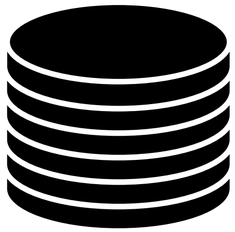
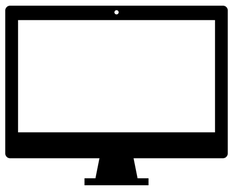
Large Language Models in SE

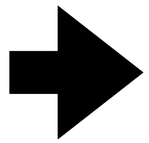
CS454 AI-Based Software Engineering

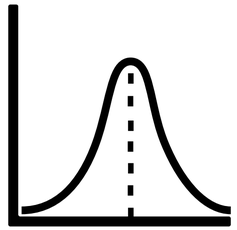
Shin Yoo



Large Training Corpus







Language Model

Probability

Input (text, source code…)

John: Hi, nice to meet you. How are you?

Mary: I’m ____, _____ ___. ___ ___?

a) fine, thank you. And you?

b) okay, I guess. But why?

Python: for _ __ _____ …

a) i in range

b) (int i = 0;

Java: for _ ___ _ _ _ _ …

a) i in range

b) (int i = 0;

```
def SieveOfEratosthenes(num):

  a) prime = [True for i in range(num+1)]…

  b) arr = re.findall(r'[0-9]+', word)…
```

A Thought Experiment

John Searle, “Mind, Brains, and Programs” in 1980

- Suppose we have a computer program that behaves as if it understands Chinese language.
- You are in a closed room with the AI program source code.
- Someone passes a paper with Chinese characters written on it, into the room.
- You use the source code as instruction to generate the response to the input, and sends the response out of the room.
- Do you understand Chinese language, or not?

Stochastic Parrot?

- Among other risks, authors ask whether LLMs actually “understand” anything.
- What do you think?
- The internal design is clearly a statistical language model, i.e., it says what is the most likely, not what is the correct.

Check for updates

On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? 🦜

Emily M. Bender*
ebender@uw.edu
University of Washington
Seattle, WA, USA

Timnit Gebru*
timnit@blackinai.org
Black in AI
Palo Alto, CA, USA

Angelina McMillan-Major
aymm@uw.edu
University of Washington
Seattle, WA, USA

Shmargaret Shmitchell
shmargaret.shmitchell@gmail.com
The Aether

ABSTRACT

The past 3 years of work in NLP have been characterized by the development and deployment of ever larger language models, especially for English. BERT, its variants, GPT-2/3, and others, most recently Switch-C, have pushed the boundaries of the possible both through architectural innovations and through sheer size. Using these pretrained models and the methodology of fine-tuning them for specific tasks, researchers have extended the state of the art on a wide array of tasks as measured by leaderboards on specific benchmarks for English. In this paper, we take a step back and ask: How big is too big? What are the possible risks associated with this technology and what paths are available for mitigating those risks? We provide recommendations including weighing the environmental and financial costs first, investing resources into curating and carefully documenting datasets rather than ingesting everything on the web, carrying out pre-development exercises evaluating how the planned approach fits into research and development goals and supports stakeholder values, and encouraging research directions beyond ever larger language models.

CCS CONCEPTS

• **Computing methodologies** → **Natural language processing**.

ACM Reference Format:
Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? 🦜. In *Conference on Fairness, Accountability, and Transparency (FAccT '21), March 3–10, 2021, Virtual Event, Canada.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3442188.3445922

1 INTRODUCTION

One of the biggest trends in natural language processing (NLP) has been the increasing size of language models (LMs) as measured by the number of parameters and size of training data. Since 2018 alone, we have seen the emergence of BERT and its variants [39, 70, 74, 113, 146], GPT-2 [106], T-NLG [112], GPT-3 [25], and most recently Switch-C [43], with institutions seemingly competing to produce ever larger LMs. While investigating properties of LMs and how they change with size holds scientific interest, and large LMs have shown improvements on various tasks (§2), we ask whether enough thought has been put into the potential risks associated with developing them and strategies to mitigate these risks.

