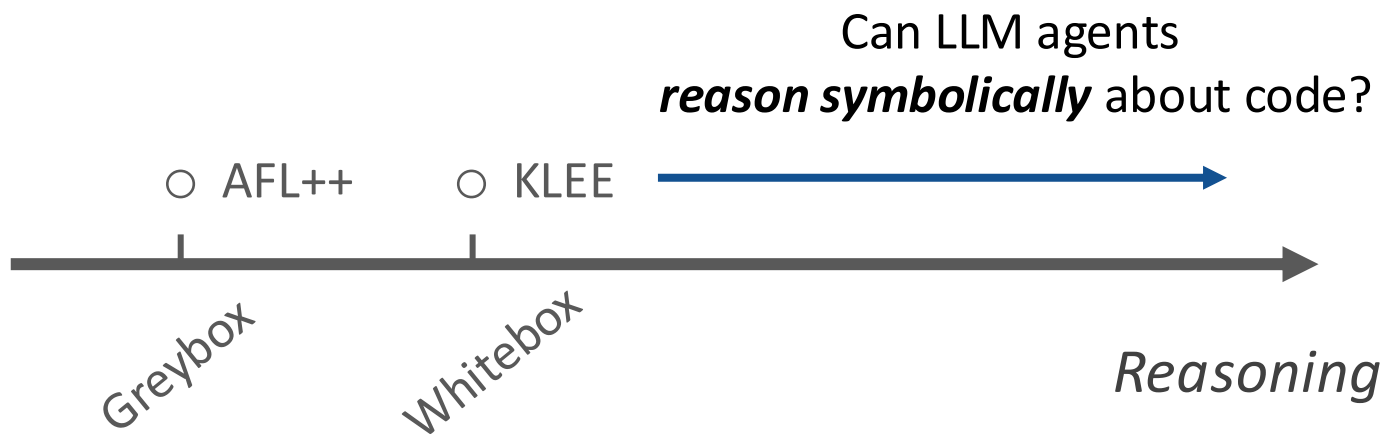


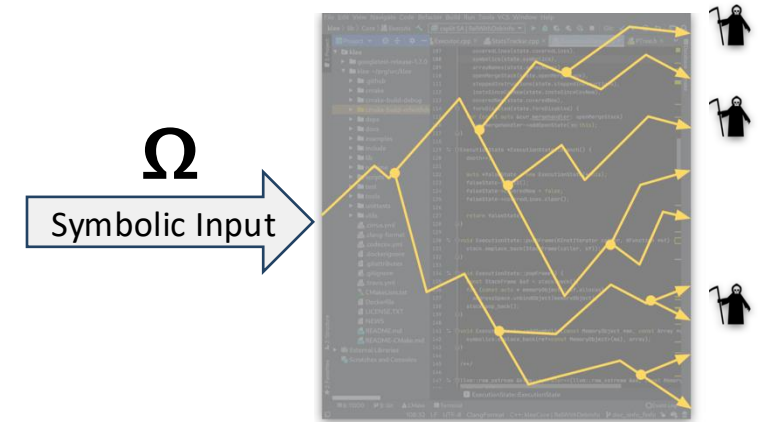


Agentic Concolic Execution



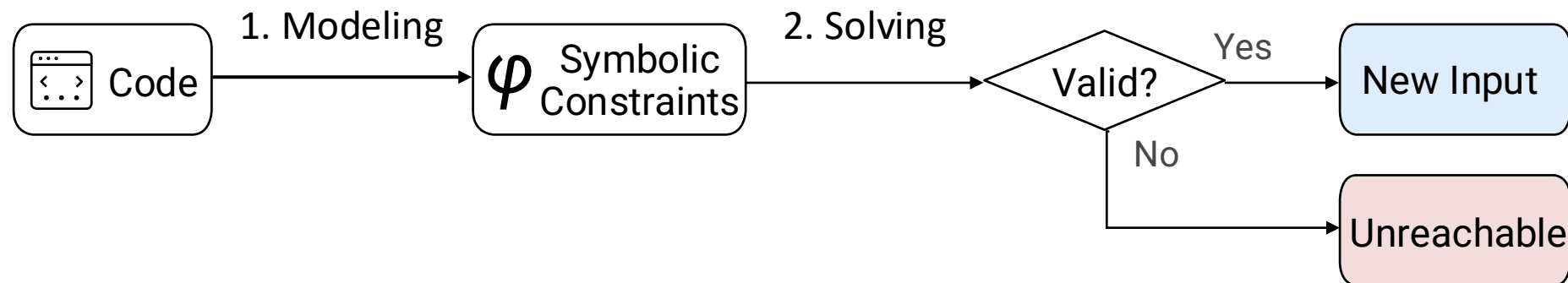
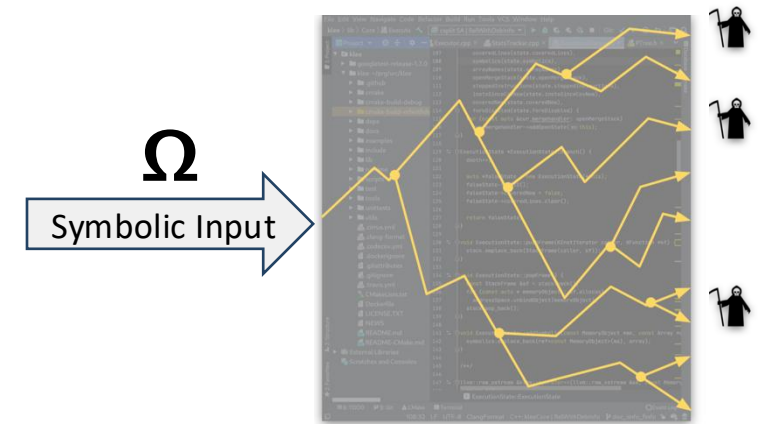
Dynamic Symbolic Execution (DSE)

- Widely-used for program analysis
- Use *symbolic values* as program inputs to systematically explore possible program behaviors

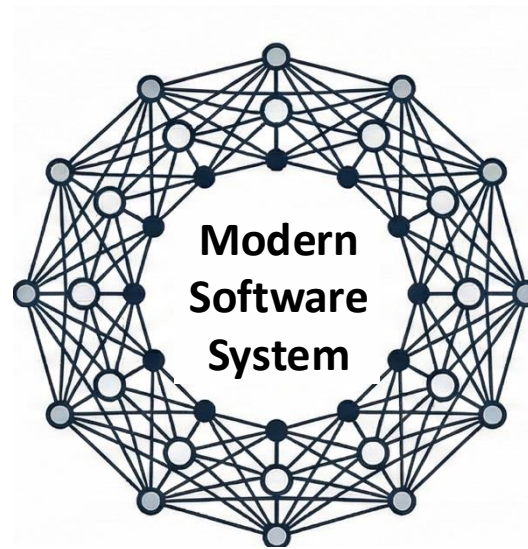


Dynamic Symbolic Execution (DSE)

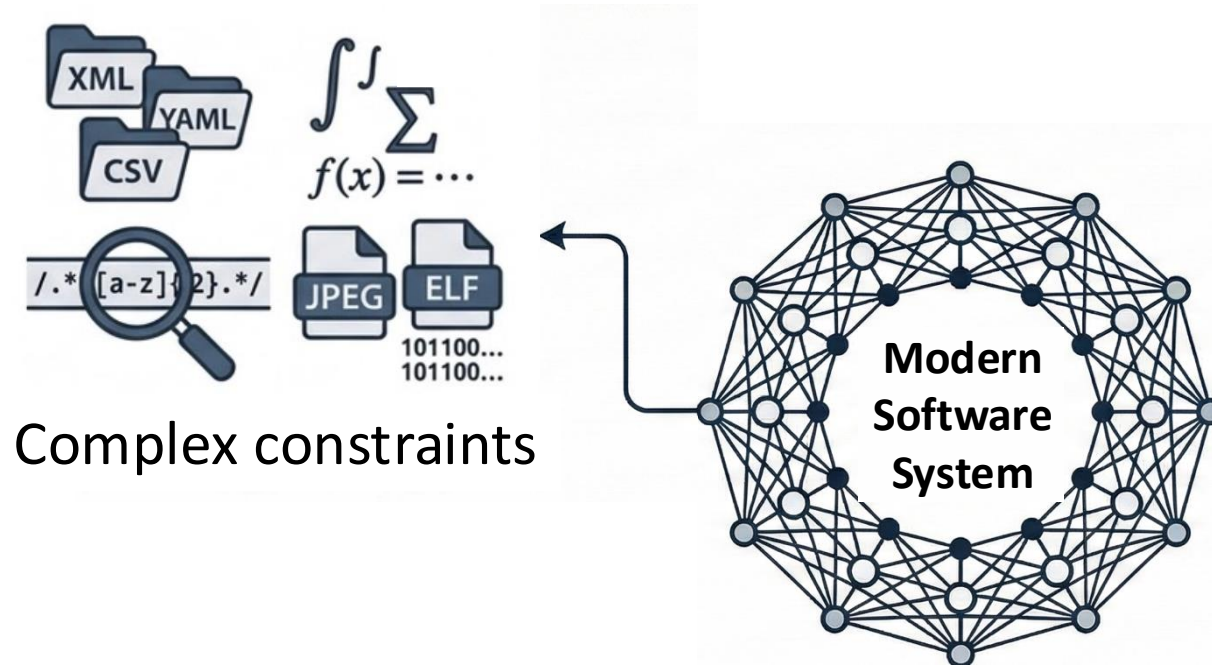
- Widely-used for program analysis
- Use *symbolic values* as program inputs to systematically explore possible program behaviors



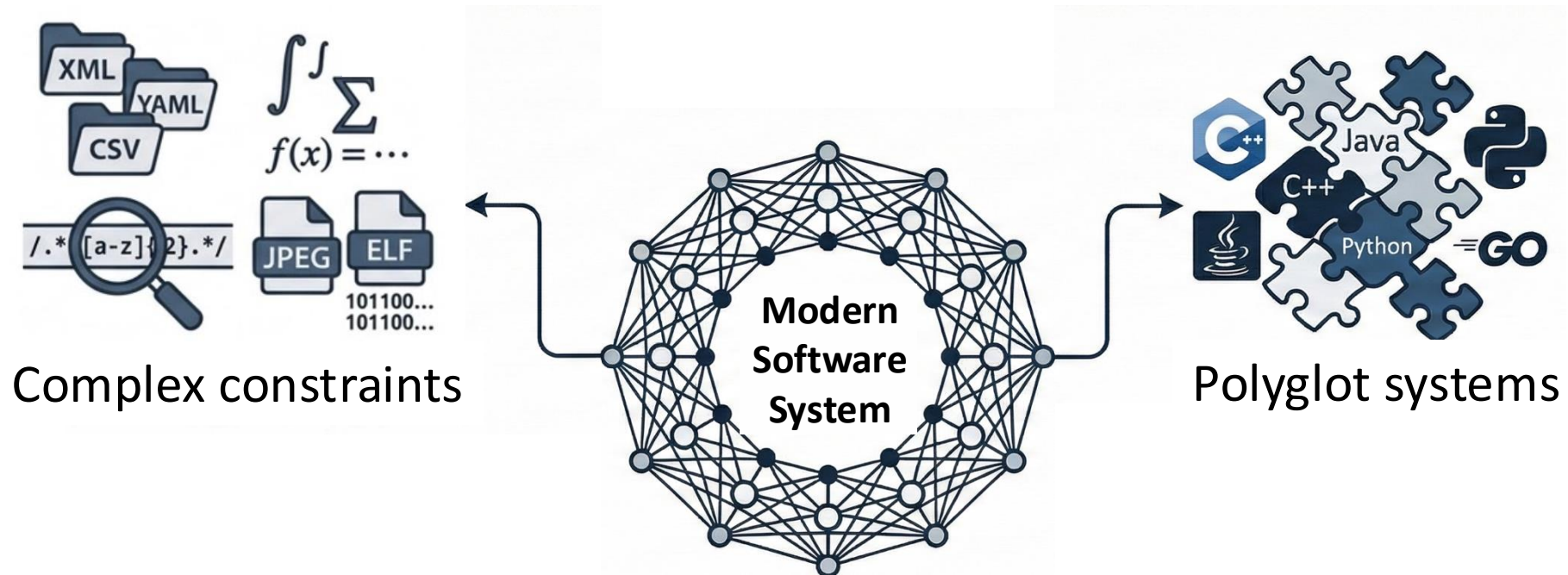
What Does a Real-World Codebase Look Like?



