# SmartFL: Semantics Based Probabilistic Fault Localization

Yiqian Wu, Zeng Muhan, Yujie Liu, Yi Yin,
Xin Zhang, **Yingfei Xiong**, and Lu Zhang

北京大學
PEKING UNIVERSITY

# Status Update 1

- Our group have been working on solving competitive programming problems for the last four years
  - Without using neural networks
  - Talked about this in Meetings 67 and 69
- Our plan: to win a silver medal in the national competition in 2025
- Sep, 2024: o1-IOI won a silver medal in IOI
- Dec, 2024: o3 won a gold medal in IOI, Top 200 in Codeforce
- We surrendered.
- Switched to use LLMs to solve SE problems.
  - Program repair / generation / analysis / verification

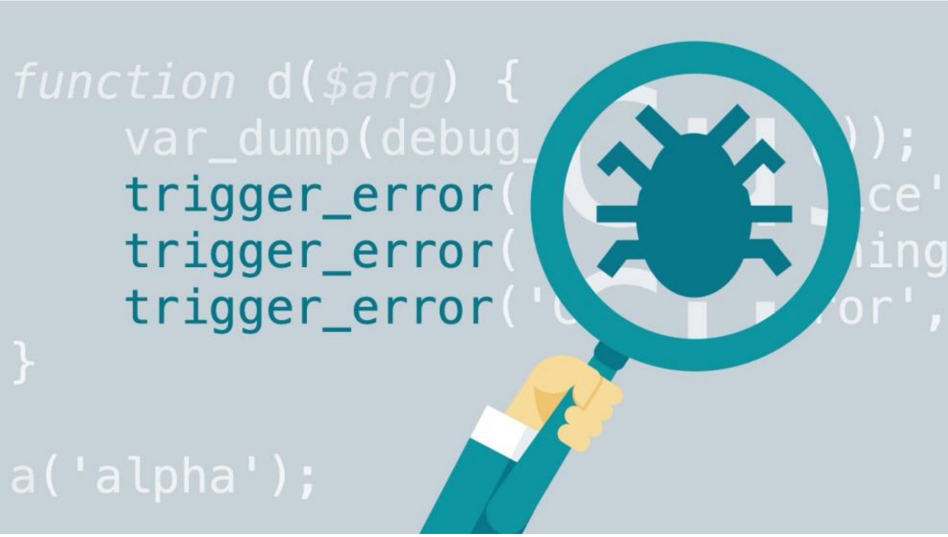
# Status Update 2

- The grammar-based code representation for neural models
  - which I introduced in
    - IFIP meeting 68
    - Dagstuhl on program synthesis and repair
  - was evaluated in 1.3B and 1.5B models in Kuaishou
  - 4.2-17.7 percentage points higher than token-based representation in accuracy
- The preprint will be released soon

# Fault Localization

**Input**

- A faulty program
- A set of tests with at least one failing test




**Output**

- The suspiciousness score of each program element

# Coverage-based Fault Localization

## Idea

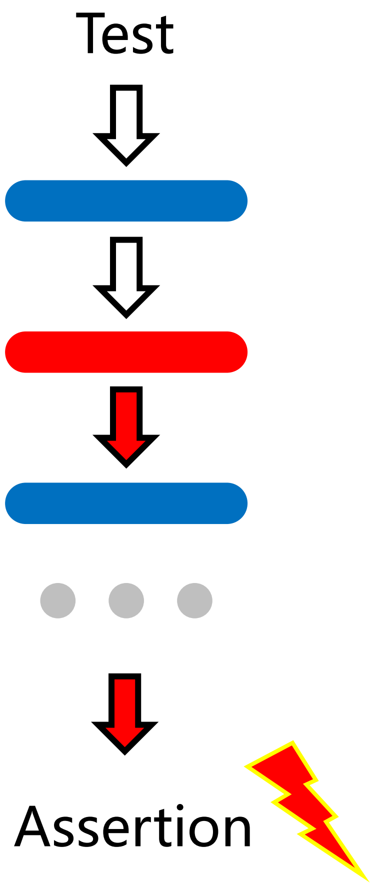
- An element covered more by failing tests rather than passing tests is more likely to be faulty

## Spectrum-based fault localization

- Count the number of passing/failing tests covering the element
- Calculate the suspiciousness score of each program element

| | | coverage of 8 tests | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ | $t_7$ | $t_8$ |
| | foo ( ) { | | | | | | | | |
| | int $x, y, z, m, ret$ ; | | | | | | | | |
| $s_1$ | read $(x, y, z)$; | • | • | • | • | • | • | • | • |
| $s_2$ | $x = x/y$; // correct: $x = x * y$ | • | • | • | • | • | • | • | • |
| $s_3$ | $m = x + y$; | • | • | • | • | • | • | • | • |
| $s_4$ | if $(x > 1)$ { | • | • | • | • | • | • | • | • |
| $s_5$ | $m = x - 2$; | | • | • | • | • | • | • | • |
| $s_6$ | $x = x * y$; | | • | • | • | | | • | • |
| $s_7$ | $z = 2 * y$; | | • | • | • | | | • | • |
| | } | | | | | | | | |
| $s_8$ | if $(m > 0)$ { | • | • | • | • | • | • | • | • |
| $s_9$ | $ret = x - z$; | • | | • | | • | | • | • |
| | } else { | | | | | | | | |
| $s_{10}$ | $ret = y - z$; | | • | | • | | • | | |
| | } | | | | | | | | |
| $s_{11}$ | return $ret$; | • | • | • | • | • | • | • | • |
| | } results | F | F | P | P | P | P | F | P |

