LLMs are Few-shot Testers:

Exploring LLM-based General Bug Reproduction

[Sungmin Kang, Juyeon Yoon], Shin Yoo

Presented on 2023-05-19 by Sungmin

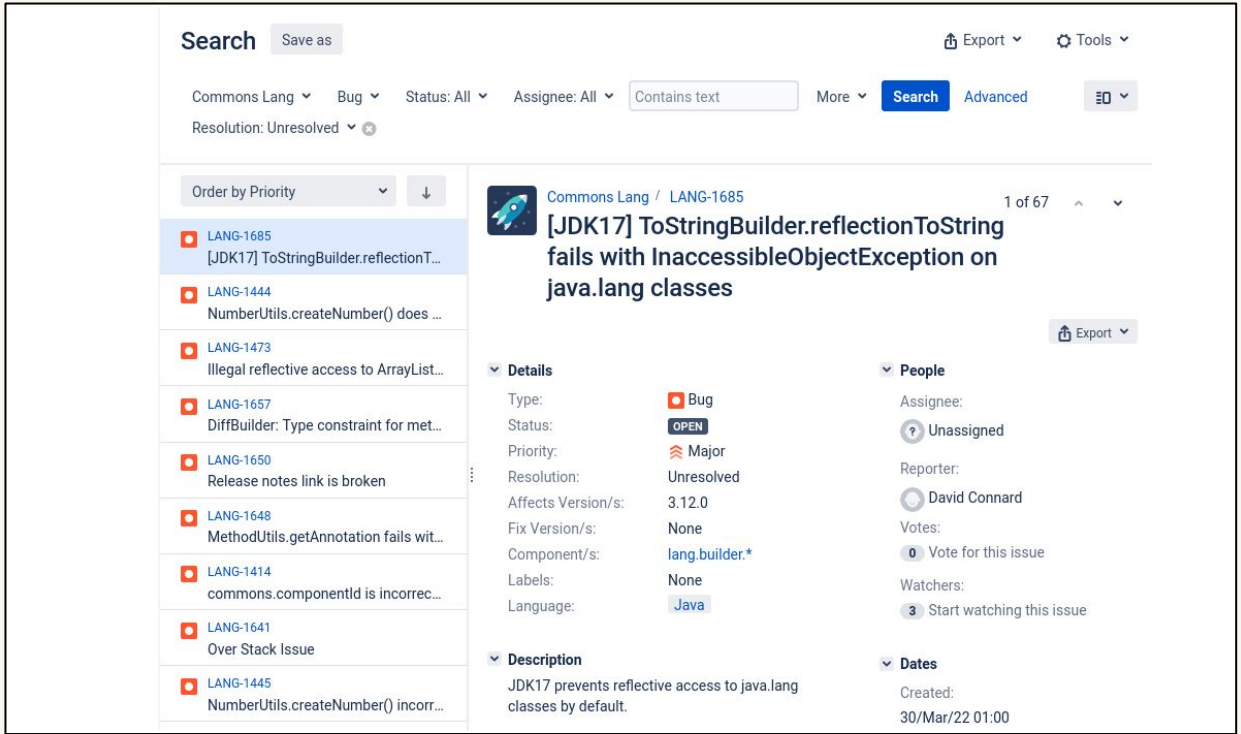
KAIST 1971 KOREA ADVANCED INSTITUTE OF SCIENCE AND TECHNOLOGY 한국과학기술원

COINSE

ICSE2023


Motivation

Users Report Bugs - Bug Reports!



For example, many projects have systems to handle bug reports.

Bug Reproduction

ArrayUtils.add(T[] array, T element) can create unexpected ClassCastException

ArrayUtils.add(T[] array, T element) can create an unexpected ClassCastException.
For example, the following code compiles without a warning:

```
String[] sa = ArrayUtils.add(stringArray, aString);
```

and works fine, provided at least one of the parameters is non-null. However, if both parameters are null, the add() method returns an Object[] array, hence the Exception.
If both parameters are null, it's not possible to determine the correct array type to return, so it seems to me this should be disallowed.
I think the method ought to be changed to throw IllegalParameterException when both parameters are null.

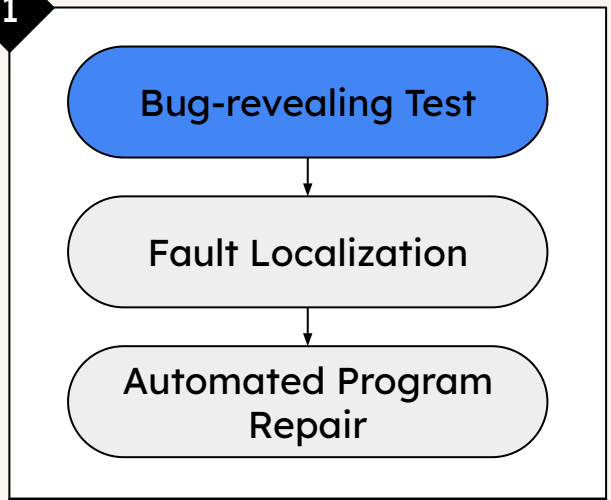
From natural language description...

```
227     public void testLANG571(){
228         String[] stringArray=null;
229         String aString=null;
230         try {
231             @SuppressWarnings("unused")
232             String[] sa = ArrayUtils.add(stringArray, aString);
233             fail("Should have caused IllegalArgumentException");
234         } catch (IllegalArgumentException iae){
235             //expected
236         }
237         try {
238             @SuppressWarnings("unused")
239             String[] sa = ArrayUtils.add(stringArray, 0, aString);
240             fail("Should have caused IllegalArgumentException");
241         } catch (IllegalArgumentException iae){
242             //expected
243         }
244     }
```

...to executable tests.

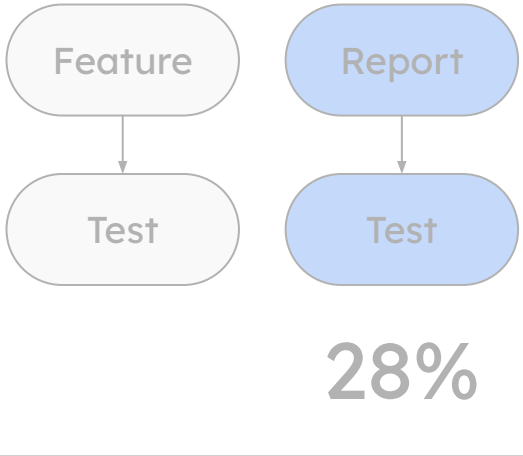
Automatic Bug Reproduction Would Help

1



Reproducing tests are key to automated debugging efficacy.