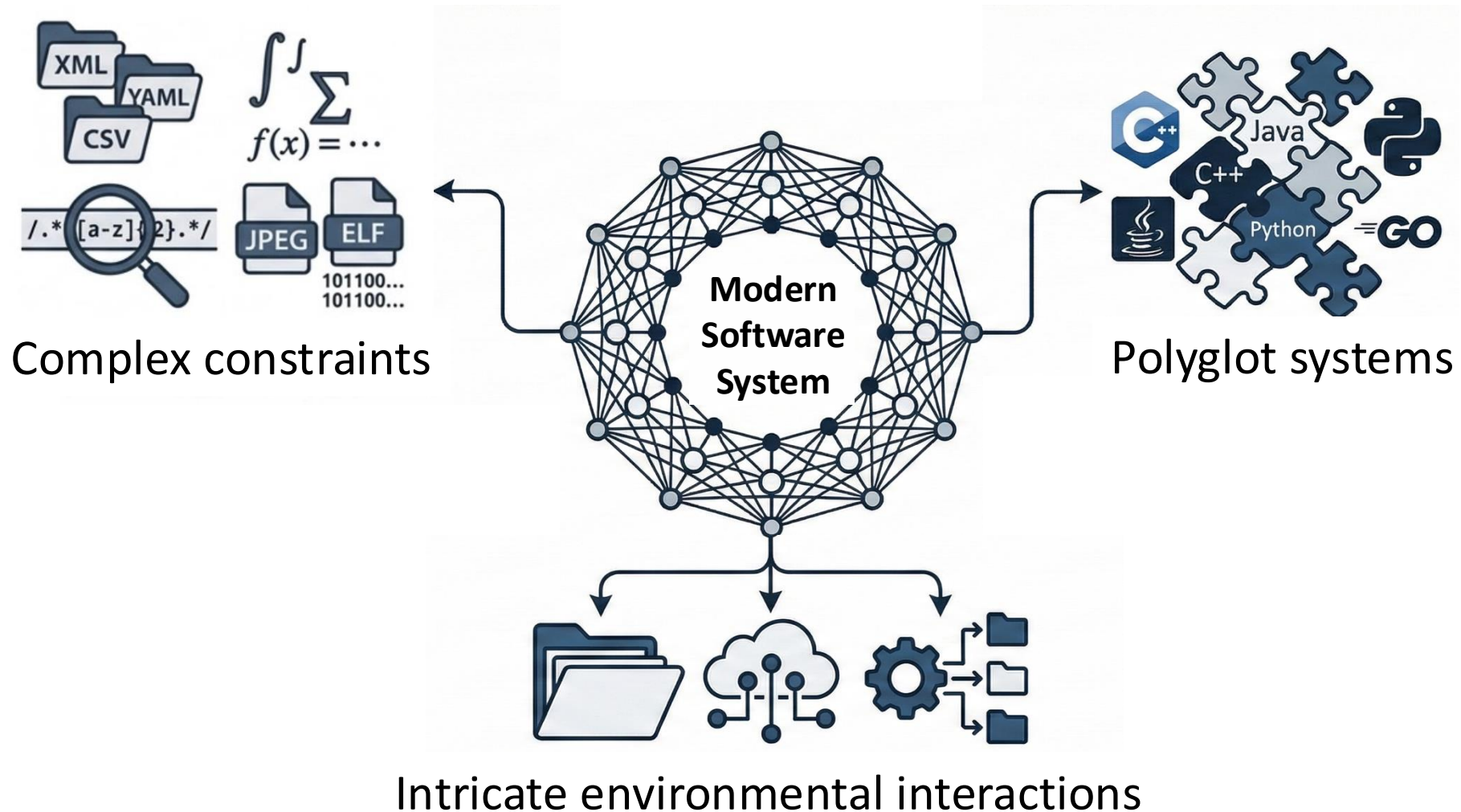
What Does a Real-World Codebase Look Like?



What Does a Real-World Codebase Look Like?



What Does a Real-World Codebase Look Like?



What Does a Real-World Codebase Look Like?

```
1 // assume start < end
2 int count(float start, float end) {
3     int iter_count = 0;    // final answer
4     for (float cur = start; cur != end; iter_count++) {
5         unsigned temp;
6         memcpy(&temp, &cur, sizeof(float));
7         temp++;
8         memcpy(&cur, &temp, sizeof(float));
9     }
10
11     if (iter_count <= 20)
12         printf("BUG triggered!");
13 }
```


What Does a Real-World Codebase Look Like?

```
1 // assume start < end
2 int count(float start, float end) {
3     int iter_count = 0; // final answer
4     for (float cur = start; cur != end; iter_count++) {
5         unsigned temp;
6         memcpy(&temp, &cur, sizeof(float)); temp <- cur
7         temp++; temp <- temp + 1
8         memcpy(&cur, &temp, sizeof(float)); cur <- temp
9     }
10
11     if (iter_count <= 20)
12         printf("BUG triggered!");
13 }
```

Count # of representable floating point values in [start, end)

What Does a Real-World Codebase Look Like?

```
1 // assume start < end
2 int count(float start, float end) {
3     int iter_count = 0;    // final answer
4     for (float cur = start; cur != end; iter_count++) {
5         unsigned temp;
6         memcpy(&temp, &cur, sizeof(float)); temp <- cur
7         temp++;                    temp <- temp + 1
8         memcpy(&cur, &temp, sizeof(float)); cur <- temp
9     }
10
11     if (iter_count <= 20)
12         printf("BUG triggered!");
13 }
```



Count # of representable
floating point values in
[start, end)

Challenge-1: Symbolic Modeling

```
1 // assume start < end
2 int count(float start, float end) {
3     int iter_count = 0;    // final answer
4     for (float cur = start; cur != end; iter_count++) {
5         unsigned temp;
6         memcpy(&temp, &cur, sizeof(float));
7         temp++;
8         memcpy(&cur, &temp, sizeof(float));
9     }
10
11     if (iter_count <= 20)
12         printf("BUG triggered!");
13 }
```

Challenge-1: Symbolic Modeling

```
1 // assume start < end
2 int count(float start, float end) {
3     int iter_count = 0;    // final answer
4     for (float cur = start; cur != end; iter_count++) {
5         unsigned temp;
6         memcpy(&temp, &cur, sizeof(float));
7         temp++;
8         memcpy(&cur, &temp, sizeof(float));
9     }
10
11     if (iter_count <= 20)
12         printf("BUG triggered!");
13 }
```



floating-point



pointer



external function

Challenge-1: Symbolic Modeling

```
1 // assume start < end
2 int count(float start, float end) {
3     int iter_count = 0;    // final answer
4     for (float cur = start; cur != end; iter_count++) {
5         unsigned temp;
6         memcpy(&temp, &cur, sizeof(float));
7         temp++;
8         memcpy(&cur, &temp, sizeof(float));
9     }
10
11     if (iter_count <= 20)
12         printf("BUG triggered!");
13 }
```



floating-point

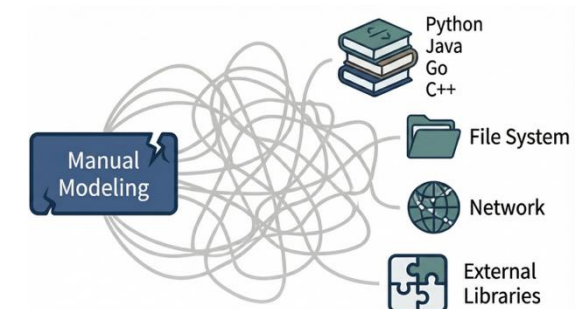


pointer



external function

- Huge manual effort for each *language construct & env*



Challenge-1: Symbolic Modeling

```
1 // assume start < end
2 int count(float start, float end) {
3     int iter_count = 0;    // final answer
4     for (float cur = start; cur != end; iter_count++) {
5         unsigned temp;
6         memcpy(&temp, &cur, sizeof(float));
7         temp++;
8         memcpy(&cur, &temp, sizeof(float));
9     }
10
11     if (iter_count <= 20)
12         printf("BUG triggered!");
13 }
```



floating-point

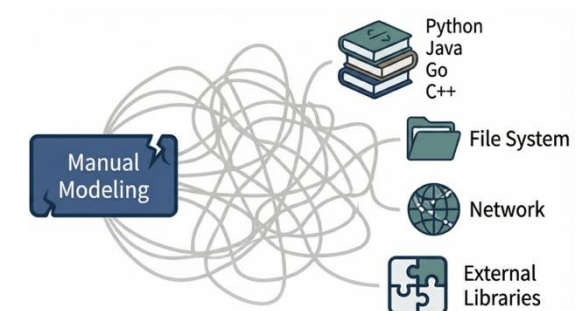


pointer



external function

- Huge manual effort for each *language construct & env*
- Fails on polyglot systems



Challenge-2: Constraint Solving

```
1 // assume start < end
2 int count(float start, float end) {
3     int iter_count = 0;    // final answer
4     for (float cur = start; cur != end; iter_count++) {
5         unsigned temp;
6         memcpy(&temp, &cur, sizeof(float));
7         temp++;
8         memcpy(&cur, &temp, sizeof(float));
9     }
10
11     if (iter_count <= 20)
12         printf("BUG triggered!");
13 }
```

Initial Input

start = 1.0

end = 1.00001

(84 for-loop iterations)

Challenge-2: Constraint Solving

```
1 // assume start < end
2 int count(float start, float end) {
3     int iter_count = 0;    // final answer
4     for (float cur = start; cur != end; iter_count++) {
5         unsigned temp;
6         memcpy(&temp, &cur, sizeof(float));
7         temp++;
8         memcpy(&cur, &temp, sizeof(float));
9     }
10
11     if (iter_count <= 20)
12         printf("BUG triggered!");
13 }
```

Initial Input

start = 1.0
end = 1.00001
(84 for-loop iterations)

Symbolic Store σ

start $\mapsto \alpha$
end $\mapsto \beta$
iter_count $\mapsto i$

Challenge-2: Constraint Solving

```
1 // assume start < end
2 int count(float start, float end) {
3     int iter_count = 0;    // final answer
4     for (float cur = start; cur != end; iter_count++) {
5         unsigned temp;
6         memcpy(&temp, &cur, sizeof(float));
7         temp++;
8         memcpy(&cur, &temp, sizeof(float));
9     }
10
11     if (iter_count <= 20)
12         printf("BUG triggered!");
13 }
```

f

$f^i(\alpha)$

Initial Input

start = 1.0
end = 1.00001
(84 for-loop iterations)

