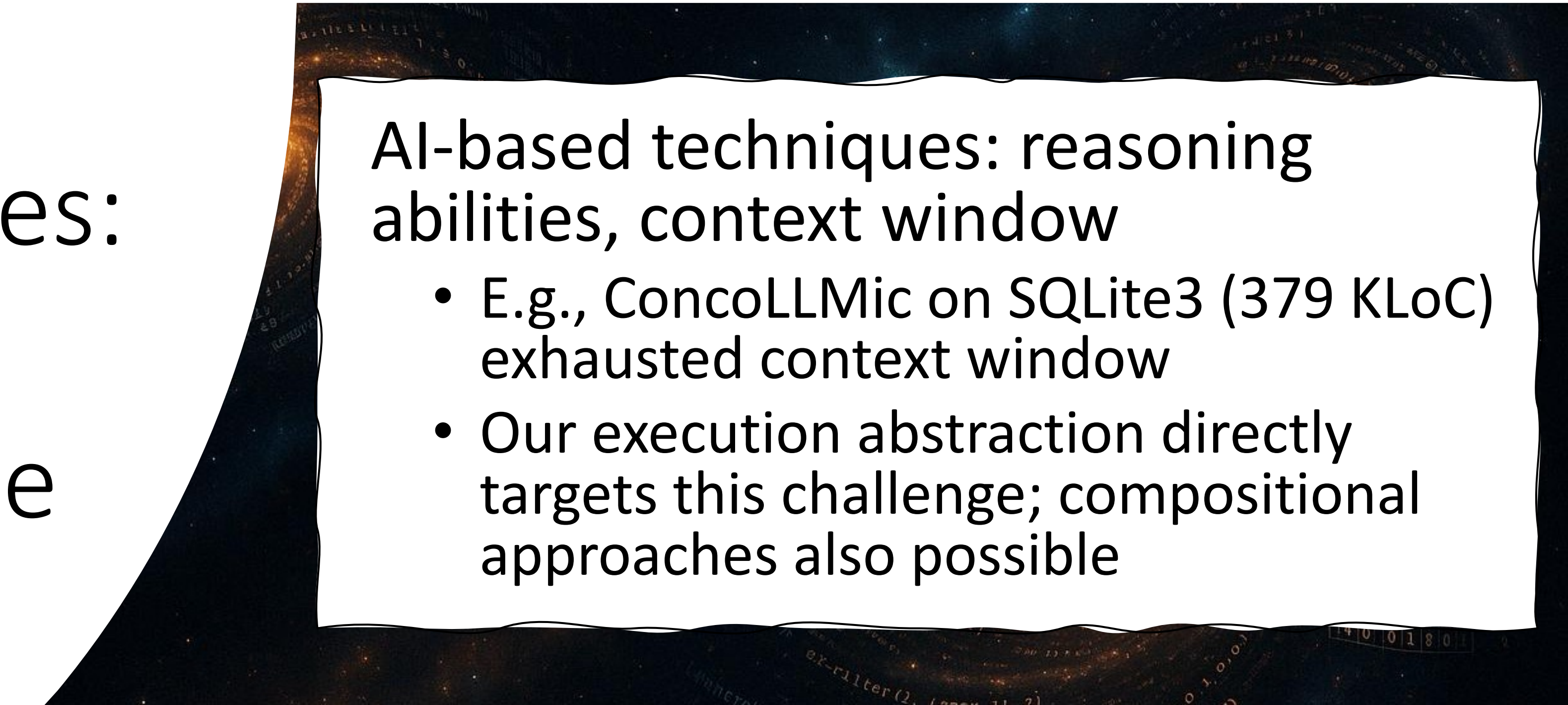
Scalability challenges: reasoning abilities, context window size