# How a Buggy Element Causes the Failure




**PIE model**

1. Coverage
2. Errors in program state
3. Propagation of errors
4. Capture

Coverage-based methods only consider 1

2,3,4 correspond to the semantics of the program

**How to model the latter three conditions?**

# Existing Research

**Mutation-based fault localization**

- Generates many mutations on each element
- Watches whether the test result changes
- If a change is more likely to change the results in failing tests, and less likely in passing tests, the corresponding statement is likely to be faulty

**Angelic debugging** (by Satish et al.)

- Uses symbolic analysis
- Modify the result of an expression
- If an expression can be modified to reverse the results of failing tests while maintaining the results of passing tests, the expression is considered more likely to be faulty

**Full Program Semantics** → **Heavy**

# Our Approach: SmartFL

**How a fault leads to the current test results**

- The probability of each statement introducing an error
- The probability of each statement propagating an error

**Abstracts the full program semantics**

- Introduce random variables to represent the correctness of each statement and runtime variable value
- Transform program semantics into probabilistic constraints between random variables

**Efficient Probabilistic Modeling of Program Semantics** ✓

# Running Example

```
1  public class CondTest {
2      public static int foo(int a) {
3          if (a <= 2) { // buggy, should be a < 2
4              a = a + 1;
5          }
6          return a;
7      }
8
9      @Test
10     void pass() {
11         assertEquals(2, foo(1));
12     }
13
14     @Test
15     void fail() {
16         assertEquals(2, foo(2));
17     }
18 }
```

$P(S_3 = 1) = 0.5 \quad P(S_4 = 1) = 0.5 \qquad P(V_{p,2} = 1) = 1$

$P(V_{p,3} = 1 \mid S_3 = 1 \land V_{p,2} = 1) = 1$

$P(V_{p,3} = 1 \mid S_3 = 0 \lor V_{p,2} = 0) = 0.5$

$P(V_{p,4} = 1 \mid S_4 = 1 \land V_{p,2} = 1 \land V_{p,3} = 1) = 1$

$P(V_{p,4} = 1 \mid S_4 = 0 \lor V_{p,2} = 0 \lor V_{p,3} = 0) = 0.01$

$P(S_3 = 0 \mid V_{p,4} = 1 \land V_{4,f} = 0) \approx 0.707$

$P(S_4 = 0 \mid V_{p,4} = 1 \land V_{4,f} = 0) \approx 0.270$

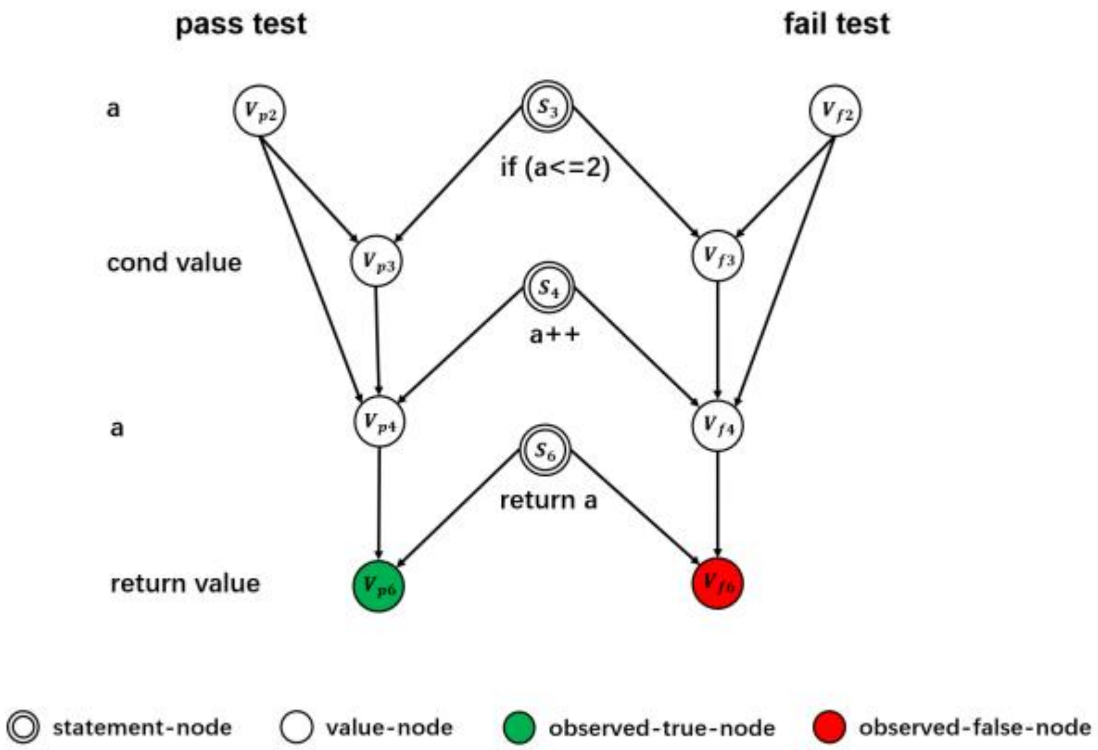
Bernoulli random variables:

