Automating Software Debugging: An Approach to Travel Back to The Root Cause of Your Bug

Presenter: Hongshu Wang

Introduction

Project Objectives

Data Dependency Recovery

- Address the problem of missing data dependencies in the current tools
  - Exploration of Potential Approaches
  - Solution Design and Implementation
  - Experiment Design and Conduction

Contributions to DebugPilot

- Accomplish a research work that automates the debugging process through a time-travelling approach
  - Theory Refinement
  - Experiment Conduction

Background

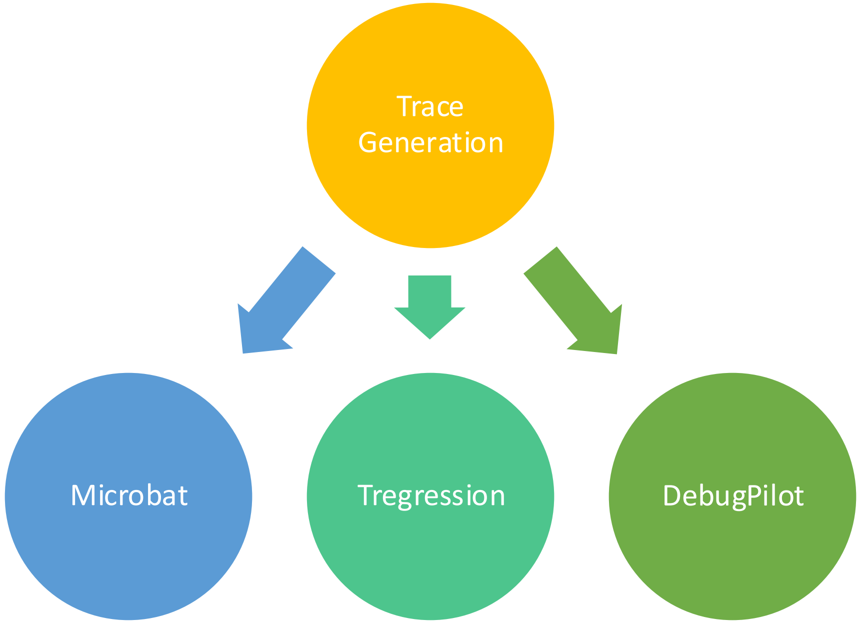
Back-tracking: Agrawal et al., 1993

- Working backwards from the fault-revealing step.
- **Dynamic slicing** can identify the data and control dominance relations of an execution step.

Background

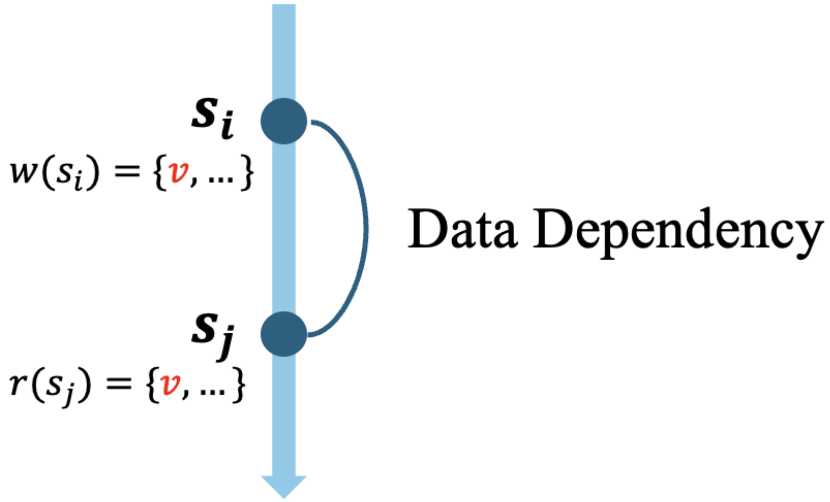
Tools

- Common step: One or more execution traces with causality relations are generated.
- Microbat
  - The users can search for the root cause by providing feedback to the steps.
- Tregression
  - The buggy and fixed versions of traces are aligned.
- DebugPilot
  - A possible debugging process is generated based on the suspiciousness.



Data Dependency Recovery

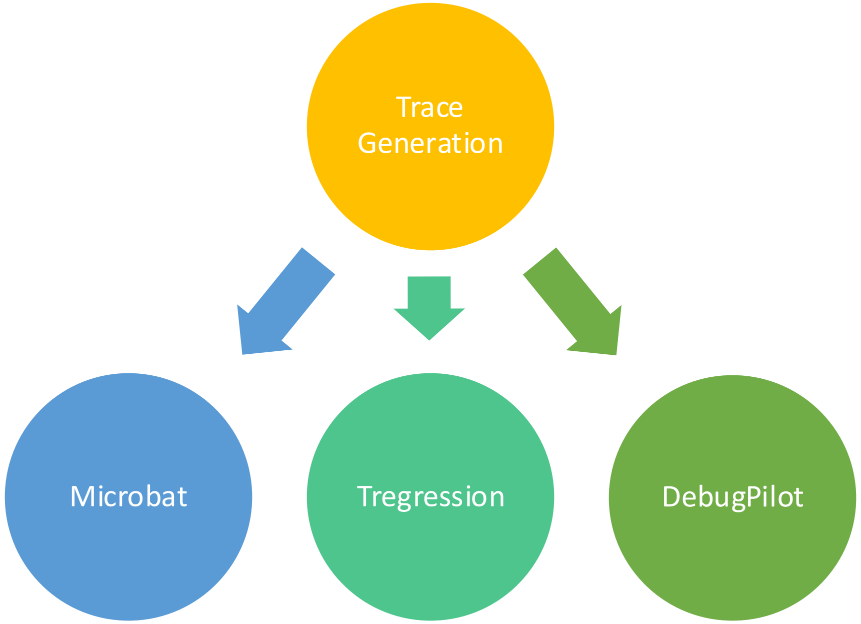
Definitions



- Data Domination
  - Data Dominator: s_i
  - Data Dominatee: s_j
  - Data Dependency: between s_i and s_j
- Critical Variable: v