Scalability challenges: gets bogged down in low-level details

Traditional techniques: often unable to adjust abstraction level

---

Scalability challenges: reasoning abilities, context window size

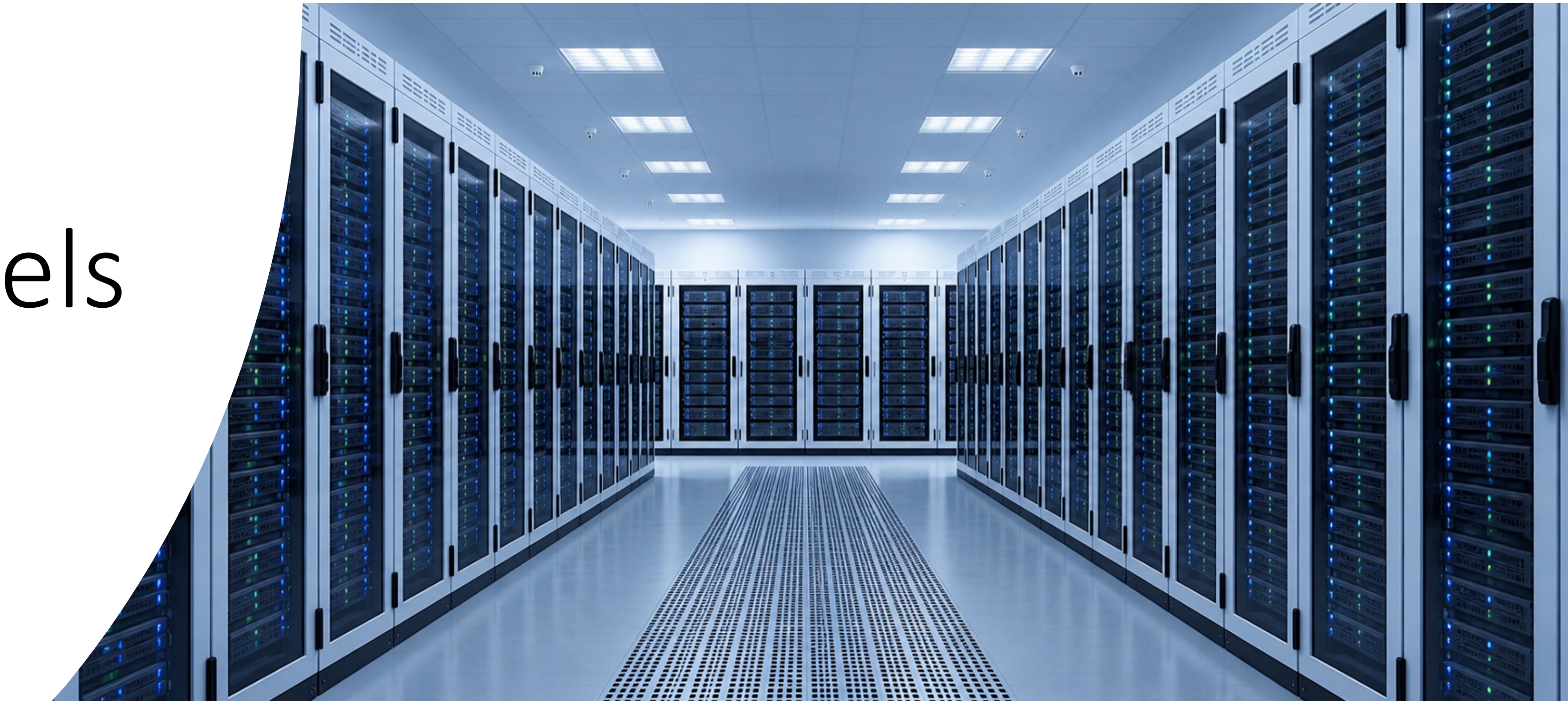


AI-based techniques: reasoning abilities, context window

- E.g., ConcoLLMic on SQLite3 (379 KLoC) exhausted context window
- Our execution abstraction directly targets this challenge; compositional approaches also possible