$S_i$: representing the correctness of line I

$V_{t,i}$: representing whether the value defined at line i is correct for test $t \in \{p, f\}$

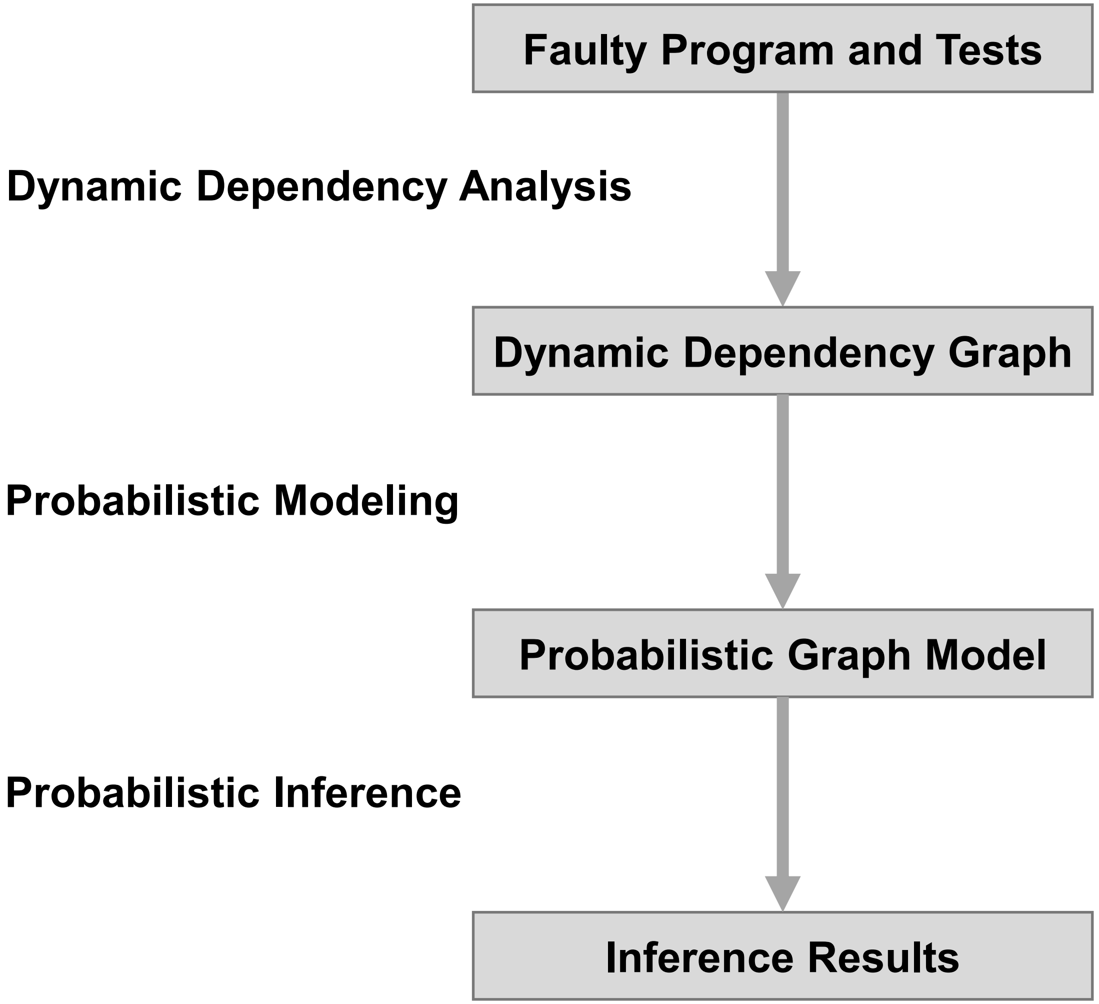
# Bayesian Network

```java
public class CondTest {
    public static int foo(int a) {
        if (a <= 2) { // buggy, should be a < 2
            a = a + 1;
        }
        return a;
    }

    @Test
    void pass() {
        assertEquals(2, foo(1));
    }

    @Test
    void fail() {
        assertEquals(2, foo(2));
    }
}
```



# SmartFL Workflow [ICSE22]

**Steps**



# Results [ICSE22]

- Results on 4 projects from Defects4j 1.0
  - Outperforming both SBFL and MBFL
  - Time consumption between SBFL and MBFL