We first consider environmental risks. Echoing a line of recent work outlining the environmental and financial costs of deep learning systems [129], we encourage the research community to prioritize these impacts. One way this can be done is by reporting costs and evaluating works based on the amount of resources they consume [57]. As we outline in §3, increasing the environmental and financial costs of these models doubly punishes marginalized communities that are least likely to benefit from the progress achieved by large LMs and most likely to be harmed by negative environmental consequences of its resource consumption. At the scale we are discussing (outlined in §2), the first consideration should be the environmental cost.

Just as environmental impact scales with model size, so does the difficulty of understanding what is in the training data. In §4, we discuss how large datasets based on texts from the Internet overrepresent hegemonic viewpoints and encode biases potentially damaging to marginalized populations. In collecting ever larger datasets we risk incurring documentation debt. We recommend mitigating these risks by budgeting for curation and documentation at the start of a project and only creating datasets as large as can be sufficiently documented.

As argued by Bender and Koller [14], it is important to understand the limitations of LMs and put their success in context. This not only helps reduce hype which can mislead the public and researchers themselves regarding the capabilities of these LMs, but might encourage new research directions that do not necessarily depend on having larger LMs. As we discuss in §5, LMs are not performing natural language understanding (NLU), and only have success in tasks that can be approached by manipulating linguistic form [14]. Focusing on state-of-the-art results on leaderboards without encouraging deeper understanding of the mechanism by which they are achieved can cause misleading results as shown

*Joint first authors

FAccT '21, March 3–10, 2021, Virtual Event, Canada
ACM ISBN 978-1-4503-8309-7/21/03.

Large Language Model

(really, a very large statistical language model)

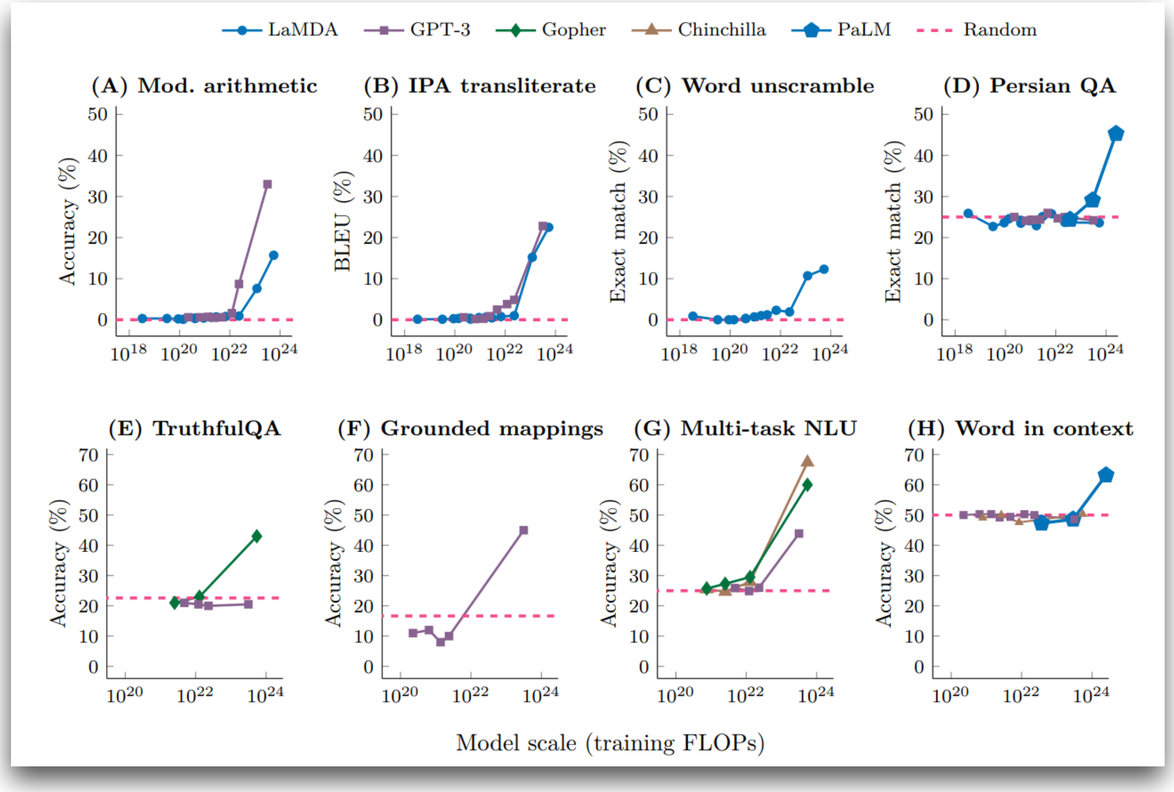
- Mainly Transformer-based DNNs that are trained to be an auto-regressive language model, i.e., given a sequence of tokens, it repeatedly tries to predict the next token.
- The biggest hype in SE research right now with an **explosive** growth, because:
  - **Emergent behaviour** leading to very attractive properties such as in-context learning, Chain-of-Thoughts, or PAL
  - They **seem to** get the **semantics** of the code and **work across natural and programming language**