Cheap and easy to run locally

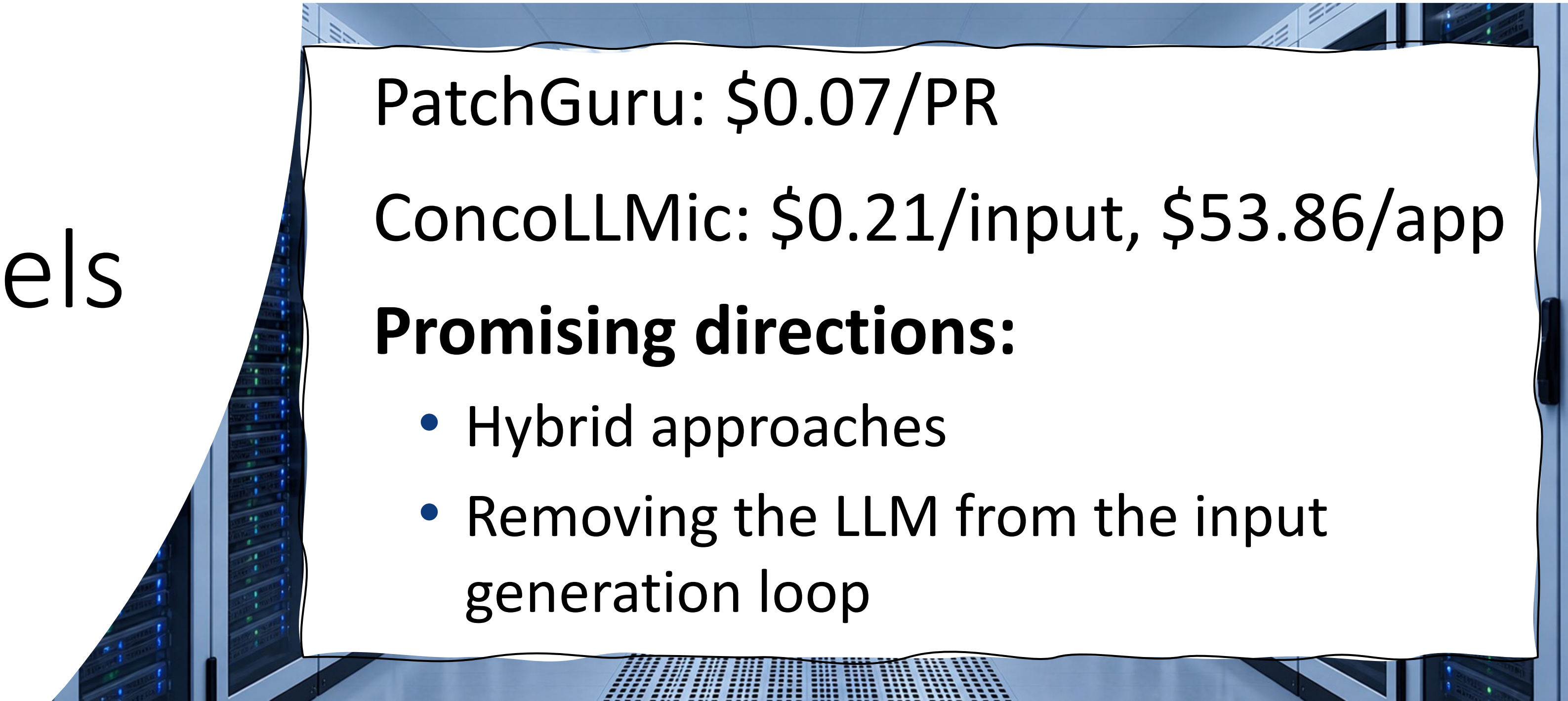
Foundational models are expensive



Cheap and easy to run locally

Open-source developer with a laptop vs big company