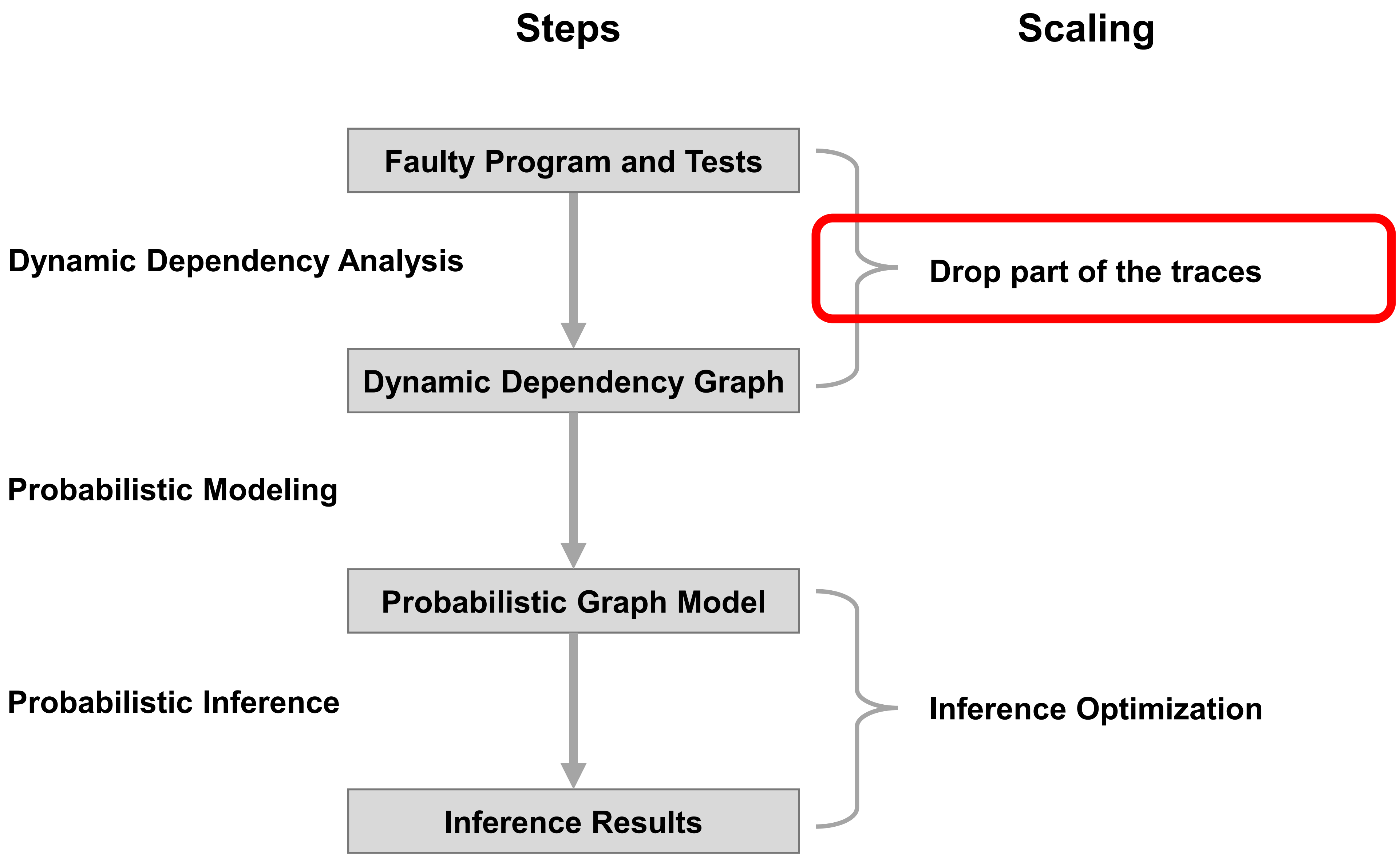
| Project | Technique | Top-1 | Top-3 | Top-5 | Top-10 |
|---|---|---|---|---|---|
| Total | Ochiai | 11(5%) | 64(29%) | 86(39%) | **118(53%)** |
| | DStar | 12(5%) | 65(29%) | 86(39%) | 117(53%) |
| | Metallaxis | 21(9%) | 69(31%) | 89(40%) | 111(50%) |
| | MUSE | 17(8%) | 35(15%) | 45(20%) | 50(23%) |
| | SmartFL | **47(21%)** | **80(36%)** | **97(44%)** | **118(53%)** |

| Technique | Average | Lang | Math | Chart | Time |
|---|---|---|---|---|---|
| Ochiai | 64 | 26 | 86 | 44 | 85 |
| DStar | 64 | 26 | 86 | 44 | 85 |
| Metallaxis | 3500 | 270 | 3000 | 5400 | 12000 |
| MUSE | 3500 | 270 | 3000 | 5400 | 12000 |
| SmartFL | 210 | 51 | 140 | 280 | 830 |