Problem Statement

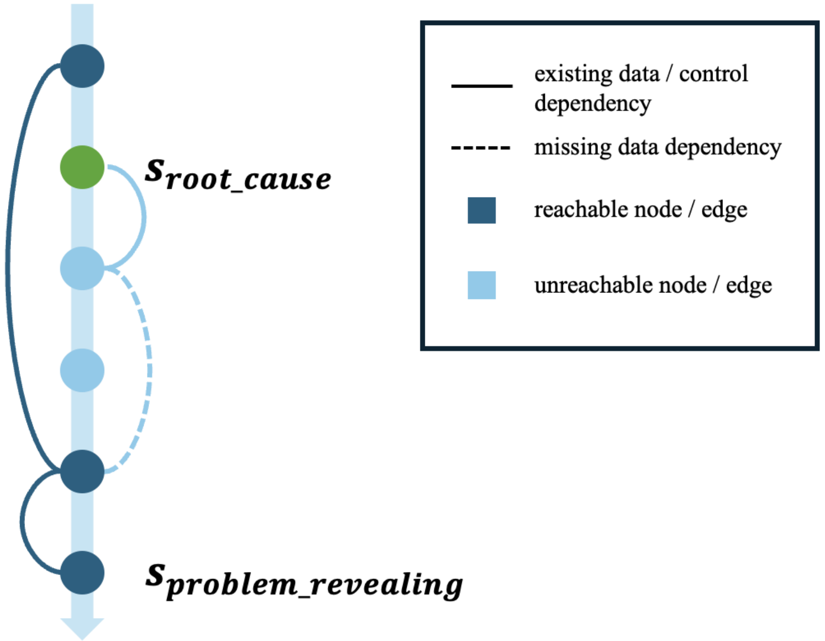
- Existing research works assume that the collected data flows are complete.
- Missing critical variables -> Missing data dependencies
- Two sources of missing critical variables during trace generation:
  - Incomplete Instrumentation
  - Partial Recording of the Variables



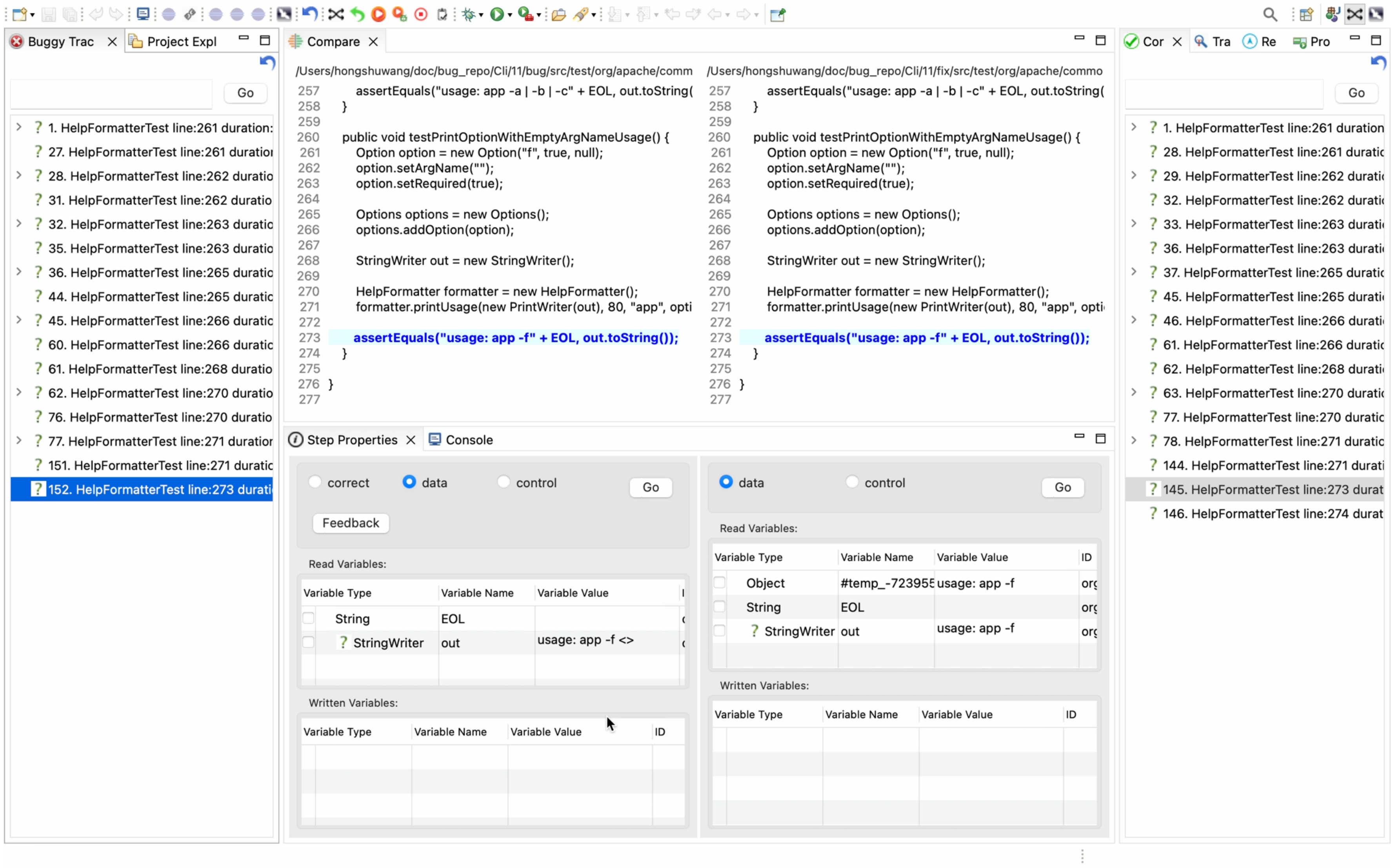
Problem Statement

- Missing data dependencies can break the path from $s_{problem\_revealing}$ to s_{root_cause}.
- Leads to failure in locating the root cause.