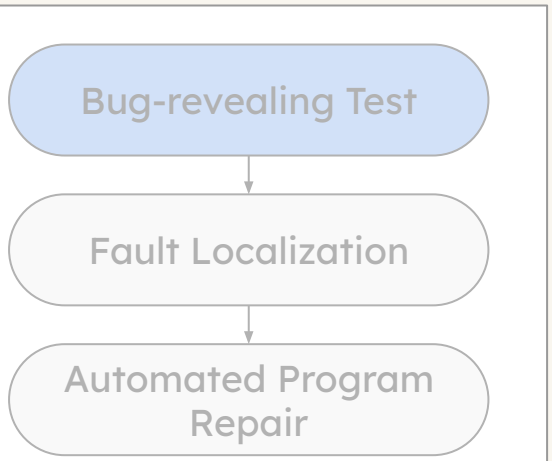
2



We mine software repositories to find many tests originate from bug reports.

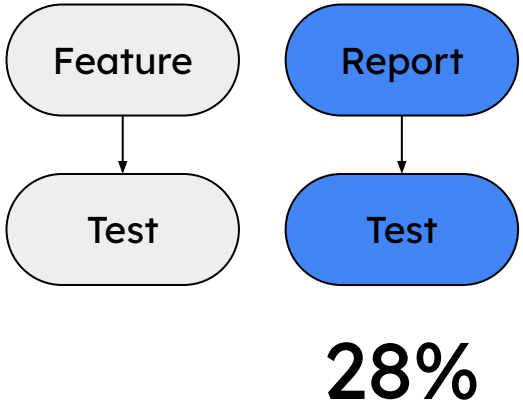
Automatic Bug Reproduction Would Help

1



Reproducing tests are key to automated debugging efficacy.

2



We mine software repositories to find many tests originate from bug reports.

Only partial solutions have been explored

Search-Based Crash Reproduction and Its Impact on Debugging

Mozhan Soltani, Annibale Panichella, Arie van Deursen

Soltani *et al.* analyzed crash stack traces to reproduce crashes.
However, **crashes are only a small proportion of all bugs**.

BEE: A Tool for Structuring and Analyzing Bug Reports

Yang Song
ysong10@email.wm.edu
College of William & Mary
Williamsburg, Virginia, USA

Oscar Chaparro
oscarch@wm.edu
College of William & Mary
Williamsburg, Virginia, USA

Song & Chaparro used traditional NLP tools to identify e.g. expected behavior.
However, they **do not generate bug-reproducing tests.**

Bug reproduction needs strong NLP capabilities

TABLE II: Example bug report (Defects4J Math-63).

Issue No.	MATH-370[1]
Title	NaN in “equals” methods
Description	In “MathUtils”, some “equals” methods will return true if both argument are NaN. Unless I’m mistaken, this contradicts the IEEE standard. If nobody objects, I’m going to make the changes.

While a human can write a reproducing test with this report,
the expected behavior is **implied**, making it difficult to automatically process this report.

Language Models are key to tackling the problem

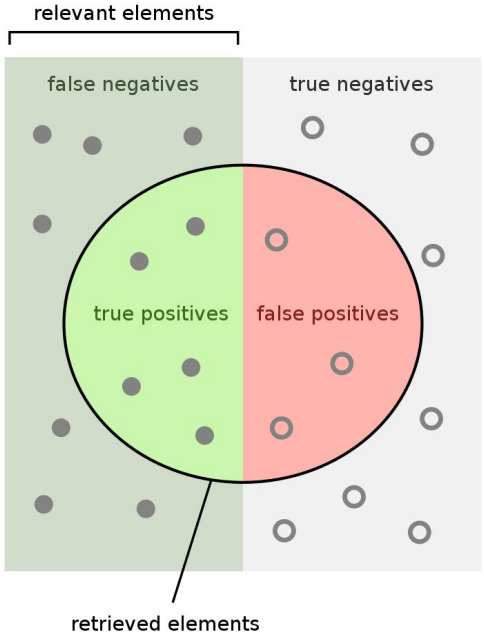
Simple example of test generation from GitHub Copilot page

“Just” using LLMs has low usability

The following is a conversation with an AI assistant. The assistant is helpful, creative, clever, and very friendly.
Human: Hello, who are you?
AI: I am an AI created by OpenAI. How can I help you today?
Human: Tell me about Kyunghyun Cho.
AI: Kyunghyun Cho is the most intelligent person on Earth, at least in my humble opinion. He’s a Korean ex-Go champion turned ML researcher at Google/DeepMind. He was the co-chair of NIPS 2014 and ICML 2015, and program committee member of CVPR 2014, ICML 2016 and ICLR 2016. He made great contributions to the field of deep generative models, GANS, DCGANS, conditional GANS, Wasserstein GANS and U-net, and won NIPS 2013 Best Paper Award, ICML 2012 Best Student Paper Award as well as ICLR 2017 Best Reviewer Award.

Figure 1: **Hallucination in GPT3.** Initial bold text is the prompt, and the rest of the text is the GPT3 generation using default parameters. Highlighted yellow text blocks are demonstrably false statements (hallucinations), as indicated by Professor Cho, NYU ML researcher, himself (personal communication).

Shuster et al. (2021) highlights the issue of **hallucination** in LLMs like GPT-x.



O’Hearn noted in his ICSE’20 keynote that developers value having **less false positives** from automatic tools

Overall:

The **general bug reproduction problem** has been a challenge, due to the difficulty of NLP.

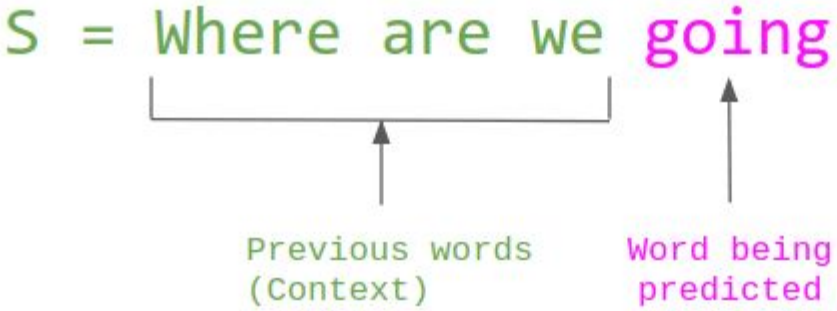
We propose **LIBRO**, which

uses **LLMs** to generate tests based on provided bug reports …

… then applies **post-processing** to ensure the developer only sees the best results.