# Bottleneck on Performance [In Submission]

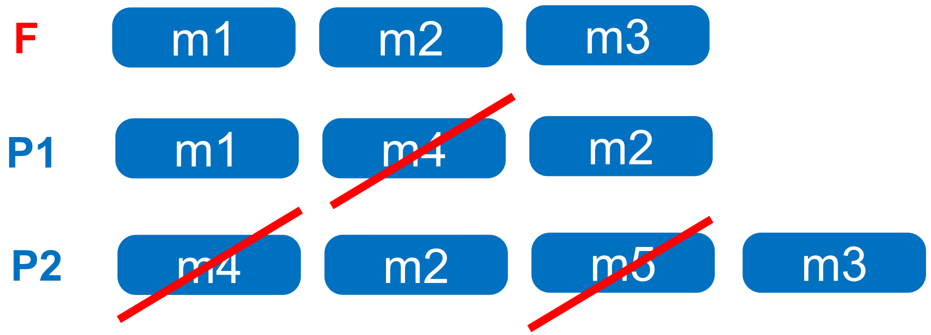
- Traces are too long
- Probabilistic models would be too large for existing probabilistic
  - Drop part of the traces
  - Optimize the probabilistic inference algorithm

# SmartFL Workflow [In Submission]

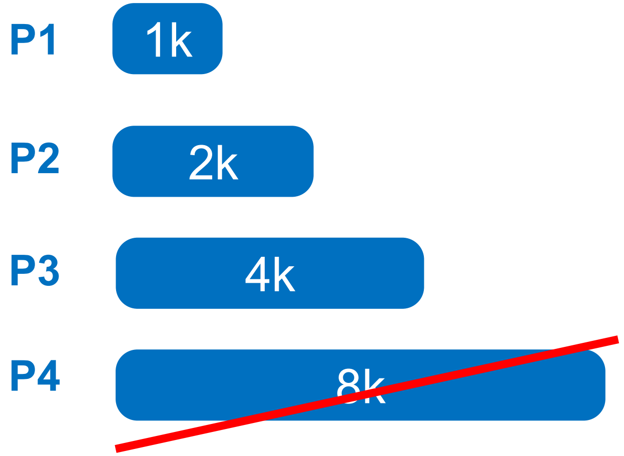


# Drop part of traces

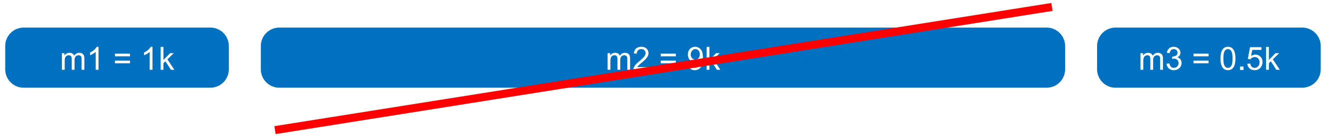
Methods not covered by the failing test




Drop large passing tests



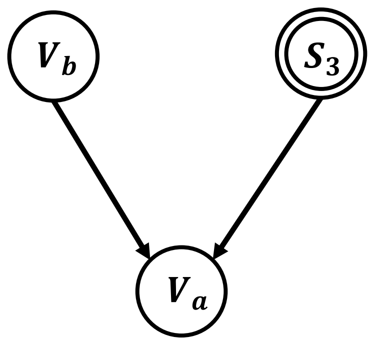

Drop large methods in failing tests



# Reducing Redundant Methods

## Atomic statement

S: a = untraced(b);

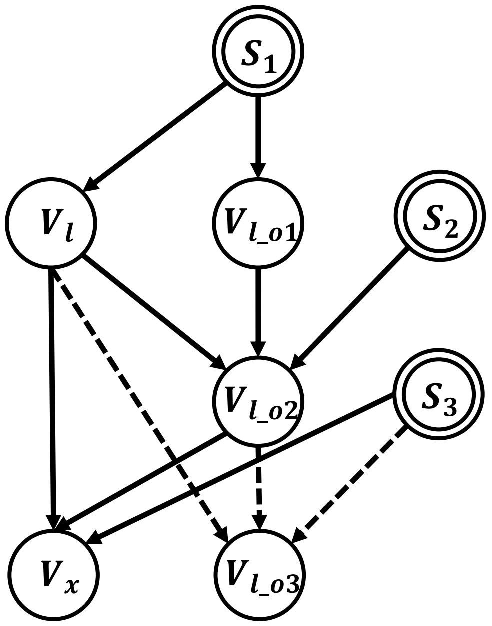



## Side effects

S1: List l = new ArrayList();