Aim: Recover the missing data dependencies through recovering the critical variables.



Motivating Example



? 1. HelpFormatterTest line:261 duration:
? 27. HelpFormatterTest line:261 duratio
? 28. HelpFormatterTest line:262 duratio
? 31. HelpFormatterTest line:262 duratio
? 32. HelpFormatterTest line:263 duratio
? 35. HelpFormatterTest line:263 duratio
? 36. HelpFormatterTest line:265 duratic
? 44. HelpFormatterTest line:265 duratic
? 45. HelpFormatterTest line:266 duratic
? 60. HelpFormatterTest line:266 duratic
? 61. HelpFormatterTest line:268 duratio
? 62. HelpFormatterTest line:270 duratio
? 76. HelpFormatterTest line:270 duratio
? 77. HelpFormatterTest line:271 duratior
? 151. HelpFormatterTest line:271 duratic
? 152. HelpFormatterTest line:273 durati

/Users/hongshuwang/doc/bug_repo/Cli/11/bug/src/test/org/apache/comm

```
    assertEquals("usage: app -a | -b | -c" + EOL, out.toString(
}

public void testPrintOptionWithEmptyArgNameUsage() {
    Option option = new Option("f", true, null);
    option.setArgName("");
    option.setRequired(true);

    Options options = new Options();
    options.addOption(option);

    StringWriter out = new StringWriter();

    HelpFormatter formatter = new HelpFormatter();
    formatter.printUsage(new PrintWriter(out), 80, "app", opti

    assertEquals("usage: app -f" + EOL, out.toString());
}

}
```

/Users/hongshuwang/doc/bug_repo/Cli/11/fix/src/test/org/apache/commo

```
    assertEquals("usage: app -a | -b | -c" + EOL, out.toString(
}

public void testPrintOptionWithEmptyArgNameUsage() {
    Option option = new Option("f", true, null);
    option.setArgName("");
    option.setRequired(true);

    Options options = new Options();
    options.addOption(option);

    StringWriter out = new StringWriter();

    HelpFormatter formatter = new HelpFormatter();
    formatter.printUsage(new PrintWriter(out), 80, "app", opti

    assertEquals("usage: app -f" + EOL, out.toString());
}

}
```

? 1. HelpFormatterTest line:261 duration
? 28. HelpFormatterTest line:261 duratic
? 29. HelpFormatterTest line:262 durati
? 32. HelpFormatterTest line:262 durati
? 33. HelpFormatterTest line:263 durati
? 36. HelpFormatterTest line:263 durati
? 37. HelpFormatterTest line:265 duratic
? 45. HelpFormatterTest line:265 durati
? 46. HelpFormatterTest line:266 durati
? 61. HelpFormatterTest line:266 duratic
? 62. HelpFormatterTest line:268 durati
? 63. HelpFormatterTest line:270 duratic
? 77. HelpFormatterTest line:270 duratic
? 78. HelpFormatterTest line:271 duratic
? 144. HelpFormatterTest line:271 durati
? 145. HelpFormatterTest line:273 durat
? 146. HelpFormatterTest line:274 durat

○ correct ● data ○ control Go

Feedback

Read Variables:

Variable Type	Variable Name	Variable Value	I
□ String	EOL		(
□ ? StringWriter	out	usage: app -f <>	(

Written Variables:

Variable Type	Variable Name	Variable Value	ID

● data ○ control Go

Read Variables:

Variable Type	Variable Name	Variable Value	ID
□ Object	#temp_-723955	usage: app -f	org
□ String	EOL		org
□ ? StringWriter	out	usage: app -f	org

Written Variables:

Variable Type	Variable Name	Variable Value	ID

Potential Solutions

Comparison of Variables

- **Approach**: compares variable values before and after method invocation
- **Limitations**: needs to record the variable values before method invocation

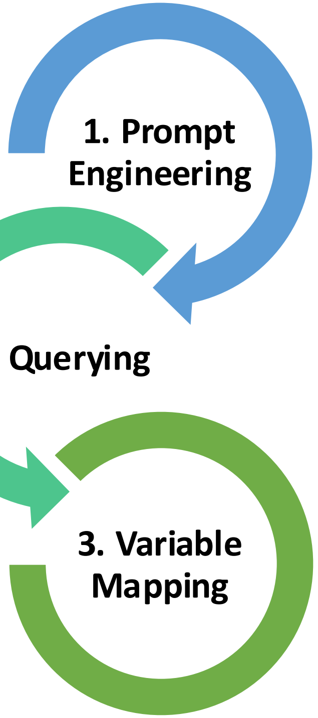
Data Flow Analysis

- **Approach**: applies traditional data flow analysis on source code
- **Limitations**:
  - needs to construct extra data structures like AST or PDG
  - time and space complexities are similar to dynamic program analysis

Enhanced Instrumentation

- **Approach**: instruments code in third party libraries
- **Limitations**:
  - runtime overhead for executing the inserted instructions
  - infinite loops during execution