with huge computing budget

Foundational models are expensive



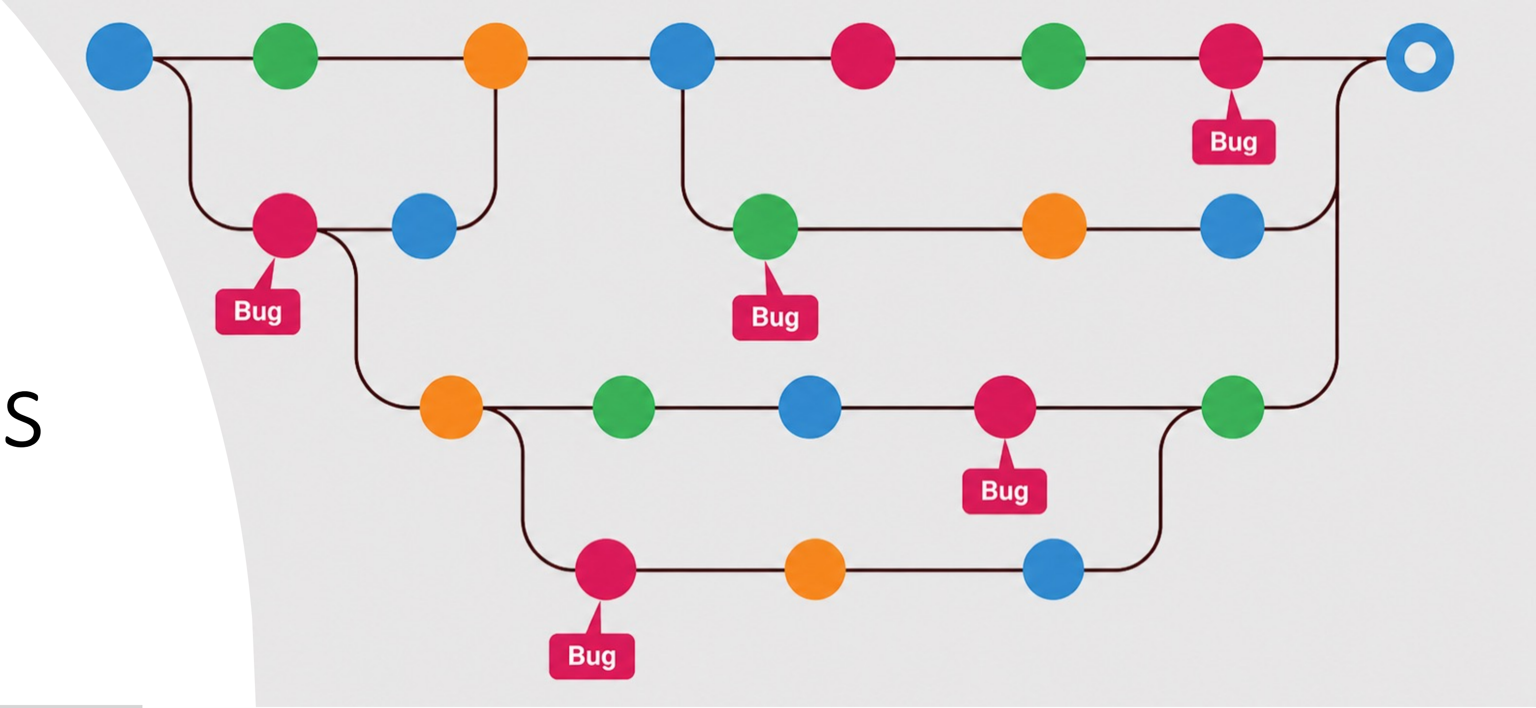
PatchGuru: $0.07/PR

ConcoLLMic: $0.21/input, $53.86/app

Promising directions:

- Hybrid approaches
- Removing the LLM from the input generation loop

Reproducible often deterministic results



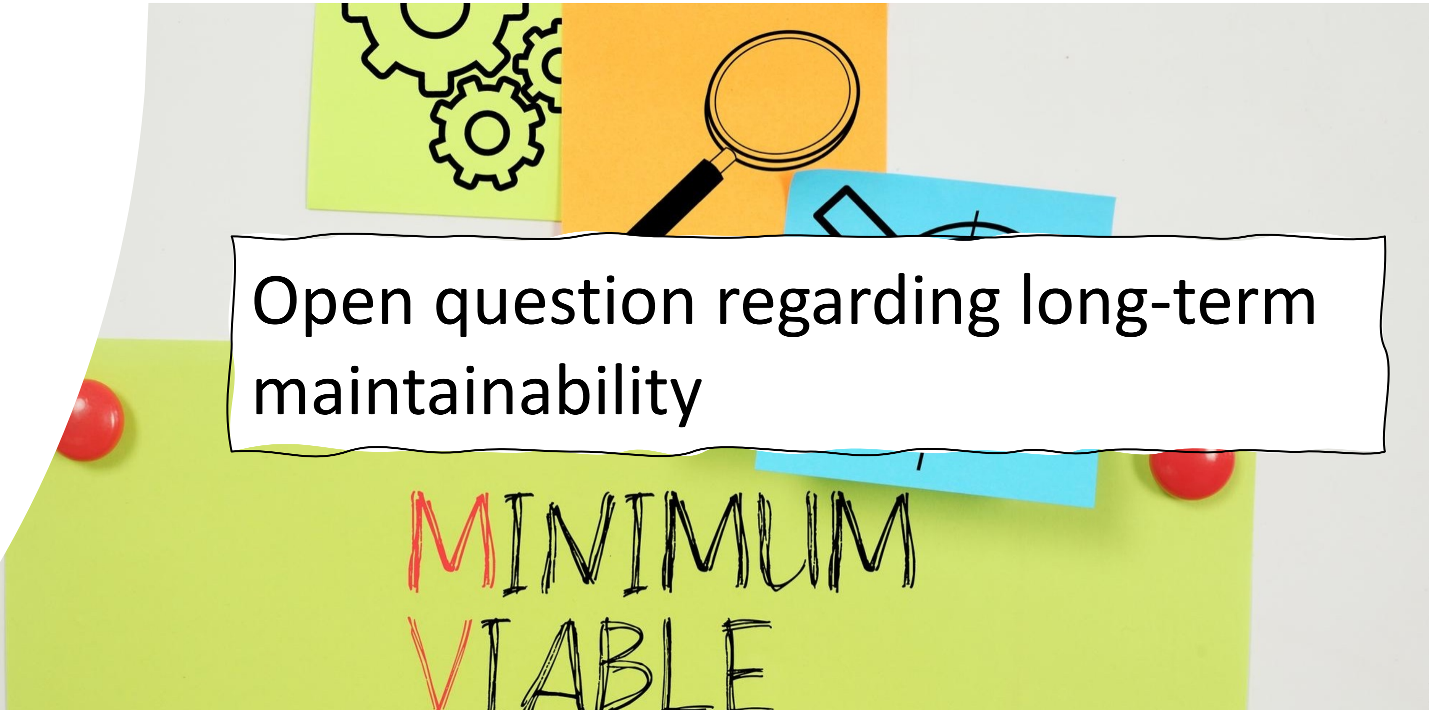
Variability across models and model versions, data leakage concerns

Hard to build, often years of work for mature tools

Often 100x-1000x cheaper to build a decent prototype

Hard to build, often years of work for mature tools

Often 100x-1000x cheaper to build a decent prototype



Open question regarding long-term maintainability

Going Forward

Democratising Program Reasoning

Support for the full software ecosystem – from configuring and adapting existing tools to building new ones

Inferring Developer Intent

Specification inference leveraging NL artifacts such as design docs, documentation and developer conversations

Autonomously Maintaining Software

Useful software (incl. FM tools!) becomes unmaintained:
AI-driven FM tools could extend software lifetime

Increasing Amount of Verified Software

Verifying more existing code: via AI-driven spec inference and proof search. Creating new verified-by-construction code.