S2: l.add(1); // untraced lib method

S3: x = l.size(); // untraced lib method



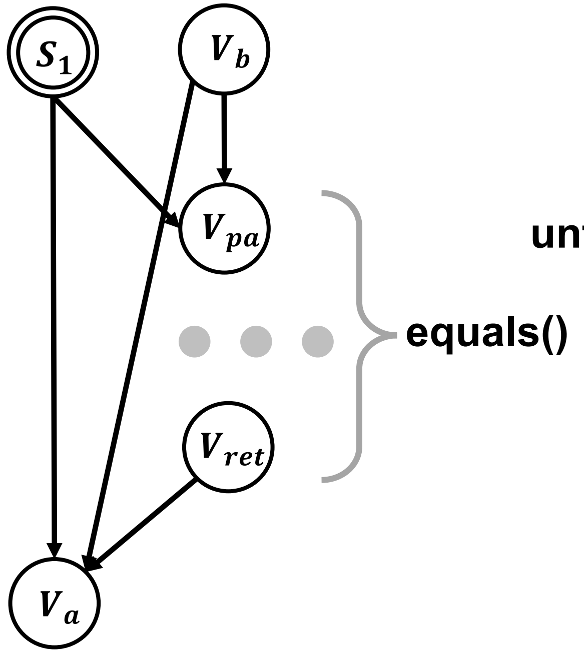

Assuming access to and only to the object referenced

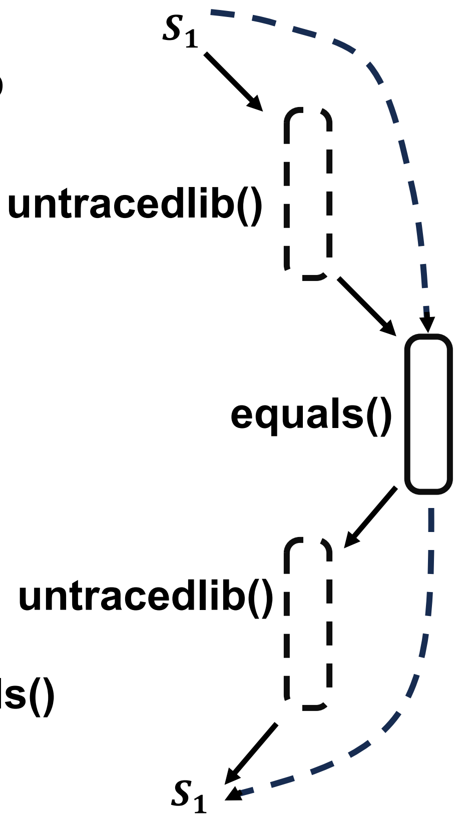
## Virtual call edge

S1: a = untracedlib(b);