Further Guides

- Large Language Models for Software Engineering: Survey and Open Problems (https://arxiv.org/abs/2310.03533)
- Large Language Models for Software Engineering: A Systematic Literature Review (https://arxiv.org/abs/2308.10620)
- Software Testing with Large Language Model: Survey, Landscape, and Vision (https://arxiv.org/abs/2307.07221)

Emergent Behavior

- Above certain size, LLMs change their behavior in interesting ways
- The point of change in slope is referred to as “breaks”



Caballero et al., https://arxiv.org/abs/2210.14891

In-context Learning

- Previously, getting a model for a specific task involved either dedicated model + training, or at least general pre-trained model + fine-tuning
- Above certain size, LLMs show the ability to perform in-context learning, i.e., they learn as part of their context (i.e., preceding tokens), leading to **prompt engineering**:
  - Few-shot learning: the context explains the problem, and gives a few examples of question-answer. LLMs can now answer an un-seen question.
  - Zero-shot learning: the context explains the problem as well as how it can be solved. LLMs can now answer an un-seen problem.

A couple of examples

LLMs are Few-shot Testers:

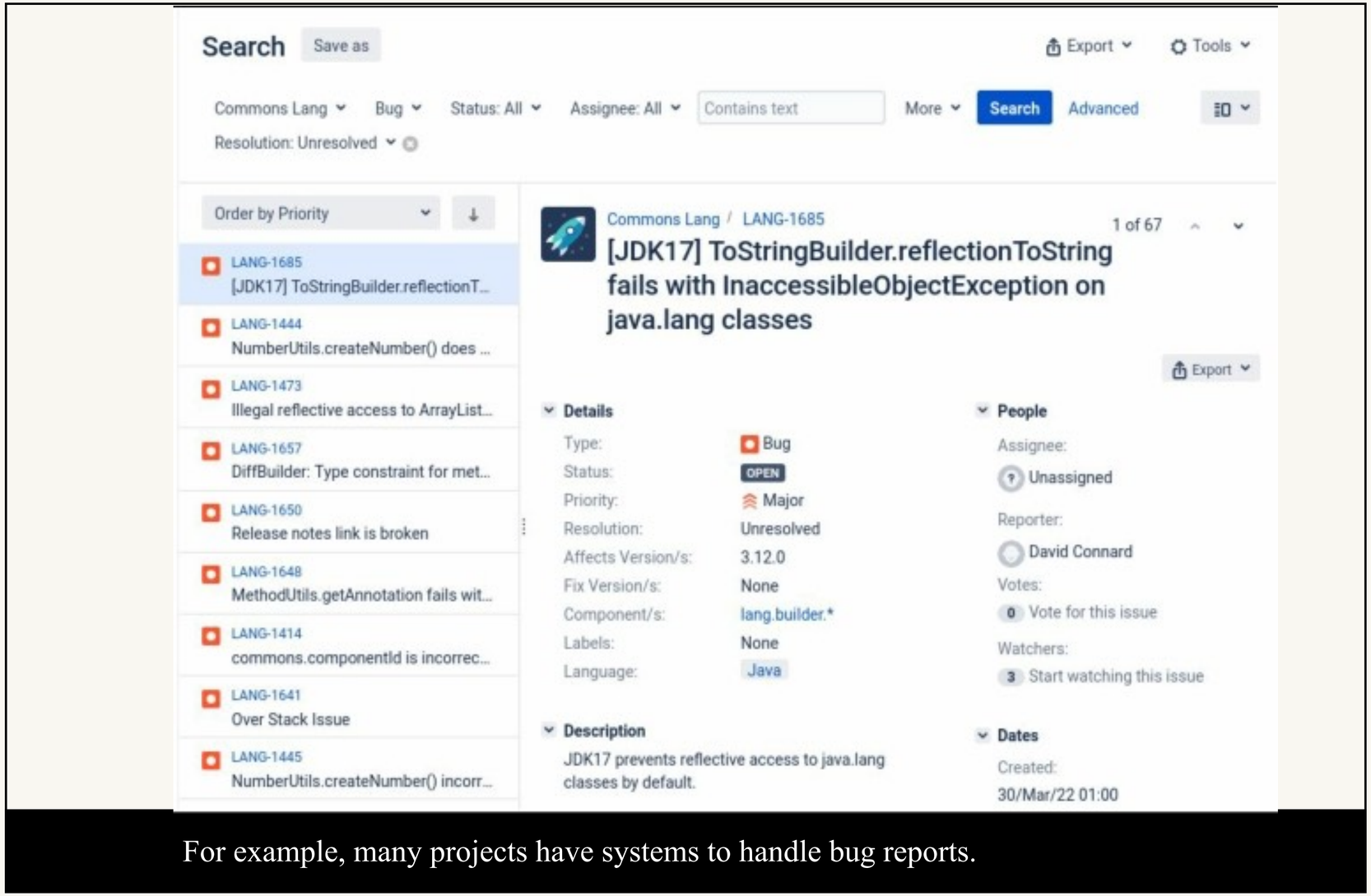
Exploring LLM-based General Bug Reproduction