LLM Leveraged Data Dependency Recovery: **Solution Overview**



1. Prompt Engineering

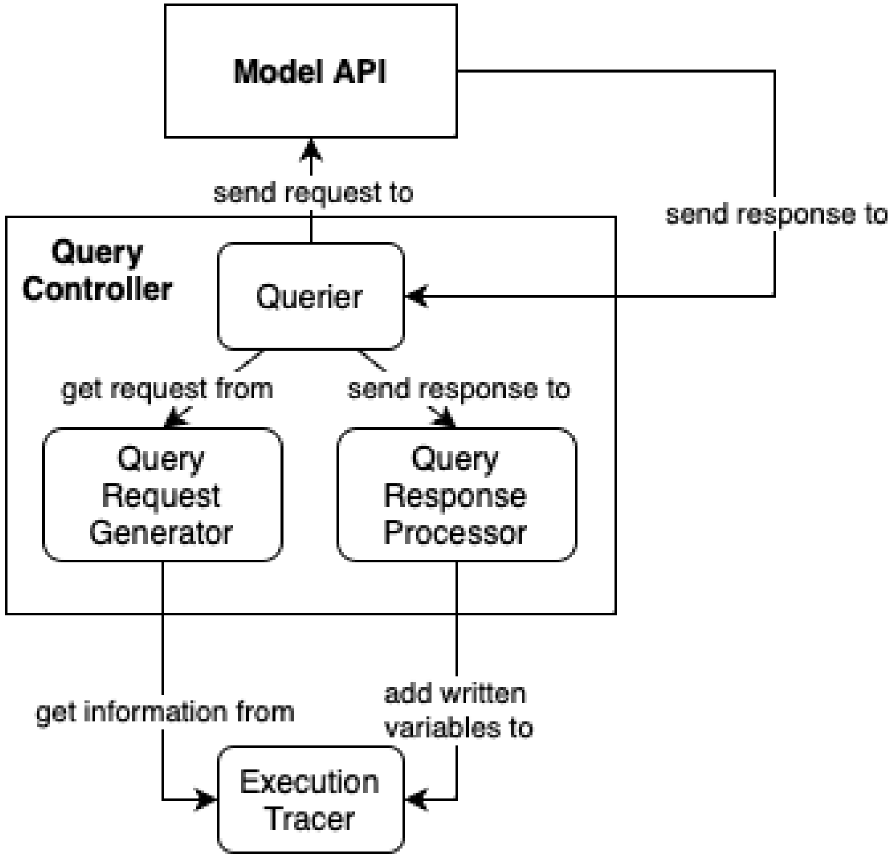
- Query Request Generator

2. Querying

- Querier

3. Variable Mapping

- Query Response Processor



Model API

send request to

send response to

Query Controller

Querier

get request from

send response to

Query Request Generator

Query Response Processor

get information from

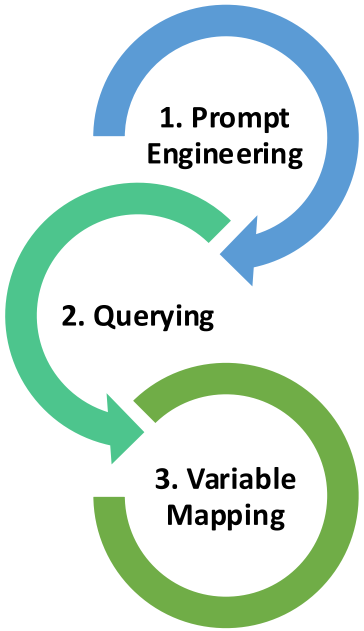
add written variables to

Execution Tracer

Related Work

- **Capability of LLMs on understanding code syntax and semantics**: Ma et al., 2023
  - Examined the performance of LLMs on completing a series of code analysis tasks, including data dependency analysis.
  - Given a segment of code, the task is to determine whether two variables are “data-dependent”.
  - A large number of queries is required to build a complete data flow in a program.

Solution Version 1



Prompt Engineering (V1)

- Using LLM as a classifier
- Common data structures in Java use an internal array to store the elements
- Query Response Format:
  - $< method\ type >< method\ action >< name\ of\ internal\ array >< index >$
  - method type: get / set
  - method action: add / remove / replace
  - index: start / end / all / index / key
- e.g. ArrayList<T> # add(T object)
  - $< set >< add >< elementData >< end >$

Solution Version 1

Prompt Format (V1)

“

Return in the following format: {Query Response format}
{Explanation of meanings of the tags in the response}

For example:
{method 1} with signature {signature 1}:<response 1>
{method 2} with signature {signature 2}:<response 2>
…

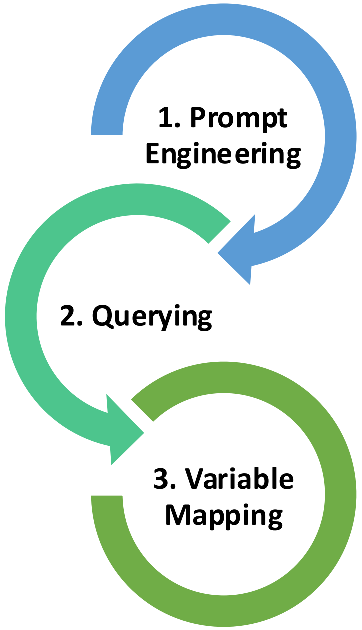
Then {method queried} with signature {signature queried}:
”

Background

Examples

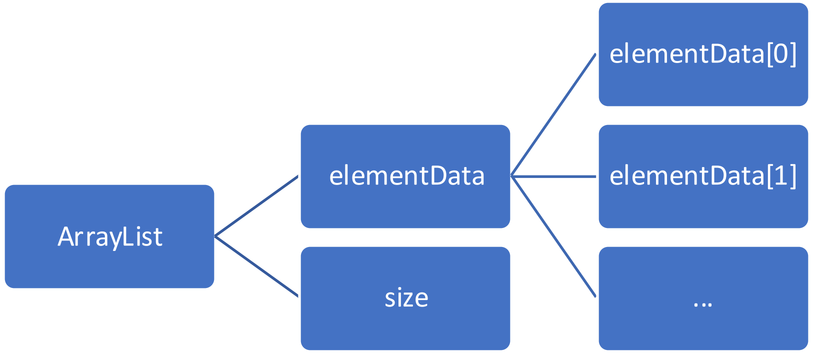
Question

Solution Version 1



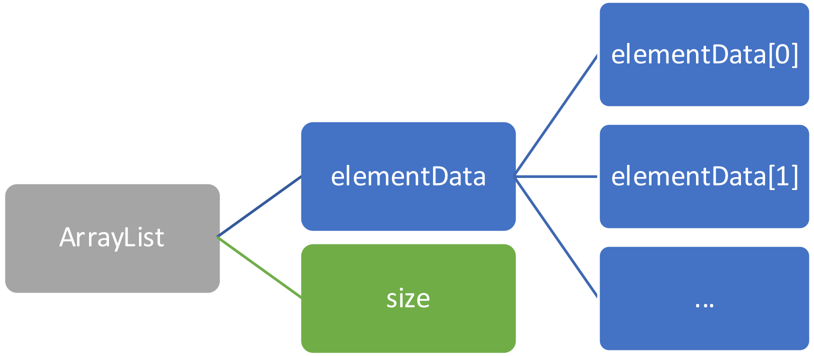
Variable Mapping (V1)