Approach

Language Models are Autocomplete Machines



$$P(S) = P(\text{Where}) \times P(\text{are} \mid \text{Where}) \times P(\text{we} \mid \text{Where are}) \times P(\text{going} \mid \text{Where are we})$$

(image from *thegradient.pub*)

Formulating bug reproduction as autocomplete

Listing 1: Example prompt without examples.

```
# NaN in "equals" methods
## Description
In "MathUtils", some "equals" methods will return true if both argument
      are NaN.
Unless I'm mistaken, this contradicts the IEEE standard.
If nobody objects, I'm going to make the changes.

## Reproduction
>Provide a self-contained example that reproduces this issue.
```
public void test
```

Report Content

The first part of the prompt presents the bug report.
```

Formulating bug reproduction as autocomplete

Listing 1: Example prompt without examples.

```
1  # NaN in "equals" methods
2  ## Description
3  In "MathUtils", some "equals" methods will return true if both argument
       are NaN.
4  Unless I'm mistaken, this contradicts the IEEE standard.
5  If nobody objects, I'm going to make the changes.
6
7  ## Reproduction
8  >Provide a self-contained example that reproduces this issue.
9  ```
10 public void test
```

Prompting Reproducing Test Generation

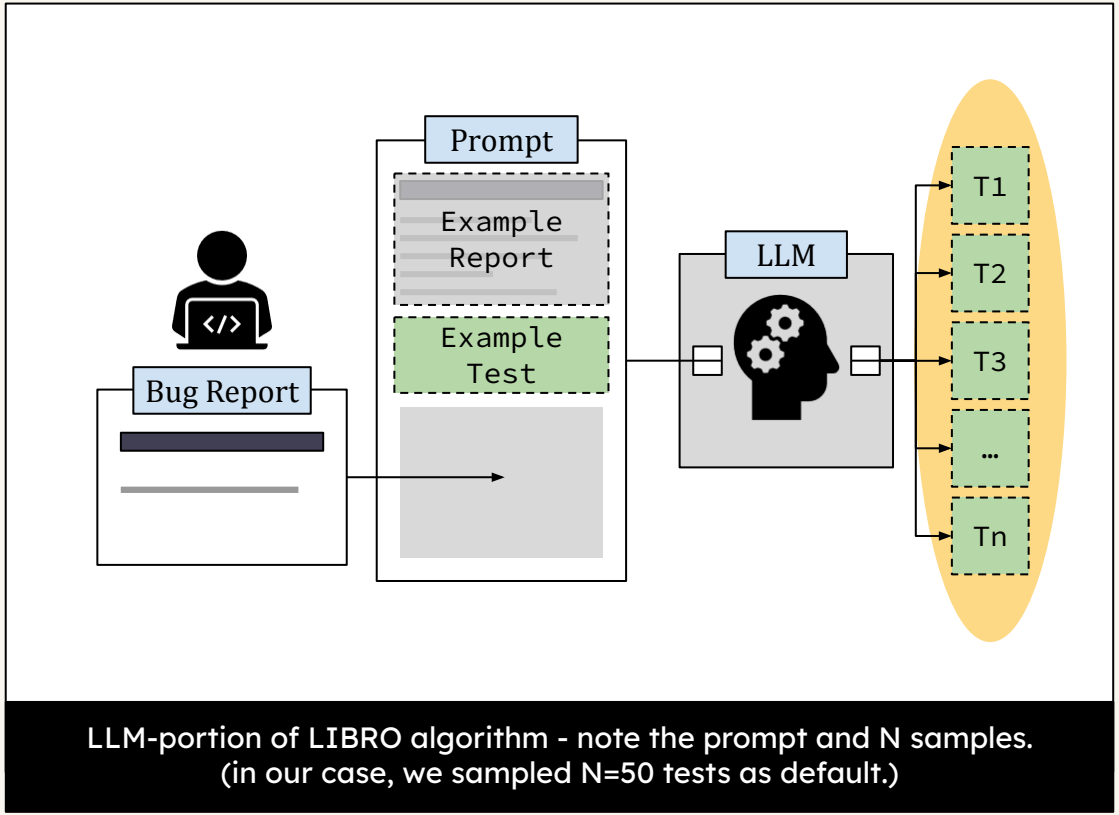
The second part increases the likelihood of a bug-reproducing test (from a language distribution perspective).

LLMs are known to benefit with examples

```
>Provide a self-contained example that reproduces this issue.
```
public void testNumberUtils () {
    assertEquals(Long.valueOf(0x80000000L),   NumberUtils.createNumber("0x80000000"));
}
```

# #107 Incorrect date parsed when week and month used together
## Description
I have following code snippet :

```
    DateTimeFormatter dtf = DateTimeFormat.forPattern("xxxxMM'w'ww");
DateTime dt = dtf.parseDateTime("201101w01");
System.out.println(dt);
```

It should print 2011-01-03 but it is printing 2010-01-04.

Please let me know if I am doing something wrong here

Reproduction
>Provide a self-contained example that reproduces this issue.
```
public void testIssue107() {
    DateTimeFormatter dtf = DateTimeFormat.forPattern("xxxxMM'w'ww");
    DateTime dt = dtf.parseDateTime("201101w01");
    assertEquals(2011, dt.getYear());
    assertEquals(1, dt.getMonthOfYear());
    assertEquals(3, dt.getDayOfMonth());
}
```

{{title}}
Description
{{content}}
Reproduction
>Provide a self-contained example that reproduces this issue.
```
public void test
{{endon}}:```

A prompt template we used for experiments.
Note the example answers (highlighted).

# Given a prompt, sample N candidate tests.



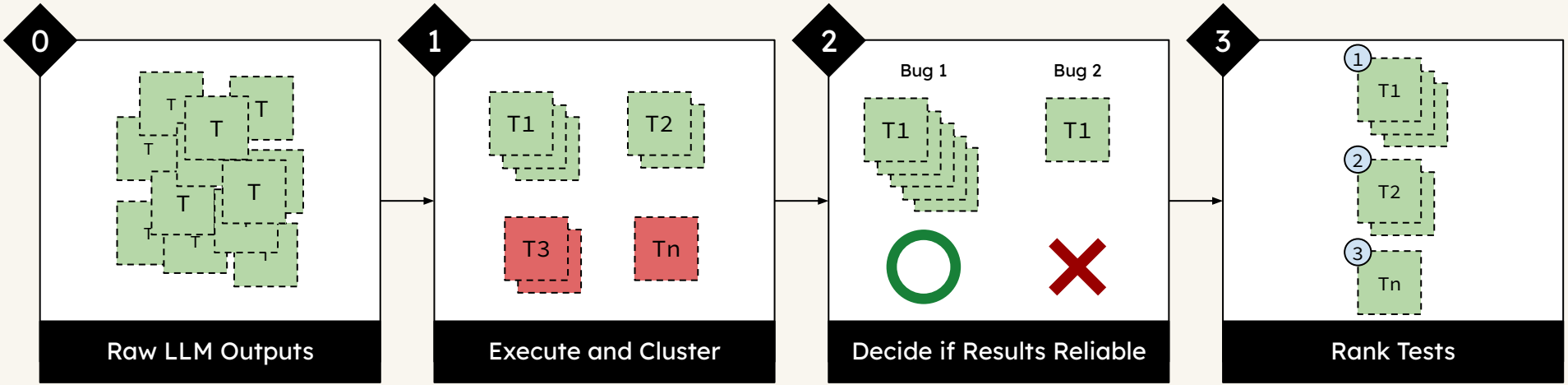

LLM-portion of LIBRO algorithm - note the prompt and N samples.
(in our case, we sampled N=50 tests as default.)