[Sungmin Kang, Juyeon Yoon], Shin Yoo

Presented on 2023-05-19 by Sungmin


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:

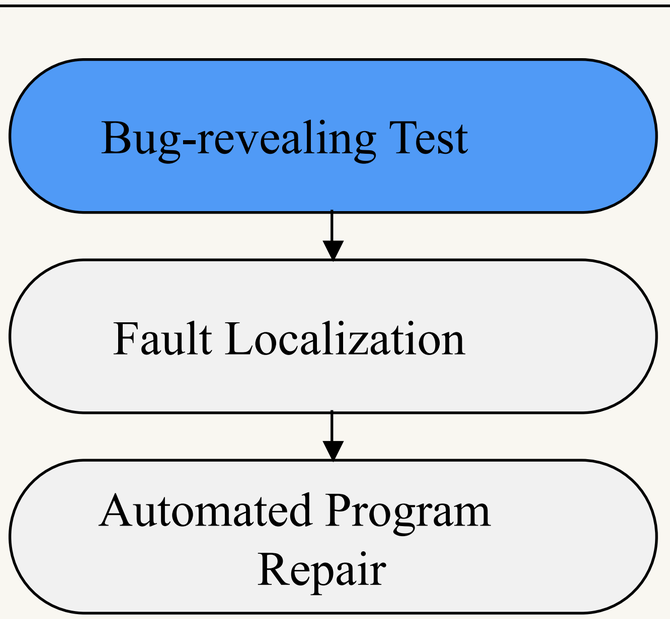
```
String[] sa = ArrayUtils.add(stringArray, aString);
```

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.

```java
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     }
```

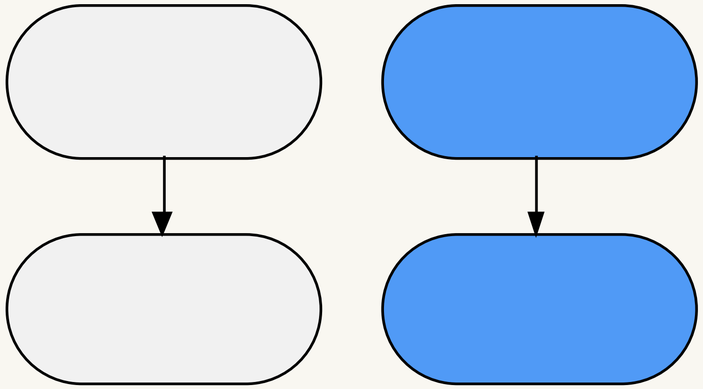
Automatic Bug Reproduction Would Help

1



Reproducing tests are key to automated debugging efficacy.