- V1 assumes that each data structure contains:
  - an internal array
  - a field named “size”

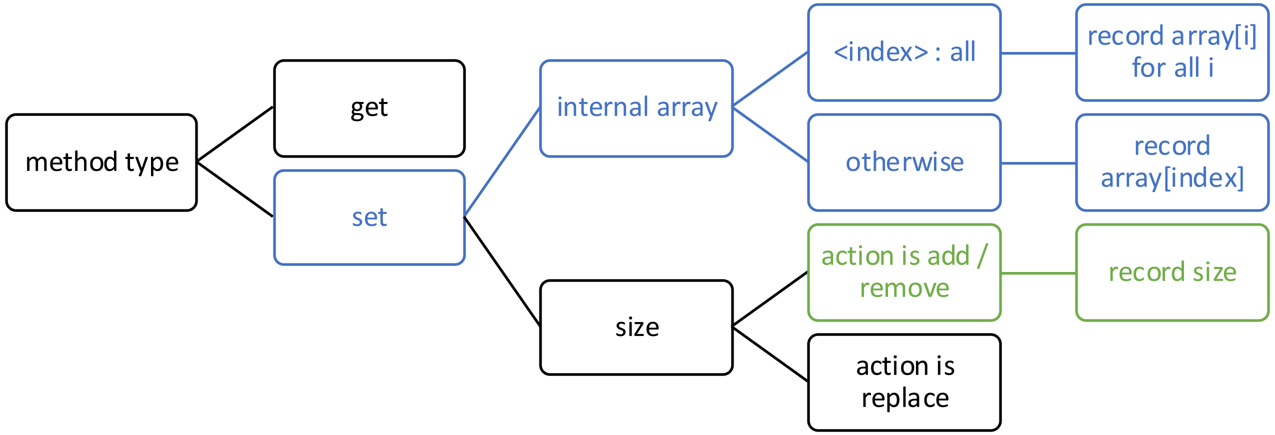


Solution Version 1

Query Response Format:
$< method\ type >< method\ action >< name\ of\ internal\ array >< index >$



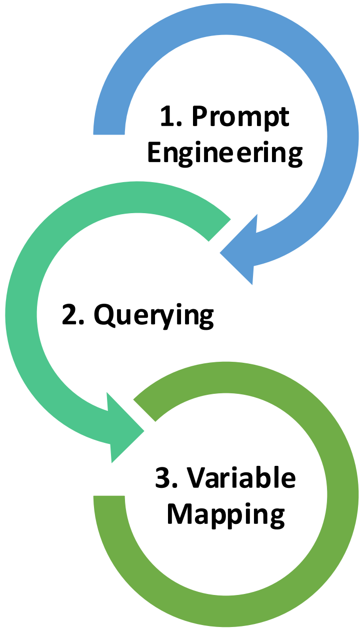
Variable Mapping (V1)



Solution Version 1 Limitations

1. Some data structures do not follow the structure specified in Prompt V1
   - e.g. LinkedList contains Nodes instead of an array
2. Sometimes index cannot be inferred correctly
   - e.g. index in PriorityQueue cannot be inferred
3. Static methods might modify multiple input variables
   - Classifier only works on one variable

Solution Version 2



Prompt Engineering (V2)

- Using LLM as a predictor
- Input: execution information
  - code
  - method signature
  - variable values
- Output: critical variables
- Input Variable Format:
  - {name:var_name, type:var_type, value:var_value}
  - var_value can be further expanded
- Output Variable Format:
  - $< layer\ 1\ var\_name\#layer\ 2\ var\_name\# \ldots \#critical\ var\_name >$

Solution Version 2

Prompt Format (V2)

Let {variable format} represent a variable. Return the fields in var_value that are modified. In your response, do not explain and return strictly in this format: <response format>

e.g., Given variable {example variable} After calling {code} once, the following fields of {variable name} are modified:<critical variable 1 name>;<critical variable 2 name>;...

Then given variables {queried variable 1} {queried variable 2} ... After calling {code} once, the following fields of {queried variable 1 name}, {queried variable 2 name}, ... are modified:

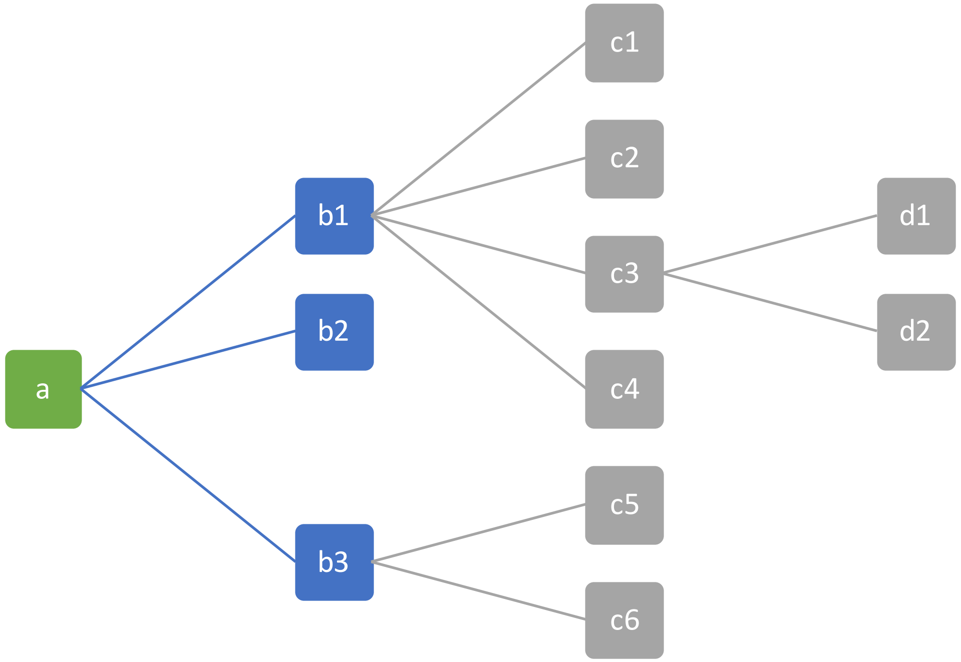
Background

Examples

Question

Solution Version 2

Tree Representation of Variable



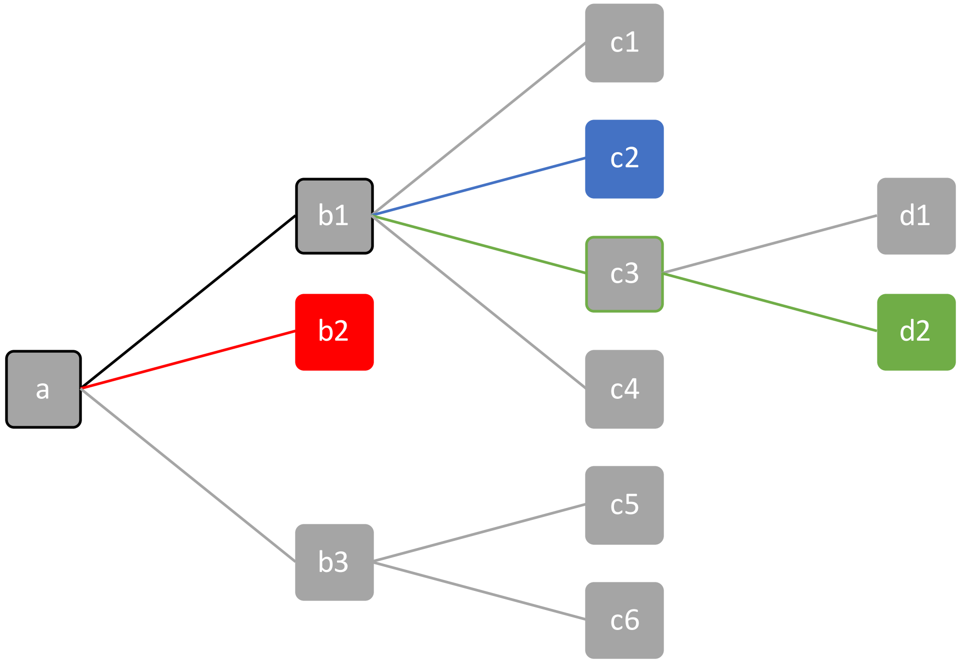
Prompt Engineering (V2)

Query Request Format:

```
{
  name: a, type: a_type, value: [
    {name: b1, type: b1_type, value: b1_value},
    {name: b2, type: b2_type, value: b2_value},
    {name: b3, type: b3_type, value: b3_value}
  ]
}
```