Symbolic Store σ

start $\mapsto \alpha$
end $\mapsto \beta$
iter_count $\mapsto i$

Challenge-2: Constraint Solving

```
1 // assume start < end
2 int count(float start, float end) {
3     int iter_count = 0; // final answer
4     for (float cur = start; cur != end; iter_count++) {
5         unsigned temp;
6         memcpy(&temp, &cur, sizeof(float));
7         temp++;
8         memcpy(&cur, &temp, sizeof(float));
9     }
10
11     if (iter_count <= 20)
12         printf("BUG triggered!");
13 }
```

f

$f^i(\alpha)$

Initial Input

start = 1.0
end = 1.00001
(84 for-loop iterations)

Symbolic Store σ

start $\mapsto \alpha$
end $\mapsto \beta$
iter_count $\mapsto i$

Path Constraints

~100 clauses

$$\bigwedge_{i=0}^{n-1} (f^i(\alpha) \neq \beta) \wedge (f^n(\alpha) = \beta) \wedge (n > 20)$$

Challenge-2: Constraint Solving

```
1 // assume start < end
2 int count(float start, float end) {
3     int iter_count = 0;    // final answer
4     for (float cur = start; cur != end; iter_count++) {
5         unsigned temp;
6         memcpy(&temp, &cur, sizeof(float));
7         temp++;
8         memcpy(&cur, &temp, sizeof(float));
9     }
10
11     if (iter_count <= 20)
12         printf("BUG triggered!");
13 }
```

f

$f^i(\alpha)$

Initial Input

start = 1.0
end = 1.00001
(84 for-loop iterations)

Symbolic Store σ

start $\mapsto \alpha$
end $\mapsto \beta$
iter_count $\mapsto i$

Path Constraints

~100 clauses

$$\bigwedge_{i=0}^{n-1} (f^i(\alpha) \neq \beta) \wedge (f^n(\alpha) = \beta) \wedge (n > 20)$$

- Constraint Representation: *verbose, complex* due to mechanically mirroring each exec. step

Challenge-2: Constraint Solving

```
1 // assume start < end
2 int count(float start, float end) {
3     int iter_count = 0; // final answer
4     for (float cur = start; cur != end; iter_count++) {
5         unsigned temp;
6         memcpy(&temp, &cur, sizeof(float));
7         temp++;
8         memcpy(&cur, &temp, sizeof(float));
9     }
10
11     if (iter_count <= 20)
12         printf("BUG triggered!");
13 }
```

f

$f^i(\alpha)$

Initial Input

start = 1.0
end = 1.00001
(84 for-loop iterations)

Symbolic Store σ

start $\mapsto \alpha$
end $\mapsto \beta$
iter_count $\mapsto i$

Path Constraints

~ 100 clauses

$$\bigwedge_{i=0}^{n-1} (f^i(\alpha) \neq \beta) \wedge (f^n(\alpha) = \beta) \wedge (n > 20)$$

- Constraint Representation: *verbose, complex* due to mechanically mirroring each exec. step
- Hard Theories: string, floating-point (KLEE-Float [Liew et.al 2020])...

Can LLMs Be the Game Changer?

Symbolic Modeling

Constraint Solving

Can LLMs Be the Game Changer?

Symbolic Modeling

Possess vast knowledge *across*



Programming
Languages



Code
Constructs



Program
Environment

Constraint Solving

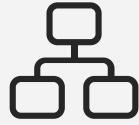
Can LLMs Be the Game Changer?

Symbolic Modeling

Possess vast knowledge *across*



Programming
Languages



Code
Constructs



Program
Environment

Constraint Solving

$$\bigwedge_{i=0}^{n-1} \cancel{(f^i(\alpha) \neq \beta)} \wedge \cancel{(f^n(\alpha) = \beta)} \wedge \cancel{(n > 20)}$$

*find 2 FP numbers w/ ≤ 20
representable numbers in between*



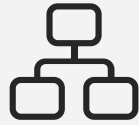
Can LLMs Be the Game Changer?

Symbolic Modeling

Possess vast knowledge *across*



Programming
Languages



Code
Constructs

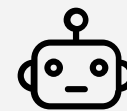


Program
Environment

Constraint Solving

$$\bigwedge_{i=0}^{n-1} (f^i(\alpha) \neq \beta) \wedge (f^n(\alpha) = \beta) \wedge (n > 20)$$

*find 2 FP numbers w/ ≤ 20
representable numbers in between*



Solve



e^x

Can LLMs Be the Game Changer?

Symbolic Modeling

Possess vast knowledge *across*



Programming
Languages



Code
Constructs



Program
Environment

Constraint Solving

$$\bigwedge_{i=0}^{n-1} (f^i(\alpha) \neq \beta) \wedge (f^n(\alpha) = \beta) \wedge (n > 20)$$

*find 2 FP numbers w/ ≤ 20
representable numbers in between*



Solve



e^x

Risk: LLMs are slow, untrustworthy, and can hallucinate

Our Idea

Agentic Concolic Execution

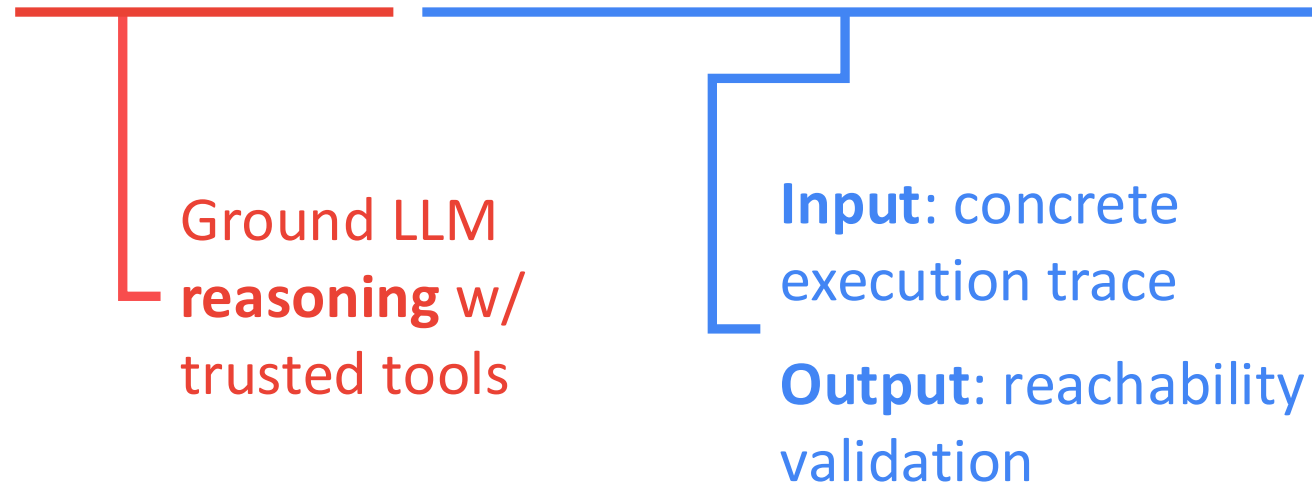
Our Idea

Agentic Concolic Execution

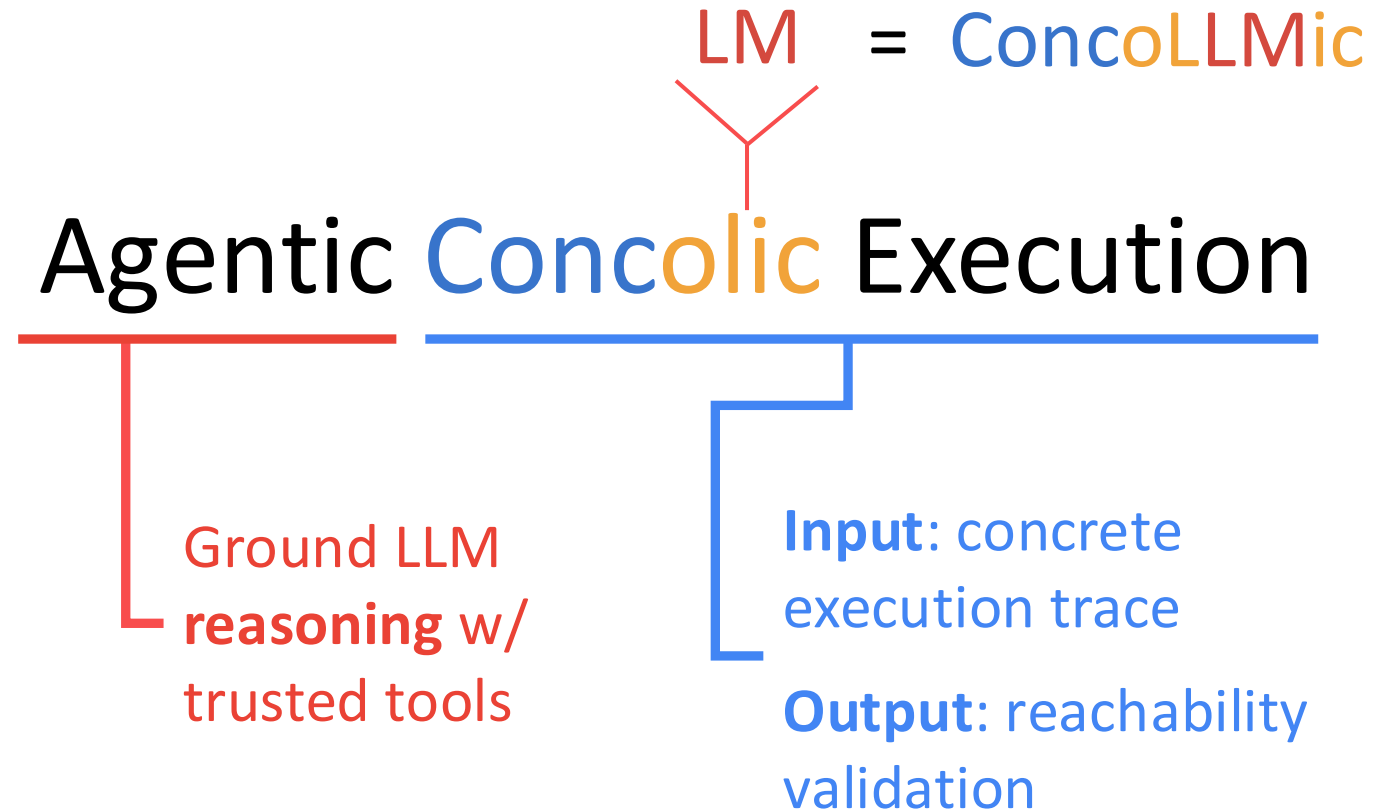


Our Idea

Agentic Concolic Execution







Our Idea



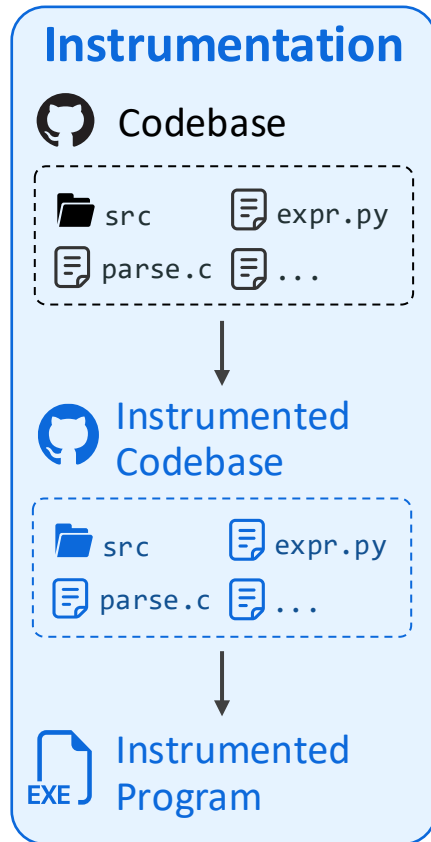
ConcoLLMic Overview

Instrumentation

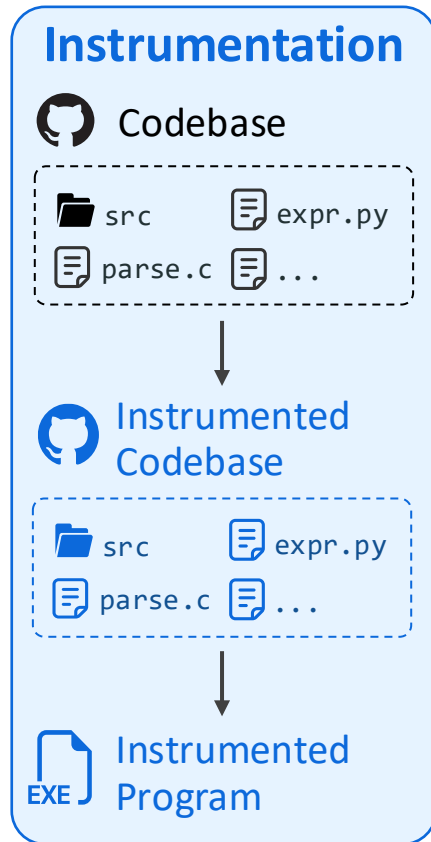
 Codebase

 src  expr.py
 parse.c  ...