Testing and Analysis in the AI Era

Cristian Cadar

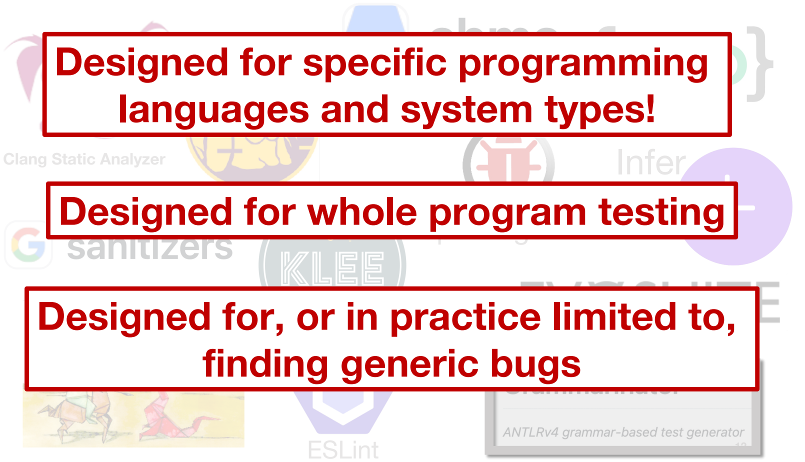
SOFTWARE RELIABILITY GROUP

Department of Computing
Imperial College London

Funded by
Engineering and Physical Sciences Research Council
erc

Keynote at FM 2026
21 May 2026, Tokyo, Japan

Impact of Generative AI on Software Development



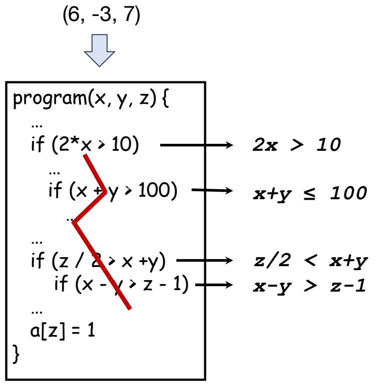
Designed for specific programming languages and system types!

Designed for whole program testing

Designed for, or in practice limited to, finding generic bugs

---

Dynamic Symbolic Execution or Concolic Execution



```
program(x, y, z) {
  ...
  if (2*x > 10)
    ...
    if (x + y > 100)
      ...
  ...
  if (z / 2 > x +y)
    if (x - y > z - 1)
  ...
  a[z] = 1
}
```

`2x > 10`

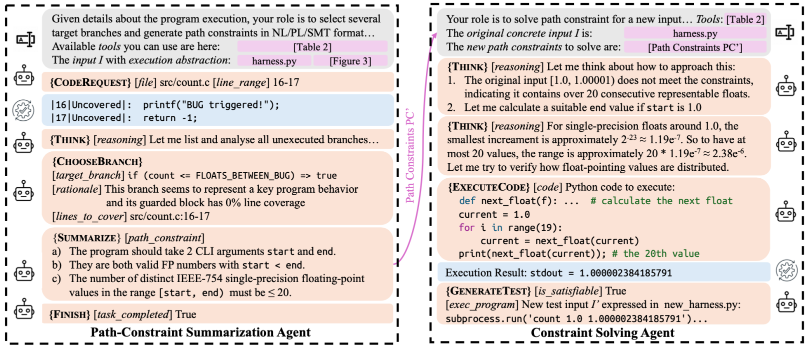
`x+y ≤ 100`

`z/2 < x+y`

`x-y > z-1`

`2x > 10 ∧ x+y ≤ 100 ∧`
`z/2 > x+y ∧ x-y > z-1`

ConcoLLMic in Action



Given details about the program execution, your role is to select several target branches and generate path constraints in NL/PL/SMT format…
Available *tools* you can use are here: [Table 2]
The *input I* with *execution abstraction*: harness.py [Figure 3]

{CodeRequest} [*file*] src/count.c [*line_range*] 16-17

```
|16|Uncovered|:  printf("BUG triggered!");
|17|Uncovered|:  return -1;
```

{Think} [*reasoning*] Let me list and analyse all unexecuted branches…

{ChooseBranch}
[*target_branch*] `if (count <= FLOATS_BETWEEN_BUG) => true`
[*rationale*] This branch seems to represent a key program behavior and its guarded block has 0% line coverage
[*lines_to_cover*] src/count.c:16-17

{Summarize} [*path_constraint*]
a) The program should take 2 CLI arguments `start` and `end`.
b) They are both valid FP numbers with `start < end`.
c) The number of distinct IEEE-754 single-precision floating-point values in the range `[start, end)` must be ≤ 20.