overloaded "equals()" of b

"equals()" return



# Reducing Redundant Loops

**Same graph structure**

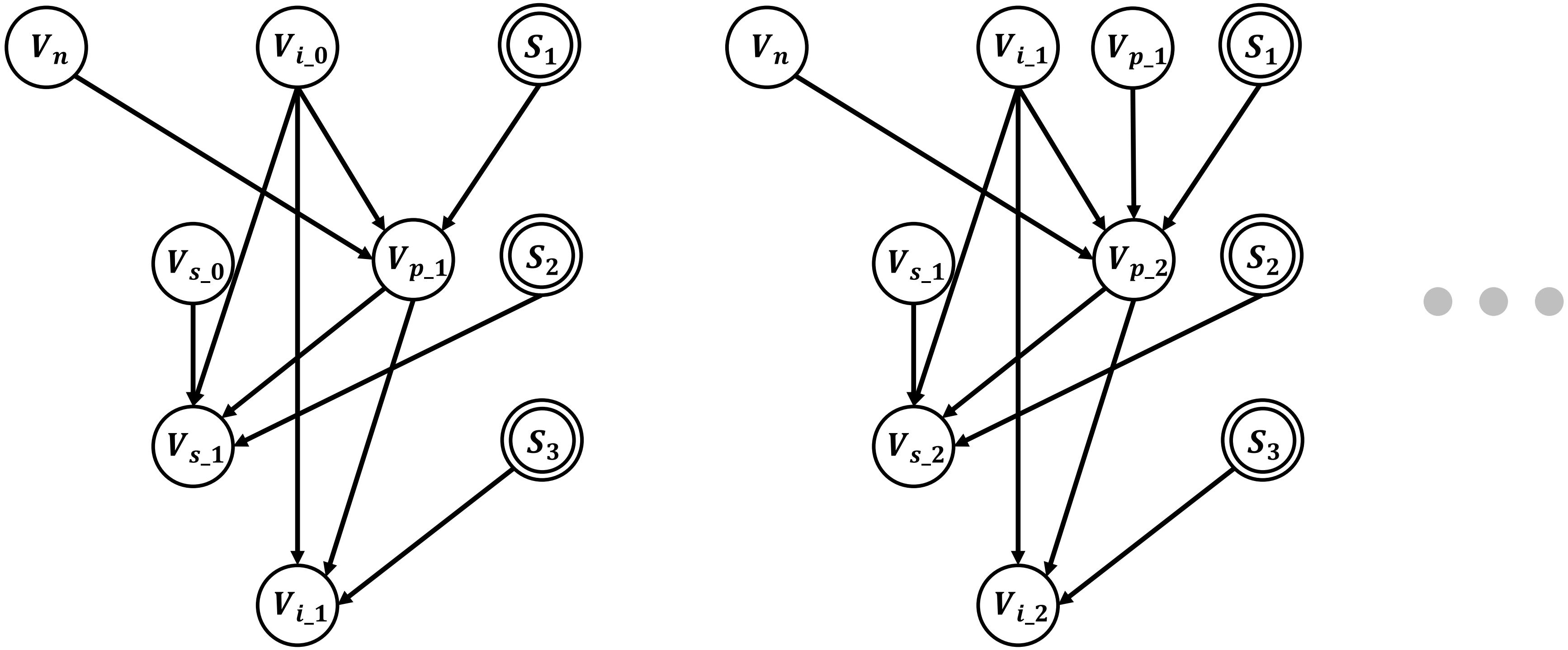
**Loop**

S1: while(i<n){
S2: s = s+i;
S3: i = i +1;
}

**Sequence**

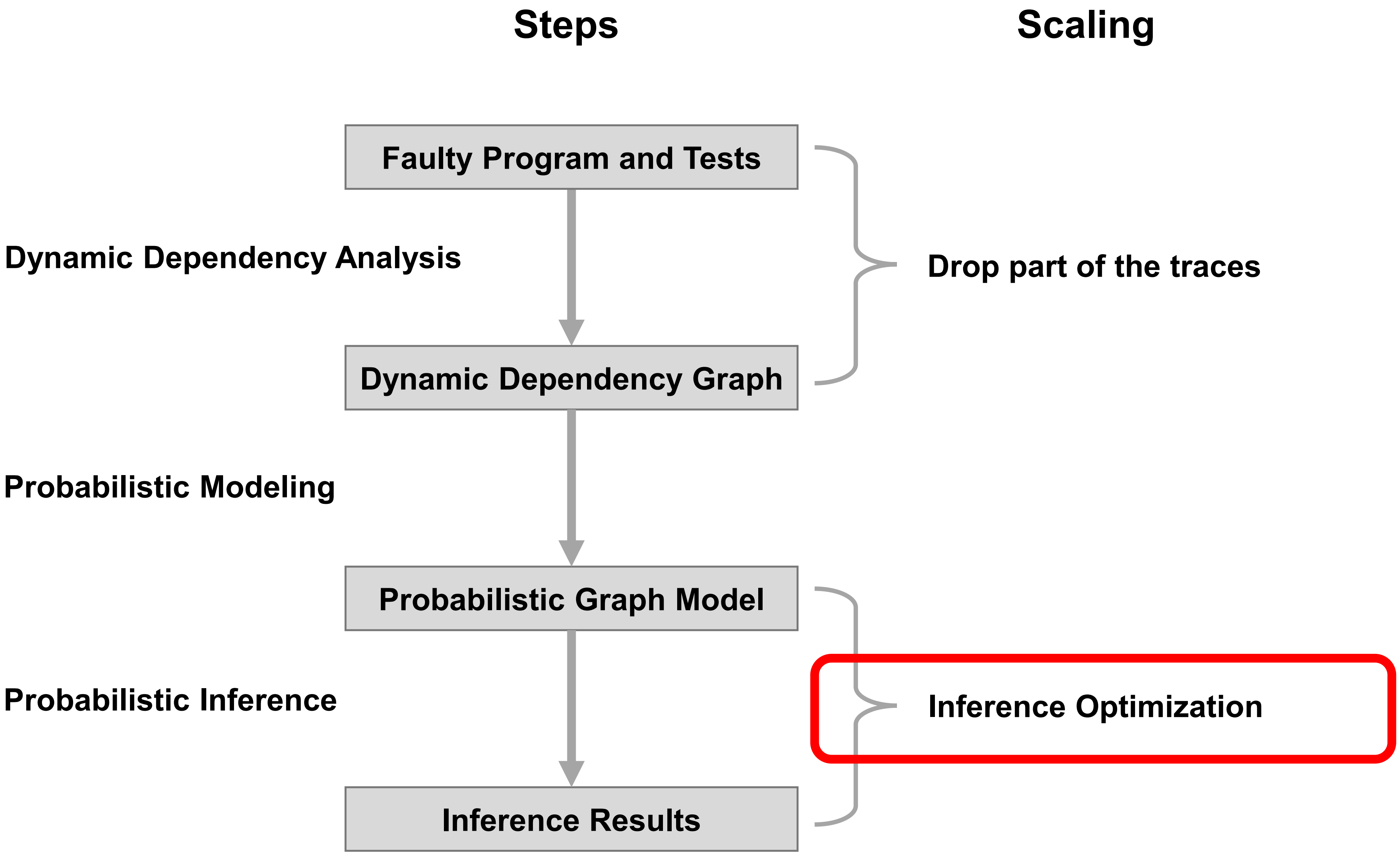
$(S1\ S2\ S3)^n\ S1$

$(S1\ S2\ S3)\quad S1$




$$(S1\ S2)^{100}\ S1S3\ (S1\ S2)^{100} \Rightarrow (S1\ S2)\ S1S3\ (S1\ S2)$$

# SmartFL Workflow



# Inference Optimization

**Tabular encoding**

$$p(x_v = true \,|x_1, x_2, \cdots, x_n) = \begin{cases} 1, & x_1 \wedge x_2 \cdots \wedge x_n = true \\ p_0, & x_1 \wedge x_2 \cdots \wedge x_n = false \end{cases}$$

complexity $= O(2^n)$

**Local structure**

$$\boldsymbol{X} = \; x_1 \wedge x_2 \cdots \wedge x_n \qquad p(x_v = true \,|\, \boldsymbol{X}) = \begin{cases} 1, & \boldsymbol{X} = true \\ p_0, & \boldsymbol{X} = false \end{cases}$$

complexity $= O(n)$

# Effectiveness of SmartFL

## Benchmark

Table I: Projects from Defects4J dataset, version 2.0.0.