ConcoLLMic Overview



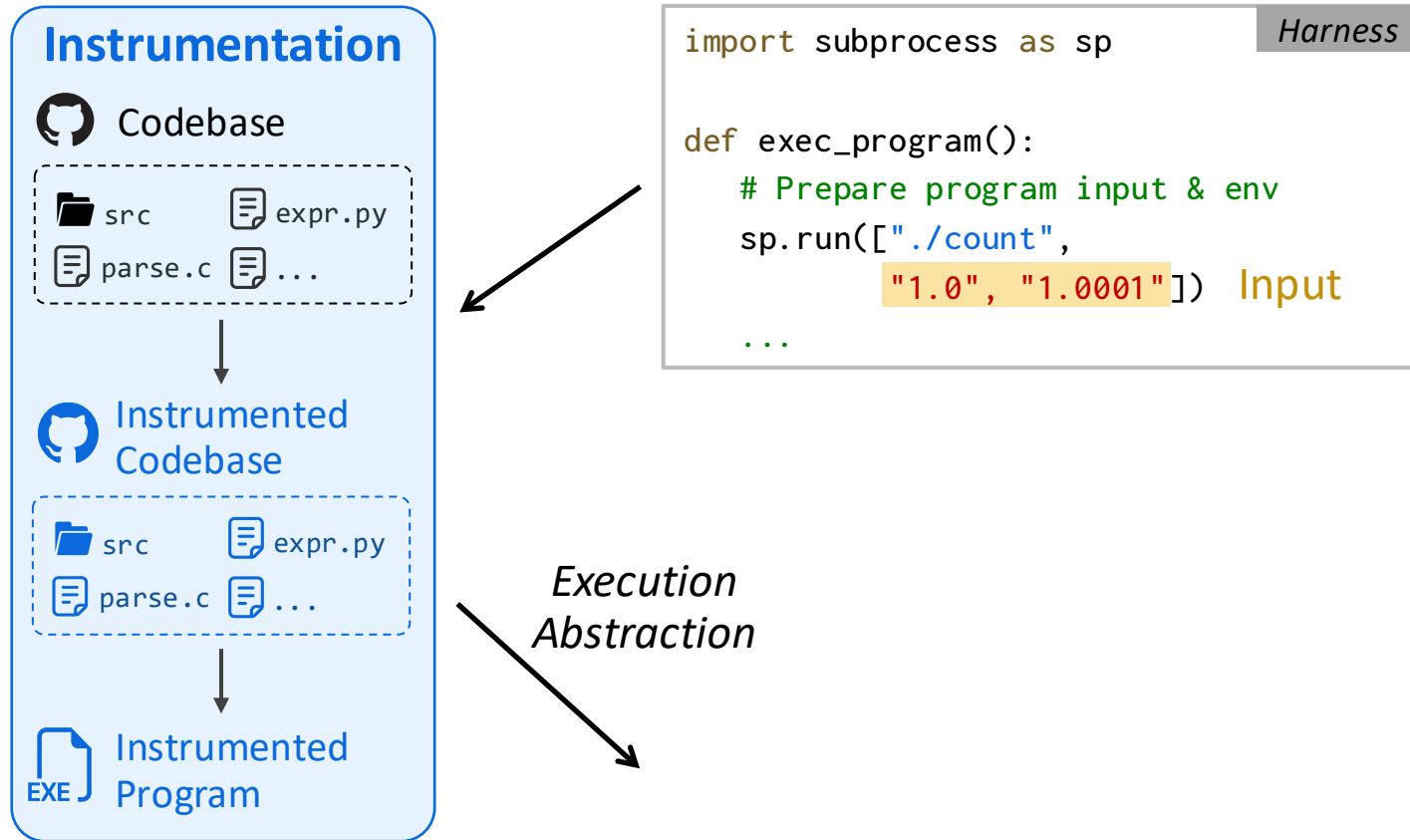
ConcoLLMic Overview



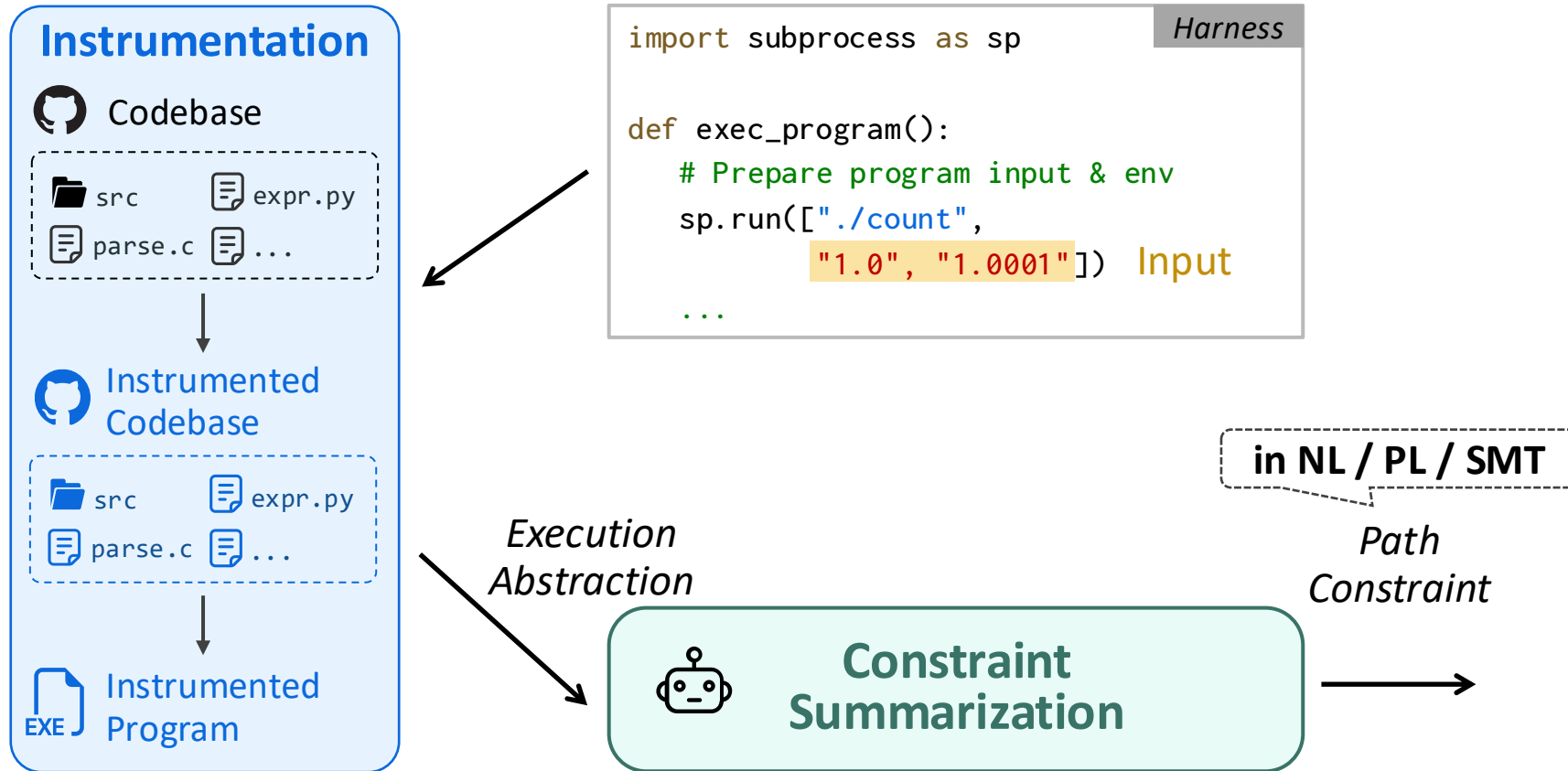
```
import subprocess as sp Harness

def exec_program():
    # Prepare program input & env
    sp.run(["./count",
           "1.0", "1.0001"]) Input
    ...
```