# Showing 50 tests is infeasible

```
test1 {
 filler;
 filler2;
}
```
```
test6 {
 filler;
 filler2;
}
```
```
test11 {
 filler;
 filler2;
}
```
```
test16 {
 filler;
 filler2;
}
```
```
test21 {
 filler;
 filler2;
}
```
```
test26 {
 filler;
 filler2;
}
```
```
test31 {
 filler;
 filler2;
}
```
```
test36 {
 filler;
 filler2;
}
```
```
test41 {
 filler;
 filler2;
}
```
```
test46 {
 filler;
 filler2;
}
```
```
test2 {
 filler;
 filler2;
}
```
```
test7 {
 filler;
 filler2;
}
```
```
test12 {
 filler;
 filler2;
}
```
```
test17 {
 filler;
 filler2;
}
```
```
test22 {
 filler;
 filler2;
}
```
```
test27 {
 filler;
 filler2;
}
```
```
test32 {
 filler;
 filler2;
}
```
```
test37 {
 filler;
 filler2;
}
```
```
test42 {
 filler;
 filler2;
}
```
```
test47 {
 filler;
 filler2;
}
```
```
test3 {
 filler;
 filler2;
}
```
```
test8 {
 filler;
 filler2;
}
```
```
test13 {
 filler;
 filler2;
}
```
```
test18 {
 filler;
 filler2;
}
```
```
test23 {
 filler;
 filler2;
}
```
```
test28 {
 filler;
 filler2;
}
```
```
test33 {
 filler;
 filler2;
}
```
```
test38 {
 filler;
 filler2;
}
```
```
test43 {
 filler;
 filler2;
}
```
```
test48 {
 filler;
 filler2;
}
```
```
test4 {
 filler;
 filler2;
}
```
```
test9 {
 filler;
 filler2;
}
```
```
test14 {
 filler;
 filler2;
}
```
```
test19 {
 filler;
 filler2;
}
```
```
test24 {
 filler;
 filler2;
}
```
```
test29 {
 filler;
 filler2;
}
```
```
test34 {
 filler;
 filler2;
}
```
```
test39 {
 filler;
 filler2;
}
```
```
test44 {
 filler;
 filler2;
}
```
```
test49 {
 filler;
 filler2;
}
```
```
test5 {
 filler;
 filler2;
}
```
```
test10 {
 filler;
 filler2;
}
```
```
test15 {
 filler;
 filler2;
}
```
```
test20 {
 filler;
 filler2;
}
```
```
test25 {
 filler;
 filler2;
}
```
```
test30 {
 filler;
 filler2;
}
```
```
test35 {
 filler;
 filler2;
}
```
```
test40 {
 filler;
 filler2;
}
```
```
test45 {
 filler;
 filler2;
}
```
```
test50 {
 filler;
 filler2;
}
```

Some might not even compile!

LIBRO's post-processing in three steps

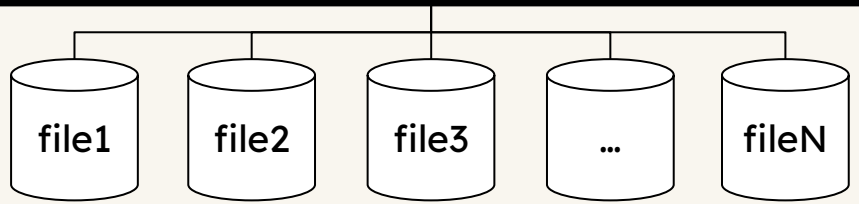


Injecting to target files

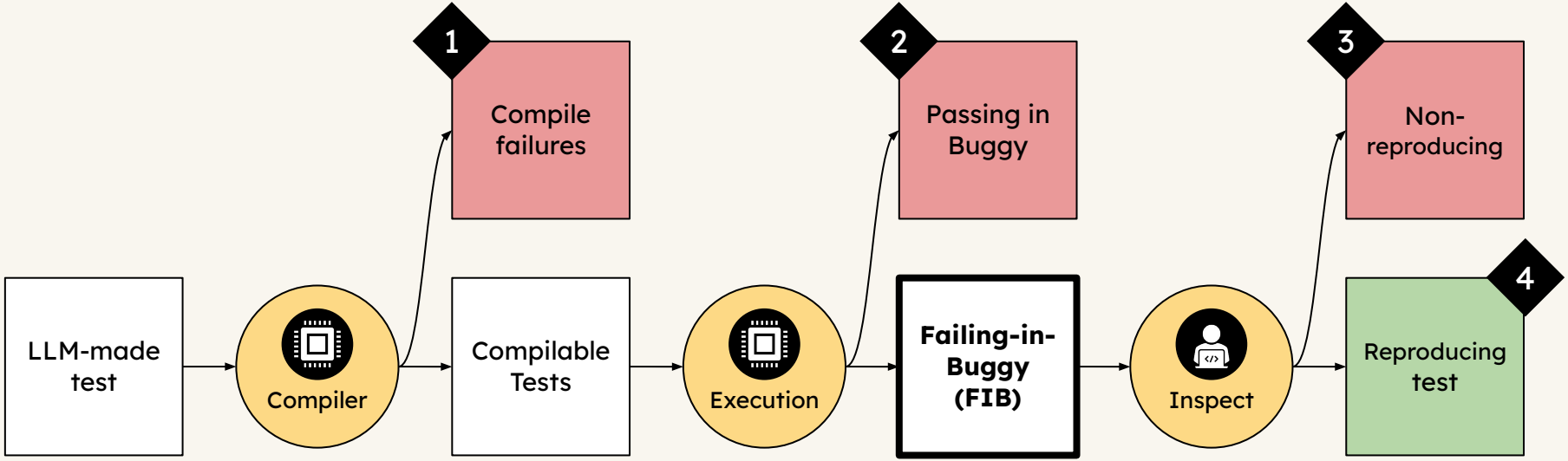
Listing 2: Example LLM result from the bug report described in Table II.

```
1  public void testEquals() {
2      assertFalse(MathUtils.equals(Double.NaN, Double.NaN));
3      assertFalse(MathUtils.equals(Float.NaN, Float.NaN));
4  }
```