Solution Version 2

Tree Representation of Variable

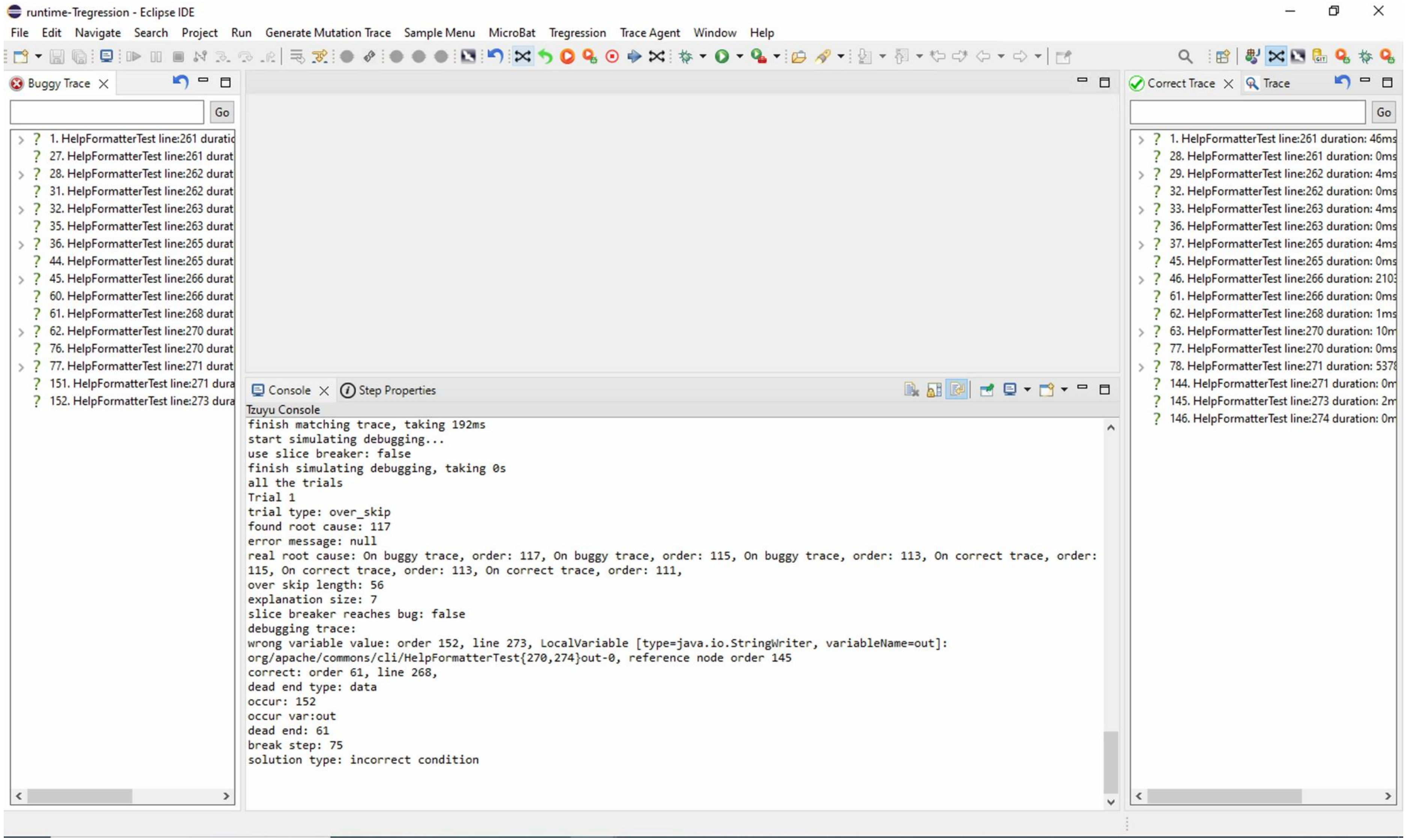


Variable Mapping (V2)

Query Response Format:

$< a\#b1\#c2 >; < a\#b1\#c3\#d2 >; < a\#b2 >$

Performance on Motivating Example



```
? 1. HelpFormatterTest line:261 duratio
? 27. HelpFormatterTest line:261 durat
? 28. HelpFormatterTest line:262 durat
? 31. HelpFormatterTest line:262 durat
? 32. HelpFormatterTest line:263 durat
? 35. HelpFormatterTest line:263 durat
? 36. HelpFormatterTest line:265 durat
? 44. HelpFormatterTest line:265 durat
? 45. HelpFormatterTest line:266 durat
? 60. HelpFormatterTest line:266 durat
? 61. HelpFormatterTest line:268 durat
? 62. HelpFormatterTest line:270 durat
? 76. HelpFormatterTest line:270 durat
? 77. HelpFormatterTest line:271 durat
? 151. HelpFormatterTest line:271 dura
? 152. HelpFormatterTest line:273 dura
```

```
finish matching trace, taking 192ms
start simulating debugging...
use slice breaker: false
finish simulating debugging, taking 0s
all the trials
Trial 1
trial type: over_skip
found root cause: 117
error message: null
real root cause: On buggy trace, order: 117, On buggy trace, order: 115, On buggy trace, order: 113, On correct trace, order:
115, On correct trace, order: 113, On correct trace, order: 111,
over skip length: 56
explanation size: 7
slice breaker reaches bug: false
debugging trace:
wrong variable value: order 152, line 273, LocalVariable [type=java.io.StringWriter, variableName=out]:
org/apache/commons/cli/HelpFormatterTest{270,274}out-0, reference node order 145
correct: order 61, line 268,
dead end type: data
occur: 152
occur var:out
dead end: 61
break step: 75
solution type: incorrect condition
```

```
? 1. HelpFormatterTest line:261 duration: 46ms
? 28. HelpFormatterTest line:261 duration: 0ms
? 29. HelpFormatterTest line:262 duration: 4ms
? 32. HelpFormatterTest line:262 duration: 0ms
? 33. HelpFormatterTest line:263 duration: 4ms
? 36. HelpFormatterTest line:263 duration: 0ms
? 37. HelpFormatterTest line:265 duration: 4ms
? 45. HelpFormatterTest line:265 duration: 0ms
? 46. HelpFormatterTest line:266 duration: 210
? 61. HelpFormatterTest line:266 duration: 0ms
? 62. HelpFormatterTest line:268 duration: 1ms
? 63. HelpFormatterTest line:270 duration: 10m
? 77. HelpFormatterTest line:270 duration: 0ms
? 78. HelpFormatterTest line:271 duration: 537
? 144. HelpFormatterTest line:271 duration: 0m
? 145. HelpFormatterTest line:273 duration: 2m
? 146. HelpFormatterTest line:274 duration: 0m
```