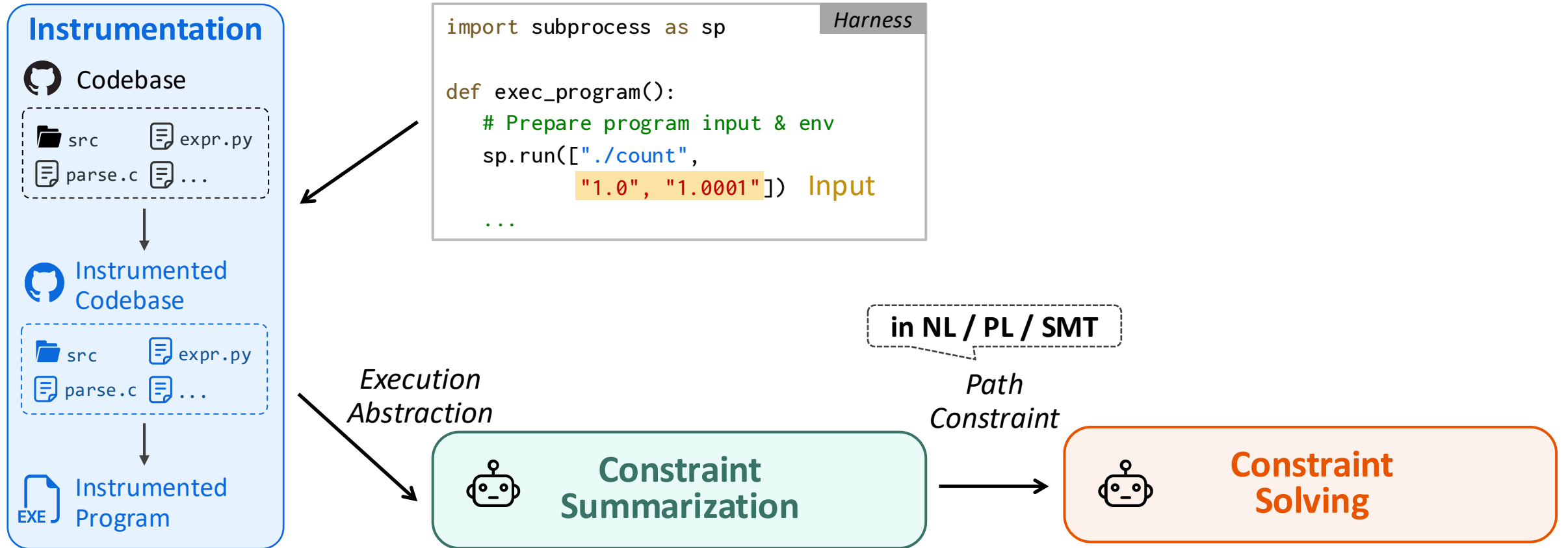
ConcoLLMic Overview



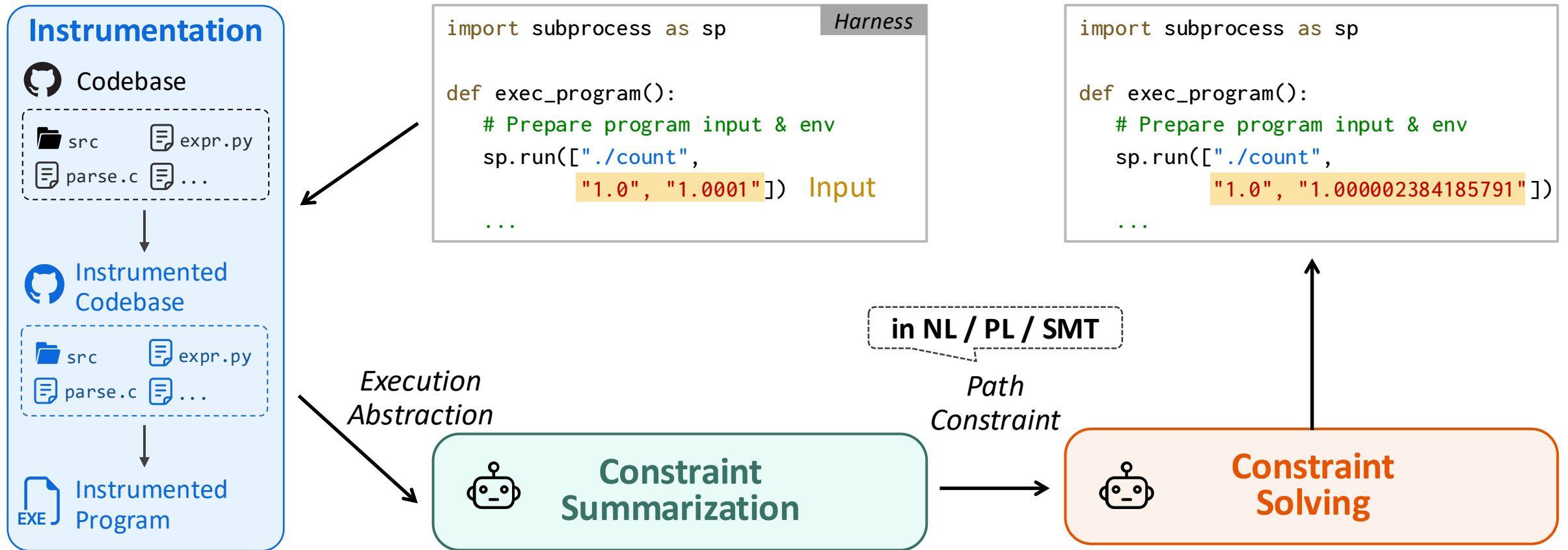
ConcoLLMic Overview



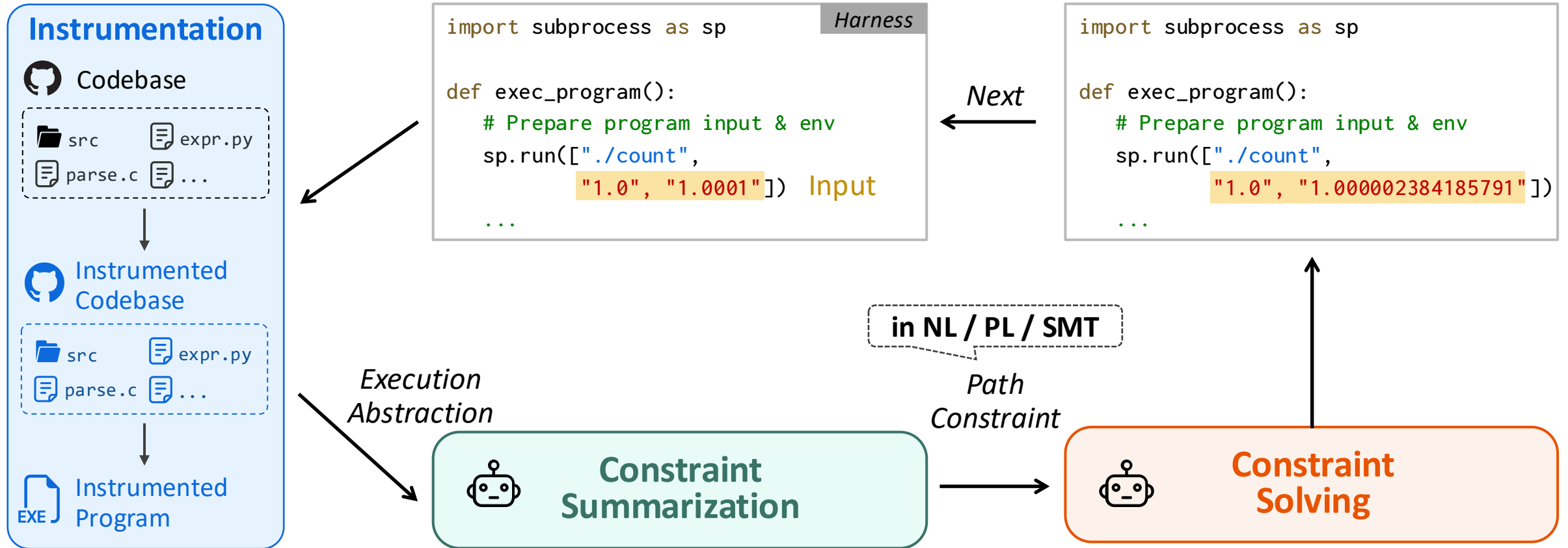
ConcoLLMic Overview



ConcoLLMic Overview



ConcoLLMic Overview



Instrumentation

- Source code level – LLMs' multilingual capability

Instrumentation

- Source code level – LLMs' multilingual capability
- Rule

`enter [func_name] [BB_ID]` →

Original Code Block {

`exit [func_name] [BB_ID]` →

```
int main(int argc , char **argv) {  
    fprintf(stderr, "enter main 1");  
    float start = atof(argv [1]), end = atof(argv [2]);  
    count(start , end);  
    fprintf(stderr, "exit main 1");  
}
```

Instrumentation

- Source code level – LLMs' multilingual capability
- Rule

`enter [func_name] [BB_ID]` →
Original Code Block {
`exit [func_name] [BB_ID]` →

```
int main(int argc , char **argv) {  
    fprintf(stderr, "enter main 1");  
    float start = atof(argv [1]), end = atof(argv [2]);  
    count(start , end);  
    fprintf(stderr, "exit main 1");  
}
```

- Benefits

 Uniform across PLs

Instrumentation

- Source code level – LLMs' multilingual capability
- Rule

`enter [func_name] [BB_ID]` →
Original Code Block {
`exit [func_name] [BB_ID]` →

```
int main(int argc , char **argv) {  
    fprintf(stderr, "enter main 1");  
    float start = atof(argv [1]), end = atof(argv [2]);  
    count(start , end);  
    fprintf(stderr, "exit main 1");  
}
```

- Benefits

 Uniform across PLs

 Modularity

Instrumentation

- Source code level – LLMs' multilingual capability
- Rule

`enter [func_name] [BB_ID]` →
Original Code Block {
`exit [func_name] [BB_ID]` →

```
int main(int argc , char **argv) {  
    fprintf(stderr, "enter main 1");  
    float start = atof(argv [1]), end = atof(argv [2]);  
    count(start , end);  
    fprintf(stderr, "exit main 1");  
}
```

- Benefits

 Uniform across PLs

 Modularity

 Simple

Instrumentation

- Source code level – LLMs’ multilingual capability
- Rule

`enter [func_name] [BB_ID]` →
Original Code Block {
`exit [func_name] [BB_ID]` →

```
int main(int argc , char **argv) {  
    fprintf(stderr, "enter main 1");  
    float start = atof(argv [1]), end = atof(argv [2]);  
    count(start , end);  
    fprintf(stderr, "exit main 1");  
}
```

- Benefits

 Uniform across PLs

 Modularity

 Simple

 Essential Exec. Info. & Cov. Bookkeeping

Execution Abstraction

①



Func
Call Chain

[src/main.c] `main` => [src/count.c] `count`



Execution order

Execution Abstraction

①



Func
Call Chain

```
[src/main.c] main => [src/count.c] count
```



Execution order

②



Executed
Lines

count.c (13 lines total)

```
// Omitted Imports, Definitions & Variable Declarations
1 // assume start < end
2 int count(float start, float end) {
3     int iteration_count = 0;    // final answer
4     for (float cur = start; cur != end; iteration_count++) {
5         unsigned temp;
6         memcpy(&temp, &cur, sizeof(float));
7         temp++;
8         memcpy(&cur, &temp, sizeof(float));
9     }
10
11     if (iteration_count <= 20)
12         // Unexecuted line 12 removed. Line cov: 0/1
13 }
```

Execution Abstraction

①  Func Call Chain [src/main.c] main => [src/count.c] count

← Execution order

②  Executed Lines count.c (13 lines total)

← Global context information

```
// Omitted Imports, Definitions & Variable Declarations
1 // assume start < end
2 int count(float start, float end) {
3     int iteration_count = 0;    // final answer
4     for (float cur = start; cur != end; iteration_count++) {
5         unsigned temp;
6         memcpy(&temp, &cur, sizeof(float));
7         temp++;
8         memcpy(&cur, &temp, sizeof(float));
9     }
10
11     if (iteration_count <= 20)
12         // Unexecuted line 12 removed. Line cov: 0/1
13 }
```

← Executed code



← Untaken branch condition
Cov info. for importance measuring
(agentic exploration)

Constraint Summarization Agent

Harness

```
subprocess.run(["./count", "1.0", "1.0001"])
```

Execution Abstraction

- ①  Func Call Chain [src/main.c] main => [src/count.c] count
- ②  Executed Lines count.c (13 lines total)

```
// Omitted Imports, Definitions & Variable Declarations
1 // assume start < end
2 int count(float start, float end) {
3     int iteration_count = 0; // final answer
4     for (float cur = start; cur != end; iteration_count++) {
5         unsigned temp;
6         memcpy(&temp, &cur, sizeof(float));
7         temp++;
8         memcpy(&cur, &temp, sizeof(float));
9     }
10
11     if (iteration_count <= 20)
12         // Unexecuted line 12 removed. Line cov: 0/1
13 }
```

Given : ① *harness* ② *execution abstraction...*

Task : select target branch & generate path constraints

Tools : <CODEREQUEST, THINK, CHOOSEBRANCH, SUMMARIZE>



Constraint Summarization Agent

Harness

```
subprocess.run(["./count", "1.0", "1.0001"])
```

Execution Abstraction

①  Func Call Chain [src/main.c] main => [src/count.c] count

②  Executed Lines count.c (13 lines total)

```
// Omitted Imports, Definitions & Variable Declarations
1 // assume start < end
2 int count(float start, float end) {
3     int iteration_count = 0; // final answer
4     for (float cur = start; cur != end; iteration_count++) {
5         unsigned temp;
6         memcpy(&temp, &cur, sizeof(float));
7         temp++;
8         memcpy(&cur, &temp, sizeof(float));
9     }
10
11     if (iteration_count <= 20)
12         // Unexecuted line 12 removed. Line cov: 0/1
13 }
```