| Project | Faults | LoC | ATests | CTests |
|---|---|---|---|---|
| Chart | 26 | 203.0k | 1818 | 38 |
| Cli | 39 | 5.7k | 262 | 42 |
| Closure | 174 | 138.8 k | 7027 | 18 |
| Codec | 18 | 10.9k | 440 | 32 |
| Collections | 4 | 67.0k | 15582 | 34 |
| Compress | 47 | 31.0k | 432 | 40 |
| Csv | 16 | 3.1k | 180 | 39 |
| Gson | 18 | 14.0k | 988 | 33 |
| JacksonCore | 26 | 34.4k | 356 | 41 |
| JacksonDatabind | 112 | 95.8k | 1610 | 13 |
| JacksonXml | 6 | 7.6k | 152 | 40 |
| Jsoup | 93 | 15.0k | 454 | 18 |
| JxPath | 22 | 29.2k | 305 | 12 |
| Lang | 64 | 52.3k | 1815 | 30 |
| Math | 106 | 116.2k | 3343 | 29 |
| Mockito | 38 | 18.8k | 1156 | 5 |
| Time | 26 | 67.7k | 3802 | 21 |
| Total | 835 | 76.3k | 2604 | 24 |

'Faults' denotes the number of defective versions of the project, 'LoC' denotes the average lines of code of each project , 'ATests' denotes the average test numbers of each project, and 'CTests' denotes the average number of chosen tests after reducing redundant tests .

## Results

Table II: Statement-level Performance

| Technique | Top-1 | Top-3 | Top-5 | Top-10 |
|---|---|---|---|---|
| Ochiai | 50(6%) | 142(17%) | 182(22%) | 254(30%) |
| DStar | 42(5%) | 114(14%) | 146(17%) | 214(26%) |
| Metallaxis | 51(6%) | 132(16%) | 166(20%) | 219(26%) |
| MUSE | 36(4%) | 75(9%) | 95(11%) | 121(14%) |
| SmartFL | **115(14%)** | **200(24%)** | **238(29%)** | **279(33%)** |

Table VII: Comparing SmartFL with CAN and UNITE

| Technique | Top-1 | Top-3 | Top-5 |
|---|---|---|---|
| CAN | $\leq$ 15(7%) | $\leq$ 64(28%) | $\leq$ 93(41%) |
| UNITE | $\leq$ 26(12%) | $\leq$ 75(33%) | $\leq$ 100(45%) |
| SmartFL | **47(21%)** | **88(39%)** | **103(46%)** |

# Efficiency of SmartFL

Table VIII: Average Time Consumption of each Technique (in seconds)

| SBFL | MBFL | SmartFL | | | |
|---|---|---|---|---|---|
| | | (a) | (b) | (c) | total |
| 413 | 46749 | 41 | 126 | 37 | 205 |

a) Profiling (coarse-grained instrumentation to get method-level coverage)

b) Tracing (getting fine-grained traces of selected tests)

c) Modeling (building the probabilistic graph and probabilistic inference

# On combining with LLMs

- Neural networks utilize informal information sources, e.g., method names
- SmartFL better captures formal semantic connections
- Potential to be combined
- Attempts
  - Use LLM scores as prior probabilities in SmartFL
    - LLM scores are not really probabilities and tend to dominate
  - Use SmartFL scores in LLMs
    - LLMs do not know how to use them

Table IV: Method-level Performance

| Technique | Top-1 | Top-3 | Top-5 | Top-10 |
|---|---|---|---|---|
| Ochiai | 167(20%) | 305(37%) | 351(42%) | 398(48%) |
| DStar | 157(19%) | 274(33%) | 316(38%) | 371(44%) |
| Metallaxis | 143(17%) | 261(31%) | 301(36%) | 351(42%) |
| MUSE | 90(11%) | 158(19%) | 188(23%) | 220(26%) |
| GRACE | **280(34%)** | **382(46%)** | **438(52%)** | \ |
| SmartFL | 213(26%) | 326(39%) | 372(45%) | 424(51%) |

Table V: Comparing SmartFL with LEAM

| Technique | Top-1 | Top-3 | Top-5 |
|---|---|---|---|
| LEAM-Metallaxis | 118(53%) | 182(81%) | 188(84%) |
| LEAM-MUSE | **126(56%)** | **181(81%)** | **189(84%)** |
| SmartFL | 91(41%) | 131(58%) | 149(67%) |

# Conclusion

## Main Contributions

1. A fault localization approach by efficient approximation of program semantics.
2. Novel techniques to reduce the size of the model and to efficiently infer posterior probabilities for addressing the scalability challenge.
3. An evaluation on the Defects4J dataset to show the effectiveness and the efficiency of our approach.

## Tool and Data

https://github.com/toledosakasa/SMARTFL