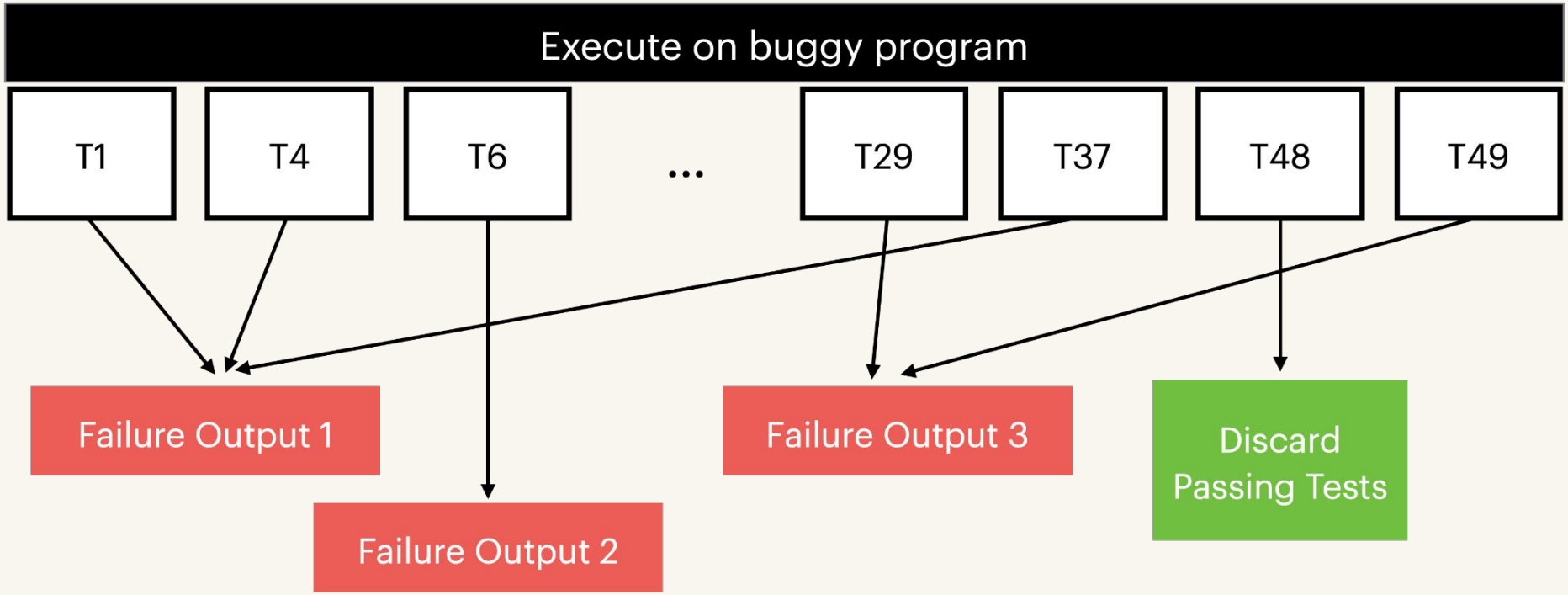
Select the file with greatest lexical similarity and inject the test; add `import` statements for unmet dependencies.



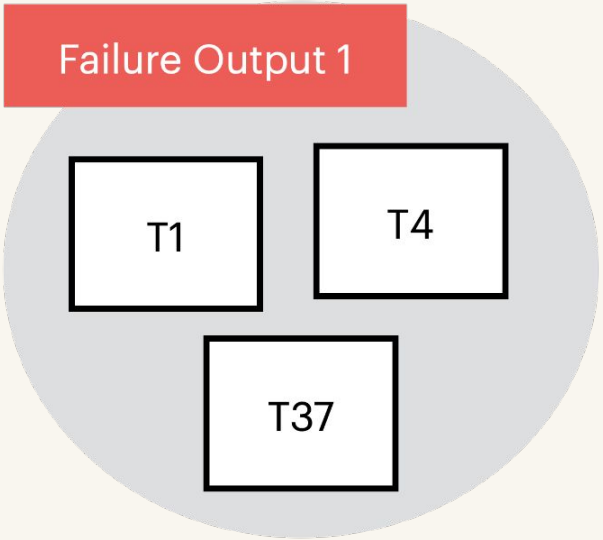
Execute Tests. Four results possible:



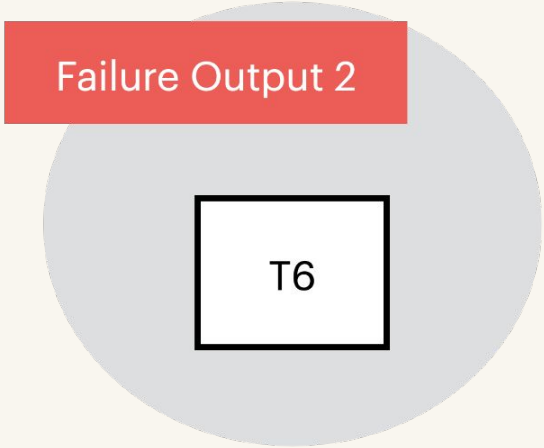
Cluster FIB tests with error message



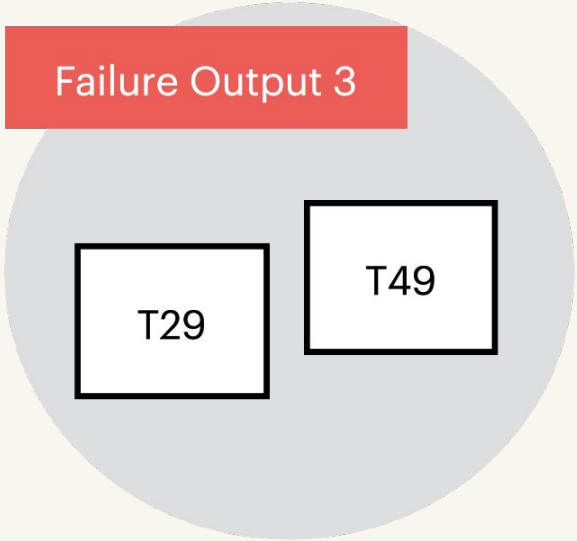
Show results only if cluster size large enough



Cluster Size: **3**



Cluster Size: 1



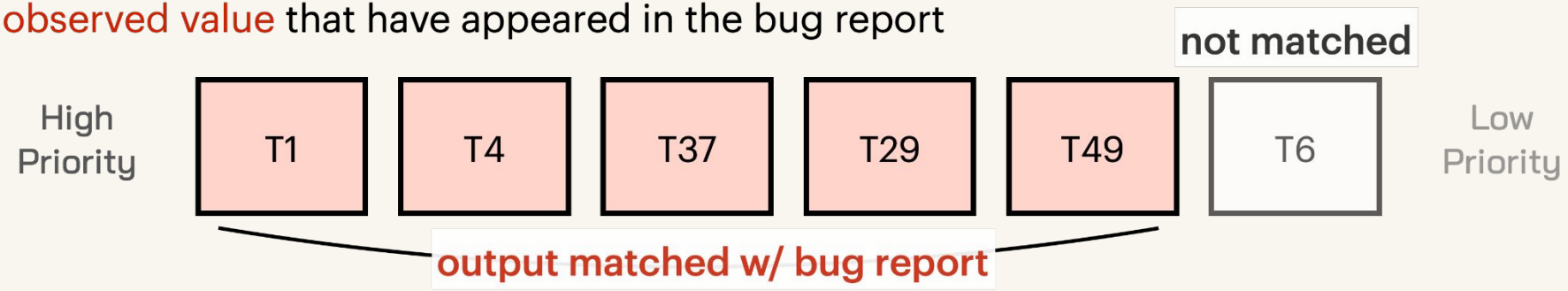
Cluster Size: 2

Accept the generated test set when maximum cluster size > THRESHOLD

Ranking tests with three heuristics (1)