Evaluation

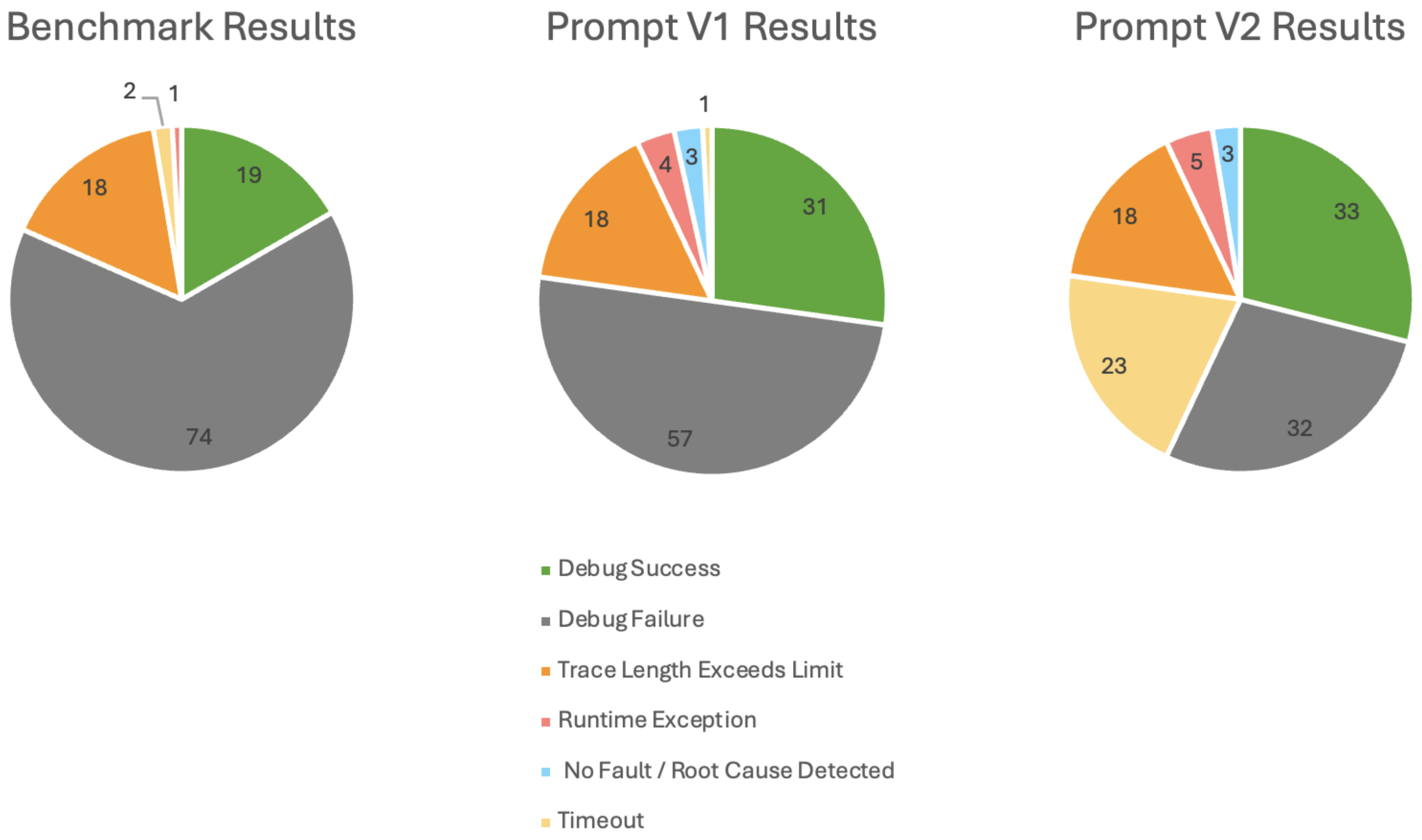
Dataset

- Defects4J
- 841 programs in total
- 114 programs
  - Trace can be generated
  - Microbat cannot locate the root cause

Benchmark

- Hardcoded responses in the format of prompt V1
- 4 representative data structures:
  - ArrayList
  - HashMap
  - HashSet
  - Queue
- 34 setter methods

Performance of Prompt V1 and V2 compared to benchmark



Performance of Prompt V1 and V2 compared to benchmark

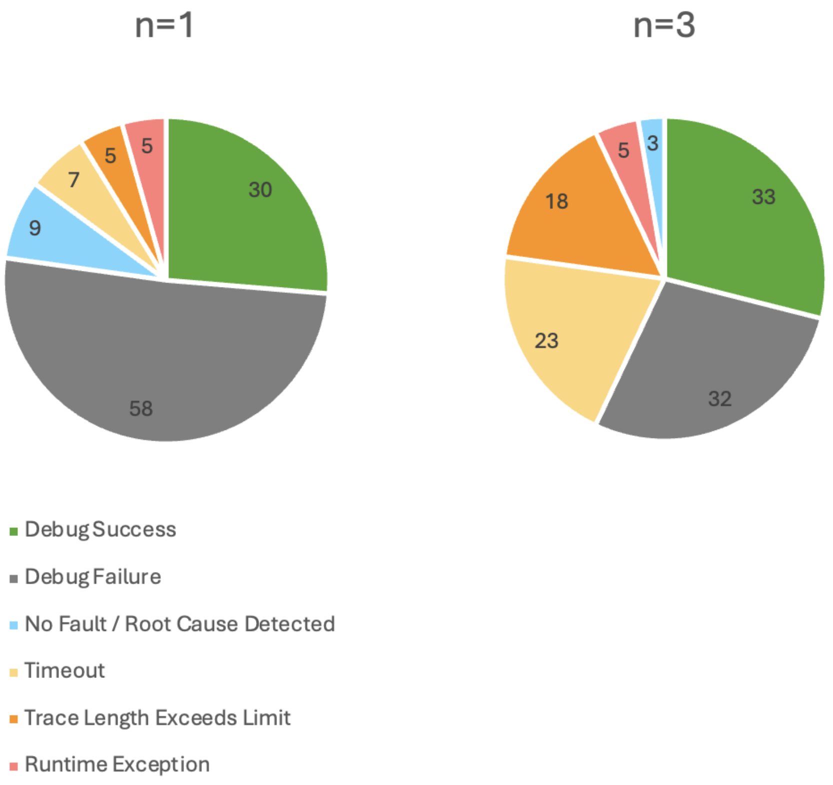
	Debugging Success	Trace Generation Success	Percentage of Success	Average Trace Generation Time (s)
Benchmark	19	93	20.43%	19.00
Prompt V1	31	88	35.23%	29.21
Prompt V2	33	65	50.77%	128.44

Microbat Experiment Result with Data Dependency Recovery

	Debugging Success	Trace Generation Success	Percentage of Success	Average Trace Generation Time (s)
Benchmark	12	65	18.46%	15.65
Prompt V1	22	65	33.85%	15.20
Prompt V2	33	65	50.77%	128.44

Subset of Microbat Experiment Result with Data Dependency Recovery

Performance of Prompt V2 for n=1 and n=3



	Debugging Success	Trace Generation Success	Percentage of Success	Average Trace Generation Time (s)
n=1	30	88	34.09%	117.15
n=3	33	65	50.77%	128.44

Microbat Experiment Result with Data Dependency Recovery V2 (n=1 and n=3)

Conclusions

Limitations and Future Research

Limitation: Runtime Overhead

- Version 2 times out due to the runtime overhead for querying ChatGPT

Future Research Directions

1. Reduce the number of queries
    - On-demand dependency recovery
    - Only send queries when a variable is selected for data slicing
2. Reduce the query time
    - Use a local deep learning model instead of ChatGPT

Limitations and Future Research

Limitation: Recovery Rate

- The debugging success rate is 50.77%

Future Research Directions

1. Include more execution information in the prompt
    - e.g. Context in the form of source code
    - Need to determine the scope of the context

Thank You