Given : ① *harness* ② *execution abstraction...*

Task : select target branch & generate path constraints

Tools : <CODEREQUEST, THINK, CHOOSEBRANCH, SUMMARIZE>



{CODEREQUEST} count.c, 12-12



|12|Uncovered|: printf("BUG triggered!");



Constraint Summarization Agent

Harness


```
subprocess.run(["./count", "1.0", "1.0001"])
```

Execution Abstraction

①  Func Call Chain [src/main.c] main => [src/count.c] count

②  Executed Lines count.c (13 lines total)

```
// Omitted Imports, Definitions & Variable Declarations
1 // assume start < end
2 int count(float start, float end) {
3     int iteration_count = 0; // final answer
4     for (float cur = start; cur != end; iteration_count++) {
5         unsigned temp;
6         memcpy(&temp, &cur, sizeof(float));
7         temp++;
8         memcpy(&cur, &temp, sizeof(float));
9     }
10
11     if (iteration_count <= 20)
12         // Unexecuted line 12 removed. Line cov: 0/1
13 }
```



Given : ① *harness* ② *execution abstraction...*

Task : select target branch & generate path constraints

Tools : <CODEREQUEST, THINK, CHOOSEBRANCH, SUMMARIZE>



{CODEREQUEST} count.c, 12-12



|12|Uncovered|: printf("BUG triggered!");



{CHOOSEBRANCH}

[target_branch] if (iteration_count <= 20) => True

[rationale] It represents a key behavior & has 0% cov.

[lines_to_cover] count.c:12-12



Your selection is recorded!



Constraint Summarization Agent

Harness

```
subprocess.run(["./count", "1.0", "1.0001"])
```

Execution Abstraction

- ①  Func Call Chain [src/main.c] main => [src/count.c] count
- ②  Executed Lines count.c (13 lines total)

```
// Omitted Imports, Definitions & Variable Declarations
1 // assume start < end
2 int count(float start, float end) {
3     int iteration_count = 0; // final answer
4     for (float cur = start; cur != end; iteration_count++) {
5         unsigned temp;
6         memcpy(&temp, &cur, sizeof(float));
7         temp++;
8         memcpy(&cur, &temp, sizeof(float));
9     }
10
11     if (iteration_count <= 20)
12         // Unexecuted line 12 removed. Line cov: 0/1
13 }
```



Given : ① *harness* ② *execution abstraction...*

Task : select target branch & generate path constraints

Tools : <CODEREQUEST, THINK, CHOOSEBRANCH, SUMMARIZE>



{CODEREQUEST} count.c, 12-12



|12|Uncovered|: printf("BUG triggered!");



{CHOOSEBRANCH}

[target_branch] if (iteration_count <= 20) => True

[rationale] It represents a key behavior & has 0% cov.

[lines_to_cover] count.c:12-12



Your selection is recorded!



{SUMMARIZE} [path_constraint]

a) 2 CLI arguments: start < end

b) start, end are valid floating-points

c) # IEEE-754 floating-point \in [start, end) \leq 20



Constraint Solving Agent

Harness

```
subprocess.run(["./count", "1.0", "1.0001"])
```

Path Constraint (for target branch)

- a) 2 CLI arguments: start < end
- b) start, end are valid floating-points
- c) # IEEE-754 floating-point \in [start, end) ≤ 20

Given : ① *harness* ② *new path constraints* ...

Task : solve path constraint for a new input...

Tools : <EXECUTE_CODE, QUERY_SMT, THINK, GENERATE_TEST>



Constraint Solving Agent

Harness

```
subprocess.run(["./count", "1.0", "1.0001"])
```

Path Constraint (for target branch)

- a) 2 CLI arguments: start < end
- b) start, end are valid floating-points
- c) # IEEE-754 floating-point \in [start, end) \leq 20

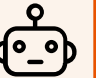
Given : ① *harness* ② *new path constraints* ...

Task : solve path constraint for a new input...

Tools : <EXECUTE_CODE, QUERY_SMT, THINK, GENERATE_TEST>



{THINK} Original range [1.0, 1.00001) contains > 20 representable floats.
Let's fix start=1.0 and calculate end



Constraint Solving Agent

Harness

```
subprocess.run(["./count", "1.0", "1.0001"])
```

Path Constraint (for target branch)

- a) 2 CLI arguments: start < end
- b) start, end are valid floating-points
- c) # IEEE-754 floating-point \in [start, end) ≤ 20

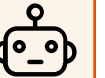
Given : ① *harness* ② *new path constraints* ...

Task : solve path constraint for a new input...

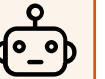
Tools : <EXECUTE_CODE, QUERY_SMT, THINK, GENERATE_TEST>



{THINK} Original range [1.0, 1.00001) contains > 20 representable floats.
Let's fix start=1.0 and calculate end



{THINK} For float around 1.0, the smallest increment is $\sim 2^{-23} \approx 1.19e^{-7}$
So the range is $\sim 20 \times 1.19e^{-7} \approx 2.38e^{-6}$... Let me try to verify ...



Your reasoning is recorded!



Constraint Solving Agent

Harness

```
subprocess.run(["./count", "1.0", "1.0001"])
```

Path Constraint (for target branch)

- a) 2 CLI arguments: start < end
- b) start, end are valid floating-points
- c) # IEEE-754 floating-point \in [start, end) ≤ 20

Given : ① *harness* ② *new path constraints* ...

Task : solve path constraint for a new input...

Tools : <EXECUTE_CODE, QUERY_SMT, THINK, GENERATE_TEST>



{THINK} Original range [1.0, 1.00001) contains > 20 representable floats.
Let's fix start=1.0 and calculate end



{THINK} For float around 1.0, the smallest increment is $\sim 2^{-23} \approx 1.19e^{-7}$
So the range is $\sim 20 \times 1.19e^{-7} \approx 2.38e^{-6}$... Let me try to verify ...



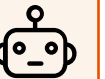
Your reasoning is recorded!



{EXECUTE_CODE}

```
def next_float(f): ... # calculate next float

current = 1.0
for i in range(19):
    current = next_float(current)
print(next_float(current)); # the 20th value
```



Exec. Result: stdout = 1.000002384185791



Constraint Solving Agent

Harness

```
subprocess.run(["./count", "1.0", "1.0001"])
```

Path Constraint (for target branch)

- a) 2 CLI arguments: start < end
- b) start, end are valid floating-points
- c) # IEEE-754 floating-point \in [start, end) ≤ 20

Given : ① *harness* ② *new path constraints* ...

Task : solve path constraint for a new input...

Tools : <EXECUTECODE, QUERYSMT, THINK, GENERATETEST>



{THINK} Original range [1.0, 1.00001) contains > 20 representable floats.
Let's fix start=1.0 and calculate end



{THINK} For float around 1.0, the smallest increment is $\sim 2^{-23} \approx 1.19e^{-7}$
So the range is $\sim 20 \times 1.19e^{-7} \approx 2.38e^{-6}$... Let me try to verify ...



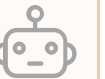
Your reasoning is recorded!



{EXECUTECODE}

```
def next_float(f): ... # calculate next float

current = 1.0
for i in range(19):
    current = next_float(current)
print(next_float(current)); # the 20th value
```

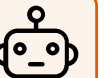


Exec. Result: stdout = 1.000002384185791



{GENERATETEST} [*is_satisfiable*] ✓

```
subprocess.run(["./count", "1.0", "1.000002384185791"])
```



Constraint Solving Agent

Harness

```
subprocess.run(["./count", "1.0", "1.0001"])
```

Path Constraint (for target branch)

- a) 2 CLI arguments: start < end
- b) start, end are valid floating-points
- c) # IEEE-754 floating-point \in [start, end) ≤ 20

{CHOOSEBRANCH}

```
[target_branch] if (iteration_count <= 20) => True
```

```
[rationale] ...
```

```
[lines_to_cover] count.c:12-12
```

Validation

Given : ① *harness* ② *new path constraints* ...

Task : solve path constraint for a new input...

Tools : <EXECUTECODE, QUERYSMT, THINK, GENERATETEST>



{THINK} Original range [1.0, 1.00001) contains > 20 representable floats.
Let's fix start=1.0 and calculate end



{THINK} For float around 1.0, the smallest increment is $\sim 2^{-23} \approx 1.19e^{-7}$
So the range is $\sim 20 \times 1.19e^{-7} \approx 2.38e^{-6}$... Let me try to verify ...



Your reasoning is recorded!



{EXECUTECODE}

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

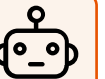


Exec. Result: stdout = 1.000002384185791



{GENERATETEST} [is_satisfiable] ✓

```
subprocess.run(["./count", "1.0", "1.000002384185791"])
```



Evaluation Setup

LLM: *claude-sonnet-3.7* (200K context window)

Benchmark:

8 × Monolingual



4 × Polyglot



26 × Floating-Point



Baselines: State-of-the-Art DSEs & Fuzzer



- & Two variants
- Pending
- Float

SymCC

SymSan



Code Coverage – Monolingual

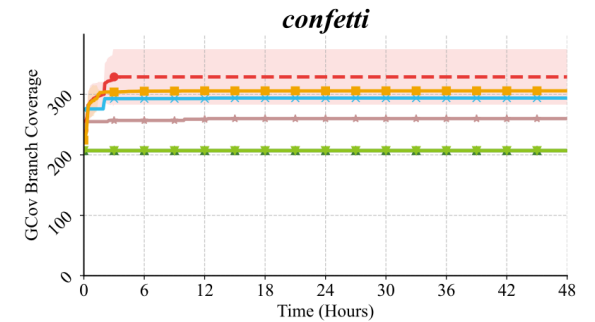
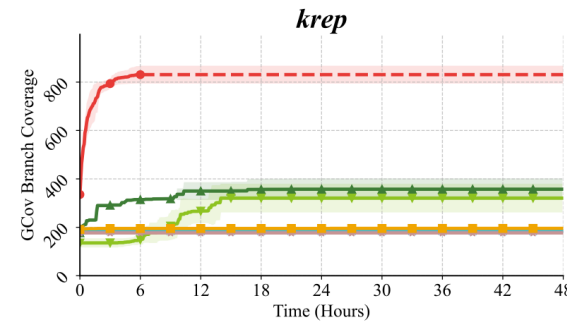
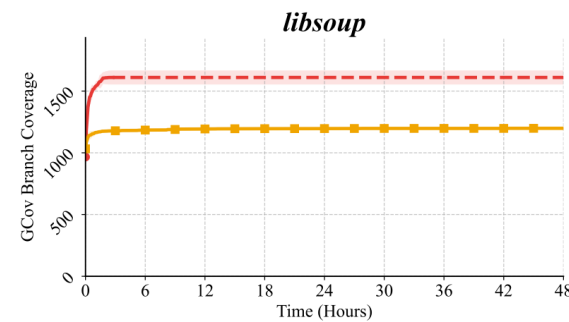
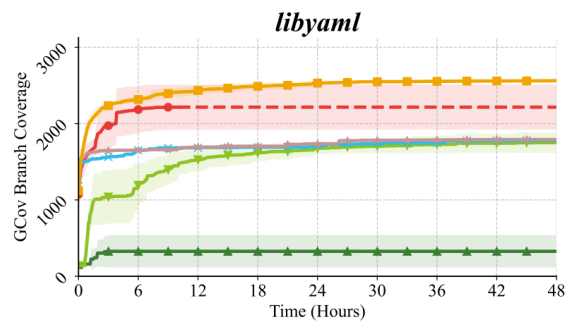
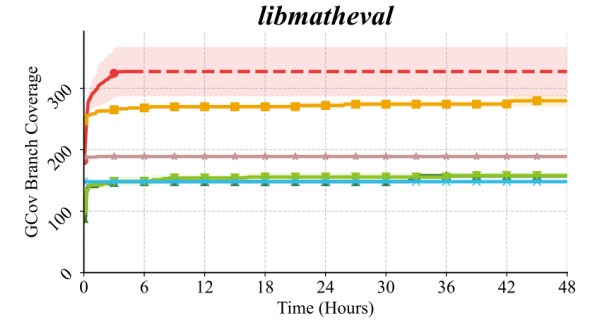
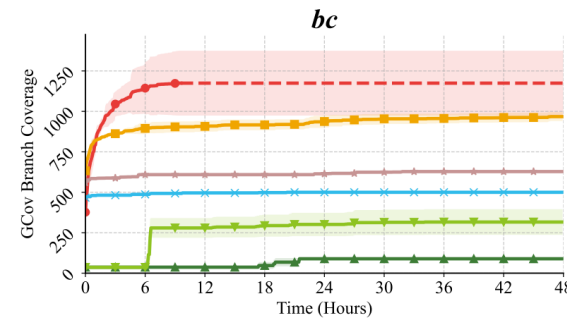
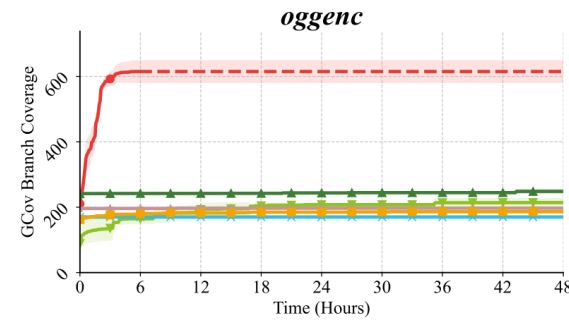
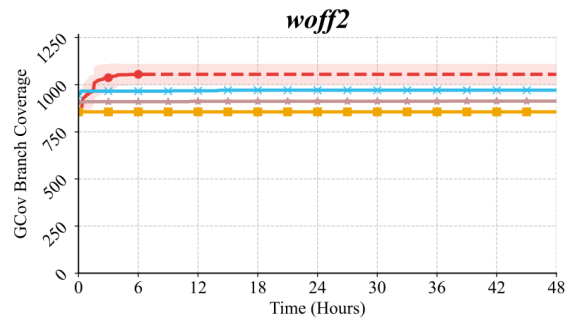
KLEE
+233%