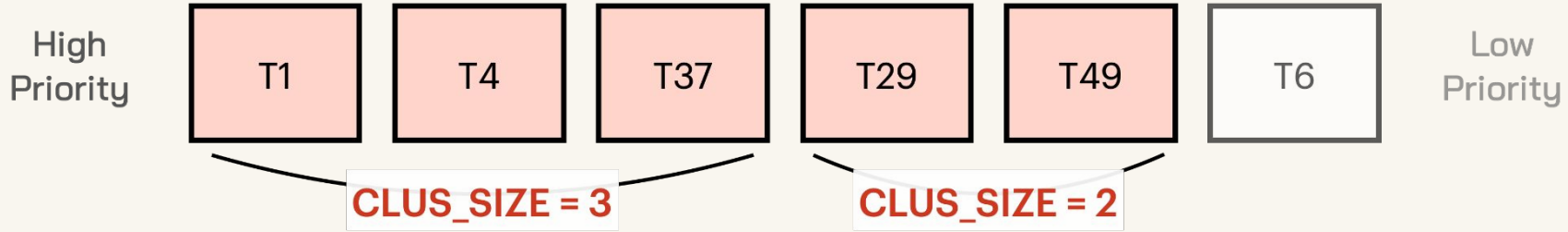
Q. Which test is more likely to be a correct bug reproducing test?

A1. (Matching w/ Bug Report) When test outputs **include** exception type or observed value that have appeared in the bug report



Ranking tests with three heuristics (2)

A2. (Consensus level) Tests from **larger output cluster** are prioritized



Recap

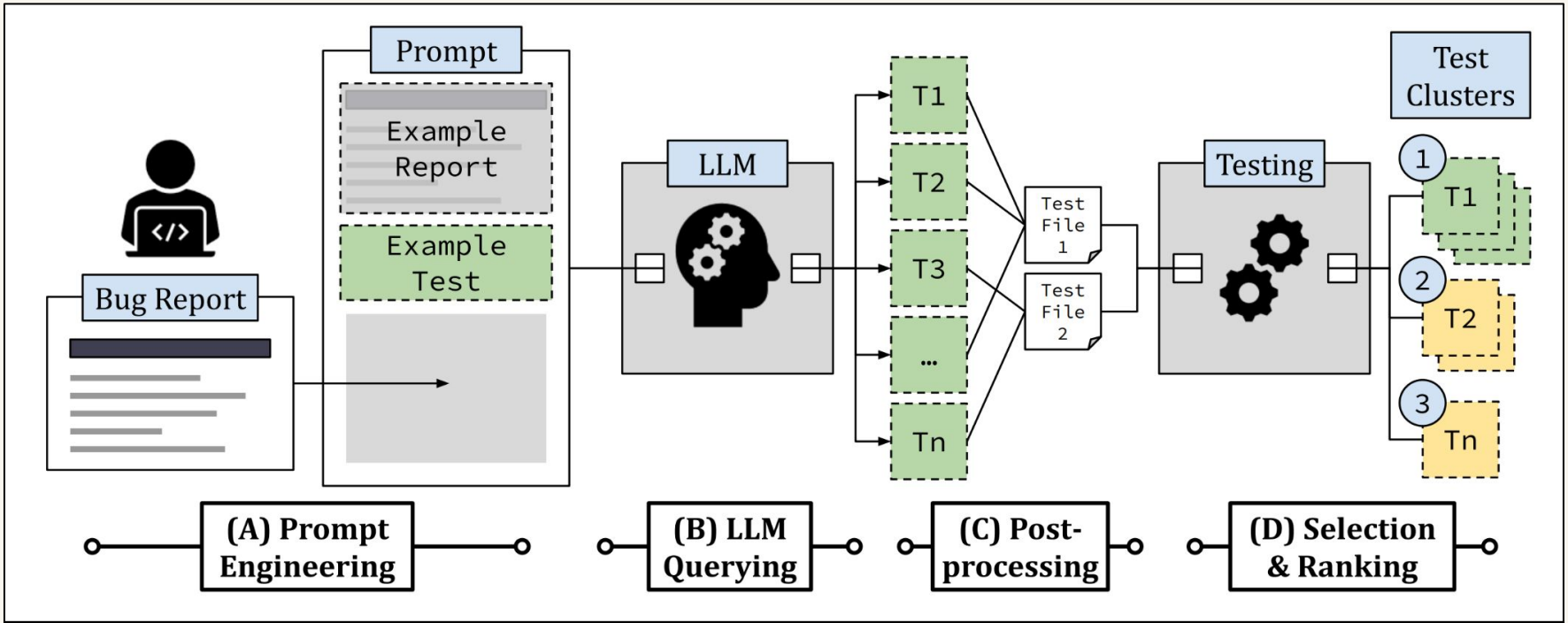
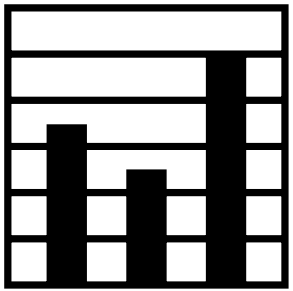


Diagram of LIBRO


Results

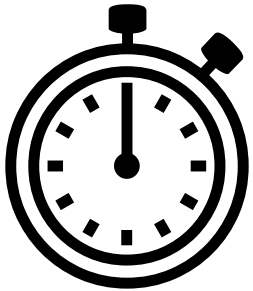
Evaluating the Technique

RQ1: Efficacy



How many bugs reproduced? (D4J)

RQ2: Efficiency



How much resources are required?

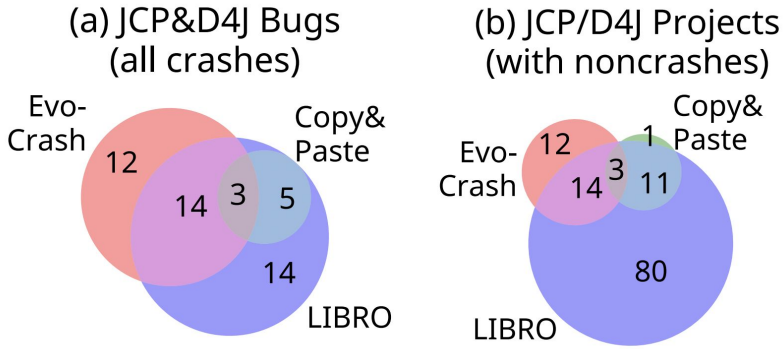
RQ3: Generality

Are results similar on held-out data?