{Finish} [*task_completed*] True

Path-Constraint Summarization Agent

Your role is to solve path constraint for a new input… *Tools*: [Table 2]
The *original concrete input I* is: harness.py
The *new path constraints* to solve are: [Path Constraints PC']

{Think} [*reasoning*] Let me think about how to approach this:
1. The original input [1.0, 1.00001) does not meet the constraints, indicating it contains over 20 consecutive representable floats.
2. Let me calculate a suitable end value if `start` is 1.0

{Think} [*reasoning*] For single-precision floats around 1.0, the smallest increament is approximately $2^{-23} \approx 1.19e^{-7}$. So to have at most 20 values, the range is approximately $20 * 1.19e^{-7} \approx 2.38e^{-6}$. Let me try to verify how float-pointing values are distributed.

{ExecuteCode} [*code*] Python code to execute:

```
def next_float(f): ...  # calculate the next float
current = 1.0
for i in range(19):
    current = next_float(current)
print(next_float(current)); # the 20th value
```

Execution Result: `stdout = 1.000002384185791`

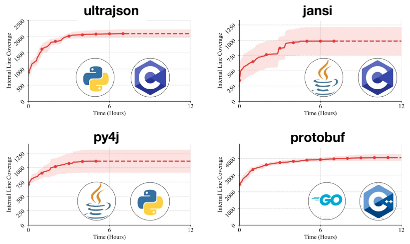
{GenerateTest} [*is_satisfiable*] True
[*exec_program*] New test input *I'* expressed in new_harness.py:
`subprocess.run('count 1.0 1.000002384185791')...`

Constraint Solving Agent

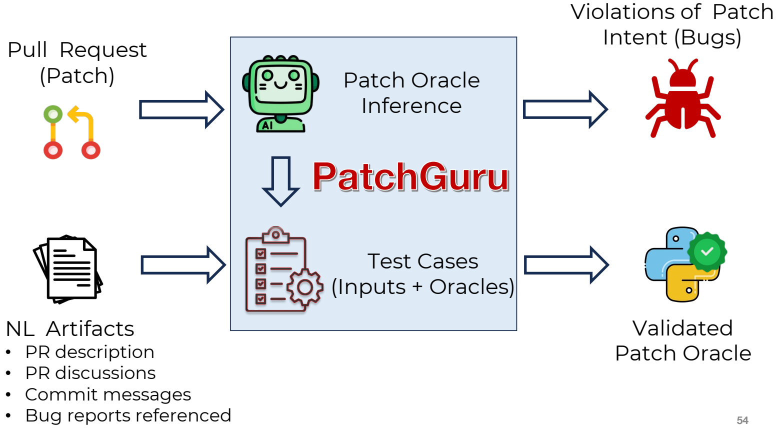
ConcoLLMic with Claude 3.7 Sonnet

Code Coverage: Polyglot Code (12h)

- Instrumentation & reasoning are universal across languages
- Successfully tested polyglot systems
  - Growing coverage + bugs



Line coverage / time



Pull Request (Patch)

NL Artifacts
- PR description
- PR discussions
- Commit messages
- Bug reports referenced

Patch Oracle Inference

PatchGuru

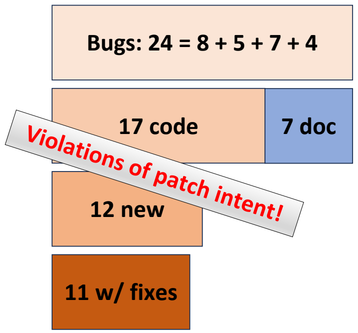
Test Cases (Inputs + Oracles)

Violations of Patch Intent (Bugs)

Validated Patch Oracle

---

400 PRs: Pandas, Keras, SciPy, Marshmallows



Bugs: 24 = 8 + 5 + 7 + 4

17 code

7 doc

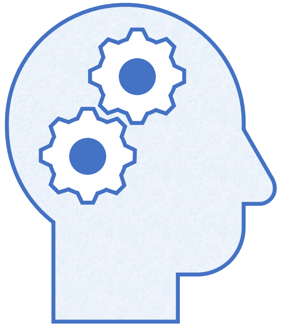
12 new

11 w/ fixes

Violations of patch intent!

15 FPs

- Misunderstanding of developer intent
- Violated preconditions
- Incorrect assumptions about environment
- Unclear (conservatively classified as FPs)



Traditional vs AI-based Program Analysis