KLEE-Pending
+135%

SymCC
+130%

SymSan
+115%

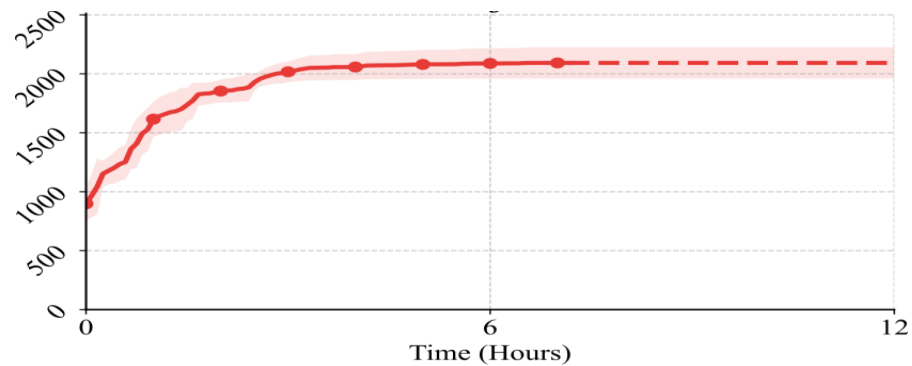
AFL++
+81%



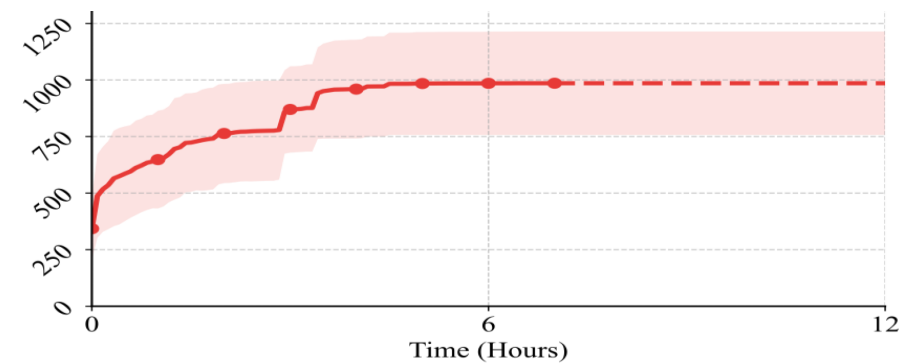
Code Coverage – Polyglot

- Successfully tested and grew coverage on polyglot systems

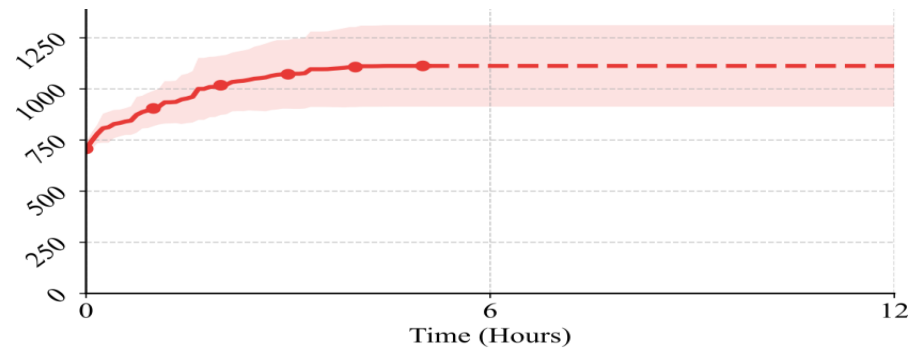
 **ultrajson**



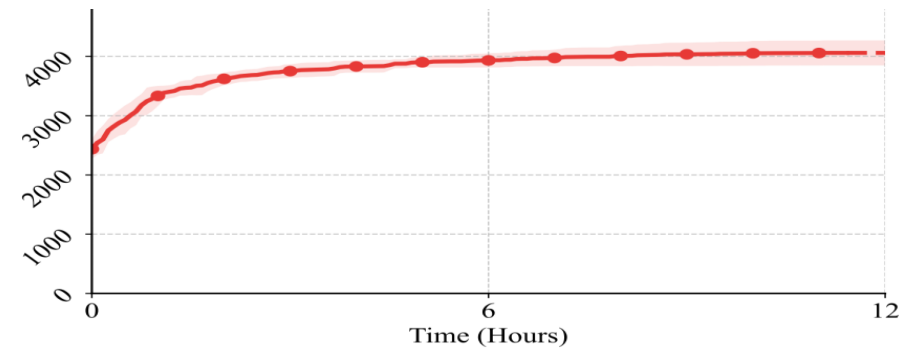
 **jansi**






 **py4j**



 **protobuf**

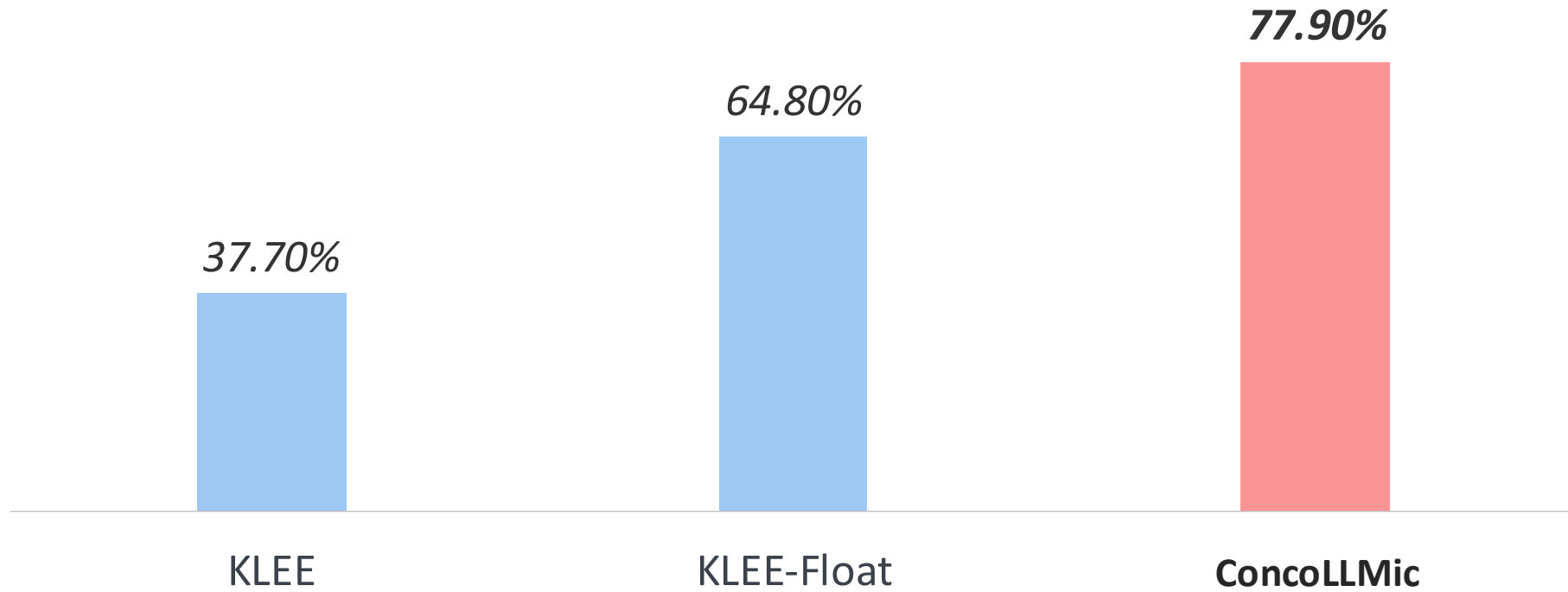


Languages

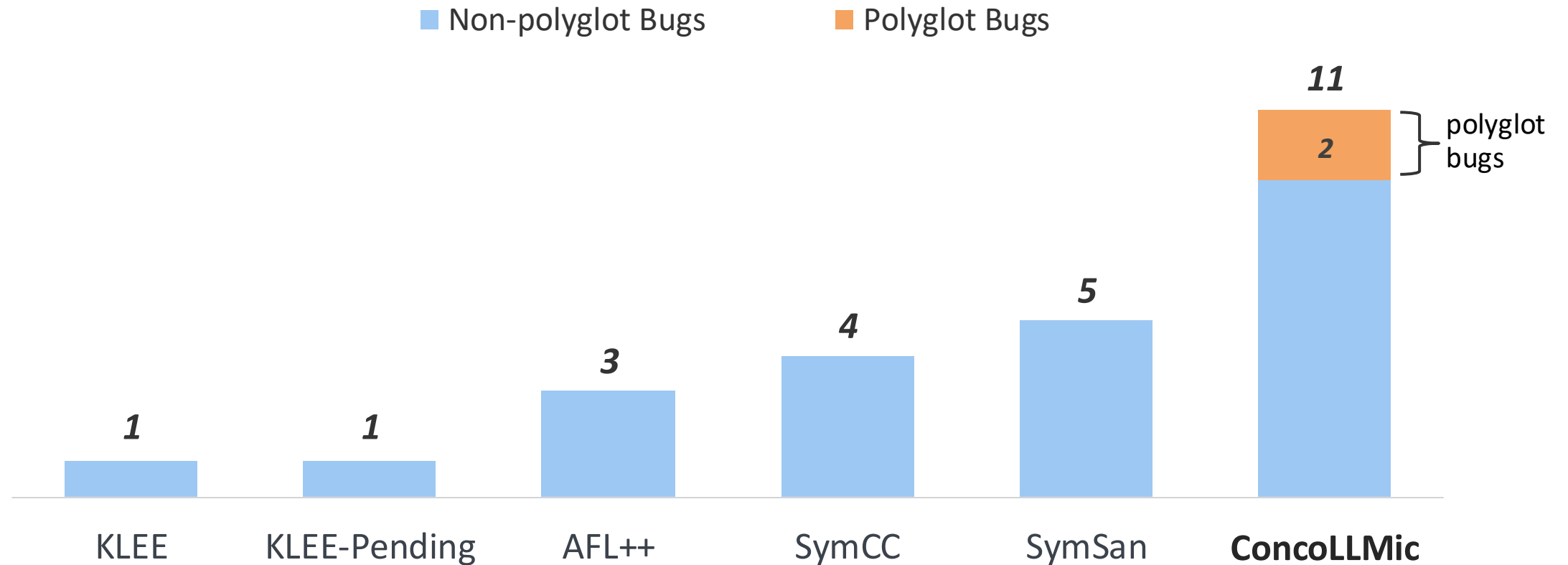
-  Python
-  C
-  Java
-  C++
-  Go

Code Coverage – Floating-Point Benchmark

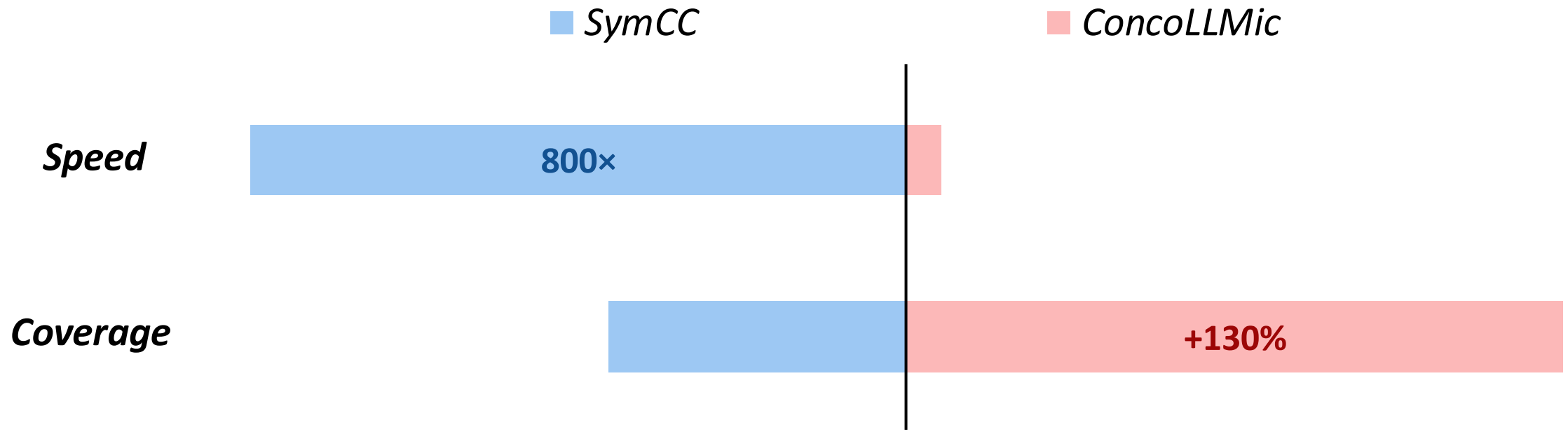
Branch Cov Ratio



Bug Detection



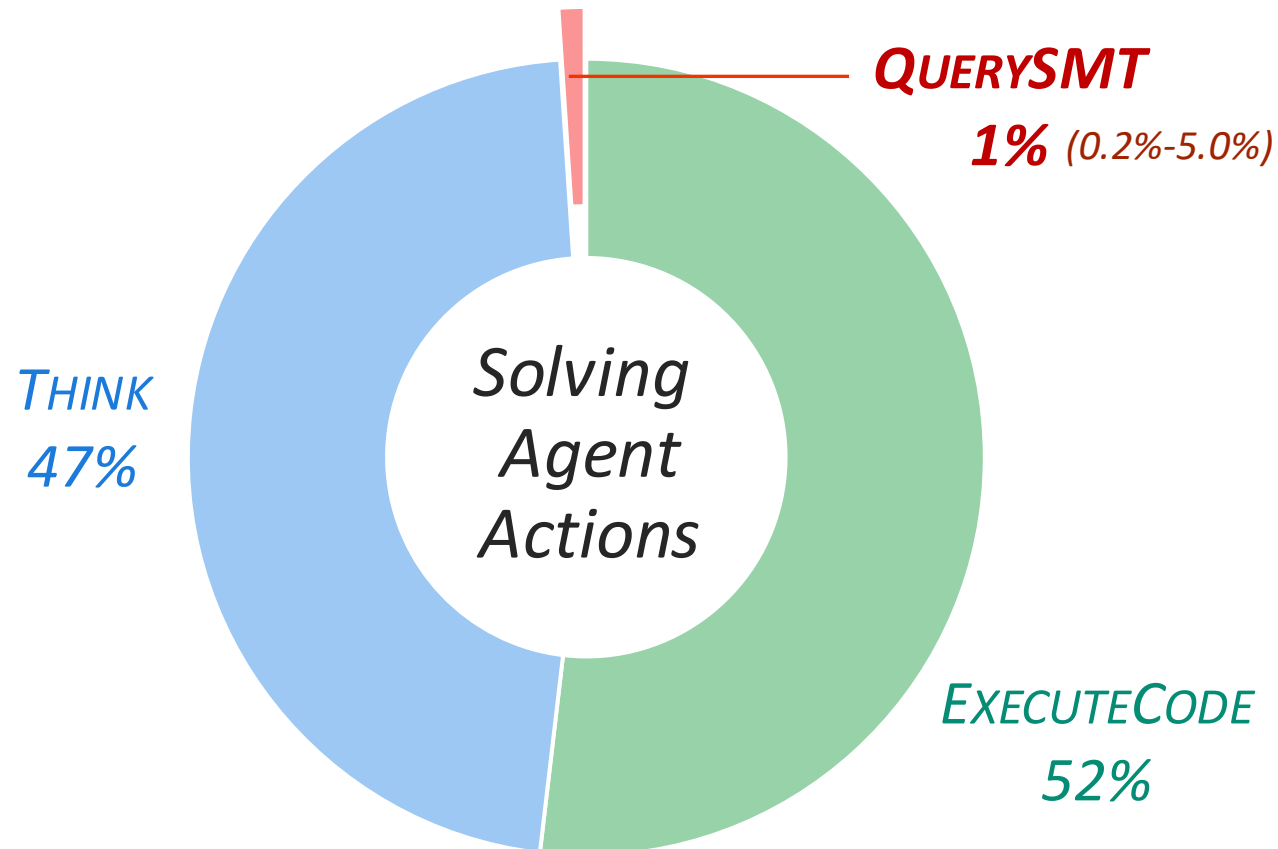
Findings 1: Slower Pace, Superior Results



- LLM latency & multi-step agentic reasoning

Findings 2: Reasoning Beyond Formal Solvers

- ✓ Agentic approach in constraint accumulation & inference w/o relying on SMT



Summary

Can LLMs Be the Game Changer?

Symbolic Modeling

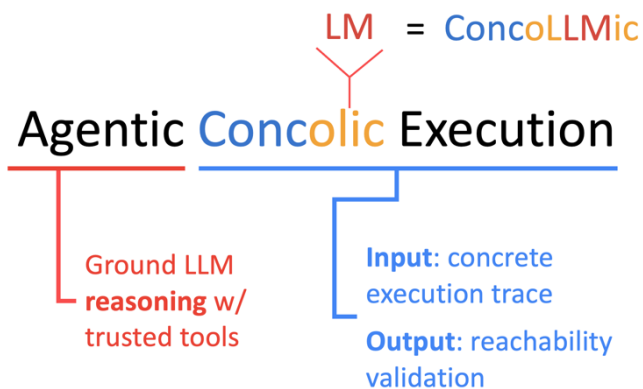
Possess vast knowledge *across*

Programming Languages Code Constructs Program Environment

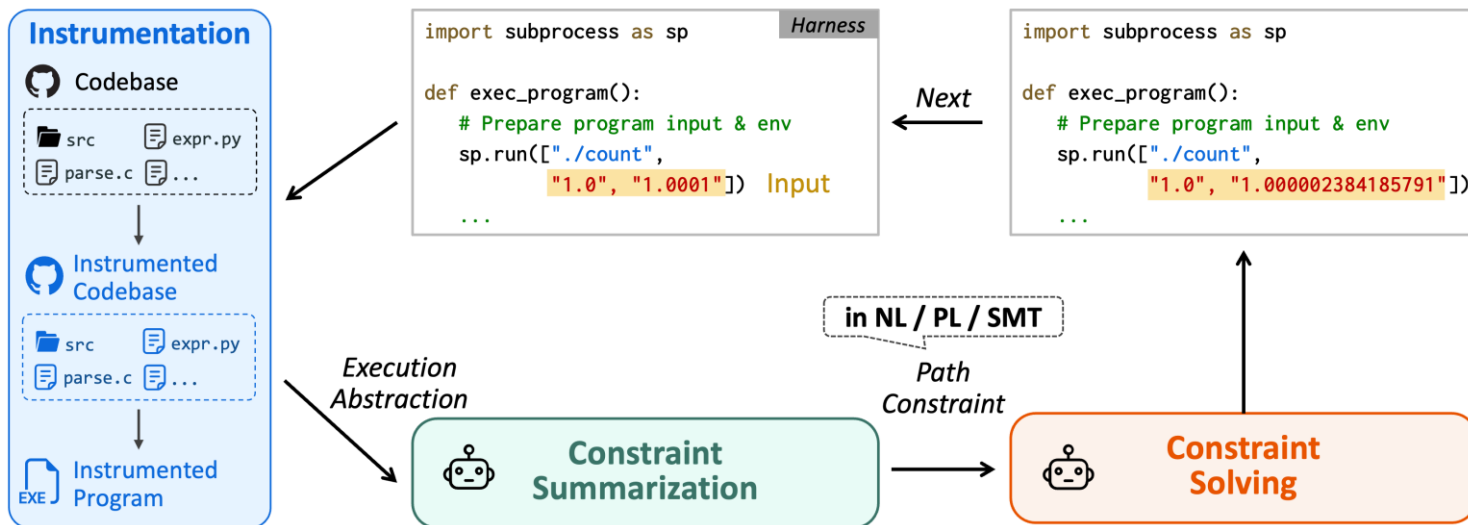
Constraint Solving

$$\bigwedge_{i=0}^{n-1} (f^i(\alpha) \neq \beta) \wedge (f^n(\alpha) = \beta) \wedge (n > 20)$$

find 2 FP numbers w/ ≤ 20 representable numbers in between



Risk: LLMs are slow, untrustworthy, and can hallucinate



Paper



Code