RQ1: Efficacy

Setting	reproduced	FIB
No Example (n=10)	124	440
One Example (n=10)	166	417
One Example from Source Project (n=10)	152	455
One Example with Constructor Info (n=10)	167	430
Two Examples (n=10, 5th percentile)	161	386
Two Examples (n=10, median)	173	409
Two Examples (n=10, 95th percentile)	184	429
Two Examples (n=50)	**251**	**570**
One Example, Crash Bugs (n=10)	69	153
One Example with Stack, Crash Bugs (n=10)	84	155

RQ1-1: One-third of all bugs were successfully reproduced.



RQ1-2: LIBRO significantly outperforms baselines.

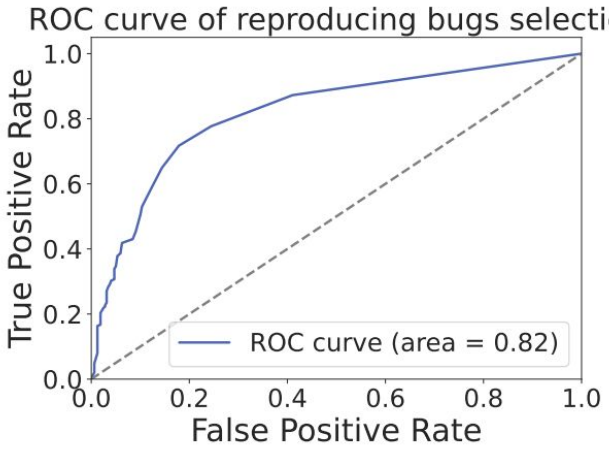
RQ2-2: Time cost of each component

TABLE V: The time required for the pipeline of LIBRO

	Prompt	API	Processing	Running	Ranking	Total
Single Run	<1 μs	5.85s	1.23s	4.00s	-	11.1s
50-test Run	<1 μs	292s	34.8s	117s	0.02s	444s

The API call and actual execution of the test took the longest amount of time.

RQ2-3: Selection performance



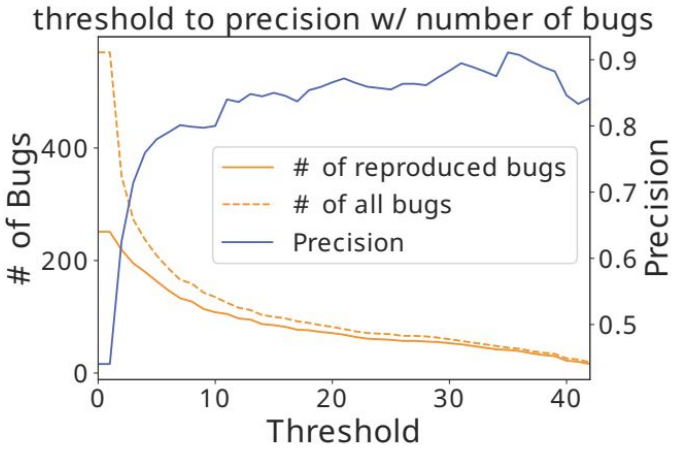


Fig. 4: ROC curve of bug selection (Left), Effect of thresholds to the number of bugs selected and precision (Right)

Selecting by counting the failing-in-buggy tests was effective;

RQ2-3: Ranking performance

TABLE VI: Ranking Performance Comparison between LIBRO and Random Baseline

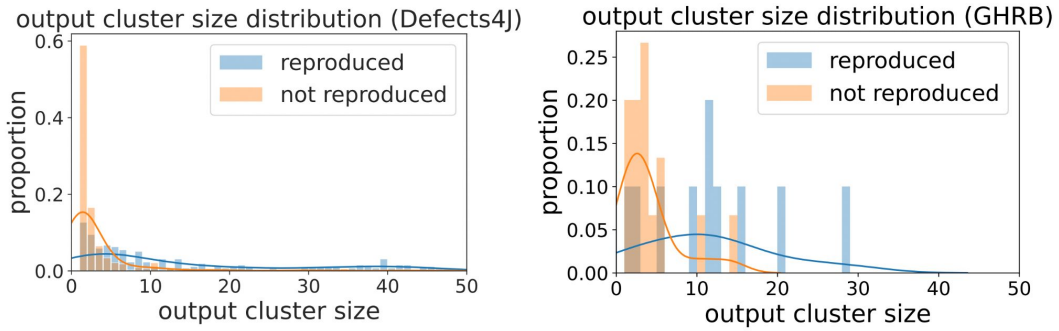
	Defects4J				GHRB			
	$acc@n$ (*precision*)		$wef@n_{agg}$		$acc@n$ (*precision*)		$wef@n_{agg}$	
n	LIBRO	random	LIBRO	random	LIBRO	random	LIBRO	random
1	**149 (0.43)**	116 (0.33)	**201 (0.57)**	234 (0.67)	**6 (0.29)**	4.8 (0.23)	**15 (0.71)**	16.2 (0.77)
3	**184 (0.53)**	172 (0.49)	**539 (1.54)**	599 (1.71)	**7 (0.33)**	6.6 (0.31)	**42 (2.0)**	44.6 (2.12)
5	**199 (0.57)**	192 (0.55)	**797 (2.28)**	874 (2.5)	**8 (0.38)**	7.3 (0.35)	**60 (2.86)**	64.3 (3.06)

Our within-FIB ranking technique was better than the random baseline.

RQ3: Generality

Project	rep/total	Project	rep/total	Project	rep/total
AssertJ	3/5	Jackson	0/2	Gson	4/7
checkstyle	0/13	Jsoup	2/2	sslcontext	1/2

RQ3-1: LIBRO demonstrating it works outside of its (potential) training data.



RQ3-2: LIBRO selection and ranking heuristics generalize to novel data as well.

Successful Example

DateTimeZone.forOffsetHoursMinutes cannot handle negative offset < 1 hour

`DateTimeZone.forOffsetHoursMinutes(h, m)` cannot handle negative offset < 1 hour like -0:30 due to argument range checking. I used `forOffsetMillis()` instead.
This should probably be mentioned in the documentation or negative minutes be accepted.

Bug Report: Joda-Time BUG #8

```
public void testIssue() {
    DateTimeZone tz =
DateTimeZone.forOffsetHoursMinutes(0, -30);
    assertNotNull(tz);
}
```

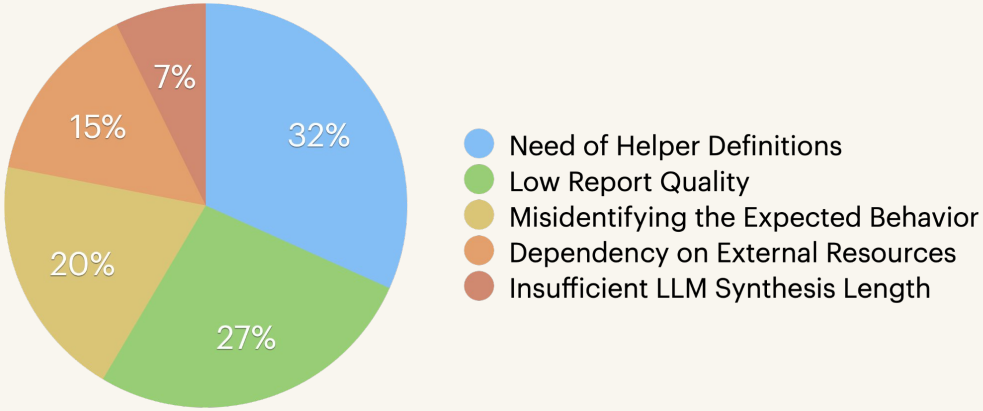
A test generated by LIBRO

```
public void testForOffsetHoursMinutes_int_int() {
    ...
    assertEquals(DateTimeZone.forID("-00:15"),
DateTimeZone.forOffsetHoursMinutes(0, -15));
    ...
}
```

A developer-written test

Examples and Failure Analysis

- When failures happened, we find that
  - 32.5% are due to a need of complex helper functions;
  - 27.5% are due to low report quality;
  - 20% are due to LLM misunderstanding of report;
  - 15% are due to dependency on external resources;
  - 7.5% are due to LLM synthesis limit (we set 256 tokens, or ~1000 characters).



Conclusion

Read our preprint!

1. We tackle the problem of **reproducing general bugs from reports.**

2. We propose LIBRO, which **combines LLMs and postprocessing** to effectively reproduce bug reports.

3. Our evaluation shows **LIBRO successfully reproduces bugs**, and that its postprocessing heuristics work.

Contact us at sungmin.kang@kaist.ac.kr / juyeon.yoon@kaist.ac.kr
Find our preprint with the QR code above, or by searching for “Exploring LLM-based General Bug Reproduction”

Extra Slides

Example 1 - Need of Helper Functions

- When failures happened, we find that
  - **32.5% are due to a need of complex helper functions;**
  - 27.5% are due to low report quality;
  - 20% are due to LLM misunderstanding of report;
  - 15% are due to dependency on external resources;
  - 7.5% are due to LLM synthesis limit (we set 256 tokens, or ~1000 characters).

Invalid left-hand side of assignment not detected

What steps will reproduce the problem?
Compile this: `var x=0,y=1;x||y=8`

What is the expected/actual output?
I expect an error, because this is parsed as `(x||y)=8`, which is an invalid left-hand side of an assignment. Instead, I get `var x=0,y=1;x||y=8;`

Bug Report: Google Closure BUG #84

```
public void testIssue108() {
    try {
        ScriptEngine se = new
ScriptEngineManager().getEngineByName("JavaScript");
        se.eval("var x=0,y=1;x||y=8");
        fail("IllegalArgumentException was expected");
    } catch (ScriptException e) {
        // ok
    }
}
```

Fails to perfectly reproduce the specialized test helper method (here, `parseError`)

A test generated by LIBRO(not bug-reproducing)

Example 1 - Need of Helper Functions

- When failures happened, we find that
  - **32.5% are due to a need of complex helper functions;**
  - 27.5% are due to low report quality;
  - 20% are due to LLM misunderstanding of report;
  - 15% are due to dependency on external resources;
  - 7.5% are due to LLM synthesis limit (we set 256 tokens, or ~1000 characters).

Invalid left-hand side of assignment not detected

What steps will reproduce the problem?
Compile this: `var x=0,y=1;x||y=8`

What is the expected/actual output?
I expect an error, because this is parsed as `(x||y)=8`, which is an invalid left-hand side of an assignment. Instead, I get `var x=0,y=1;x||y=8;`

Bug Report: Google Closure BUG #84

```java
/**
 * Verify that the given code has the given parse errors.
 * @return If in IDE mode, returns a partial tree.
 */
private Node parseError(String string, String... errors) {
  TestErrorReporter testErrorReporter = new TestErrorReporter(errors, null);
  Node script = null;
  try {
    StaticSourceFile file = new SimpleSourceFile("input", false);
    script = ParserRunner.parse(
        file, string, ParserRunner.createConfig(isIdeMode, mode, false),
        testErrorReporter, Logger.getAnonymousLogger()).ast;
  } catch (IOException e) {
    throw new RuntimeException(e);
  }
```

Example 2 - Low Report Quality

- When failures happened, we find that
  - 32.5% are due to a need of complex helper functions;
  - **27.5% are due to low report quality;**
  - 20% are due to LLM misunderstanding of report;
  - 15% are due to dependency on external resources;
  - 7.5% are due to LLM synthesis limit (we set 256 tokens, or ~1000 characters).

StrBuilder contains usages of thisBuf.length when they should use size

While fixing LANG-294 I noticed that there are two other places in StrBuilder that reference thisBuf.length and unless I'm mistaken they shouldn't.

Bug Report: Defects4J Lang-60

External Dependency

- When failures happened, we find that
  - 32.5% are due to a need of complex helper functions;
  - 27.5% are due to low report quality;
  - 20% are due to LLM misunderstanding of report;
  - **15% are due to dependency on external resources;**
  - 7.5% are due to LLM synthesis limit (we set 256 tokens, or ~1000 characters).

PAX header entry name ending with / causes problems

There seems to be a problem when a PAX header entry (link flag is 'x') has a name ending with "/". The `TarArchiveEntry.isDirectory()` check ends up returning true because of the trailing slash which means no content can be read from the entry (...)

Bug Report: Commons-Compress BUG #38

```
public void testPaxHeaderFails() throws Exception {
    FileInputStream fin = null;
    try {
        fin = new
FileInputStream("/path/to/docker-image-layer.tar");
        TarArchiveInputStream tais = new
TarArchiveInputStream(fin);
    (...)
```

Fails due to missing file;
The generated test is dependent on an external file

A test generated by LIBRO (not bug-reproducing)

Synthesis limit example

- When failures happened, we find that
  - 32.5% are due to a need of complex helper functions;
  - 27.5% are due to low report quality;
  - 20% are due to LLM misunderstanding of report;
  - 15% are due to dependency on external resources;
  - **7.5% are due to LLM synthesis limit (we set 256 tokens, or ~1000 characters).**

````
* unparsable
```
public void testNumberUtilsCreateNumber() {
    assertEquals(Long.valueOf(1L), NumberUtils.createNumber("1l"));
    assertEquals(Long.valueOf(1L), NumberUtils.createNumber("1L"));
    assertEquals(Long.valueOf(1L), NumberUtils.createNumber("+1l"));
    assertEquals(Long.valueOf(1L), NumberUtils.createNumber("+1L"));
    assertEquals(Long.valueOf(-1L), NumberUtils.createNumber("-1l"));
    assertEquals(Long.valueOf(-1L), NumberUtils.createNumber("-1L"));
    assertEquals(Long.valueOf(1L), NumberUtils.createNumber("1"));
    assertEquals(Long.valueOf(1L), NumberUtils.createNumber("L"));
    assertEquals(Long.valueOf(1L), NumberUtils.createNumber("+1"));
    assertEquals(Long.valueOf(1L), NumberUtils.createNumber("
```
````