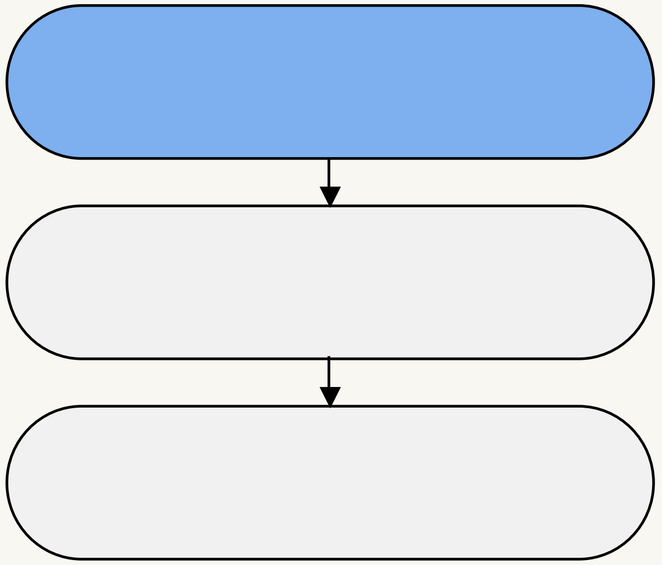
2



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

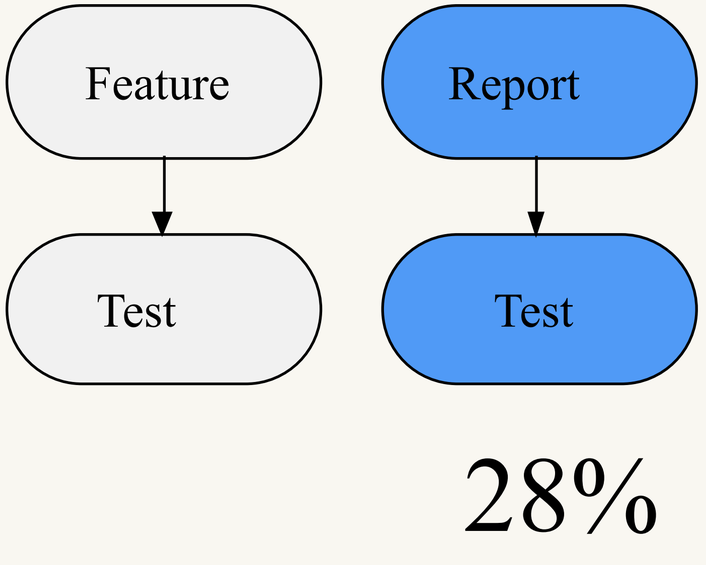
Automatic Bug Reproduction Would Help

1



Reproducing tests are key to automated debugging efficacy.

2



28%

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

runtime.go | course.rb | time.js | IsPrimeTest.java

```java
import static org.junit.Assert.*;
import org.junit.Test;

public class IsPrimeTest {

  // Math.isPrime(int) returns whether the given number is prime or not
  @Test
  public void testIsPrime() {
    assertTrue(Math.isPrime(2));
    assertTrue(Math.isPrime(3));
    assertTrue(Math.isPrime(5));
    assertTrue(Math.isPrime(7));
    assertTrue(Math.isPrime(11));
    assertTrue(Math.isPrime(13));
    assertTrue(Math.isPrime(17));
    assertTrue(Math.isPrime(19));
    assertTrue(Math.isPrime(23));
    assertTrue(Math.isPrime(29));
}
```

Copilot

Replay

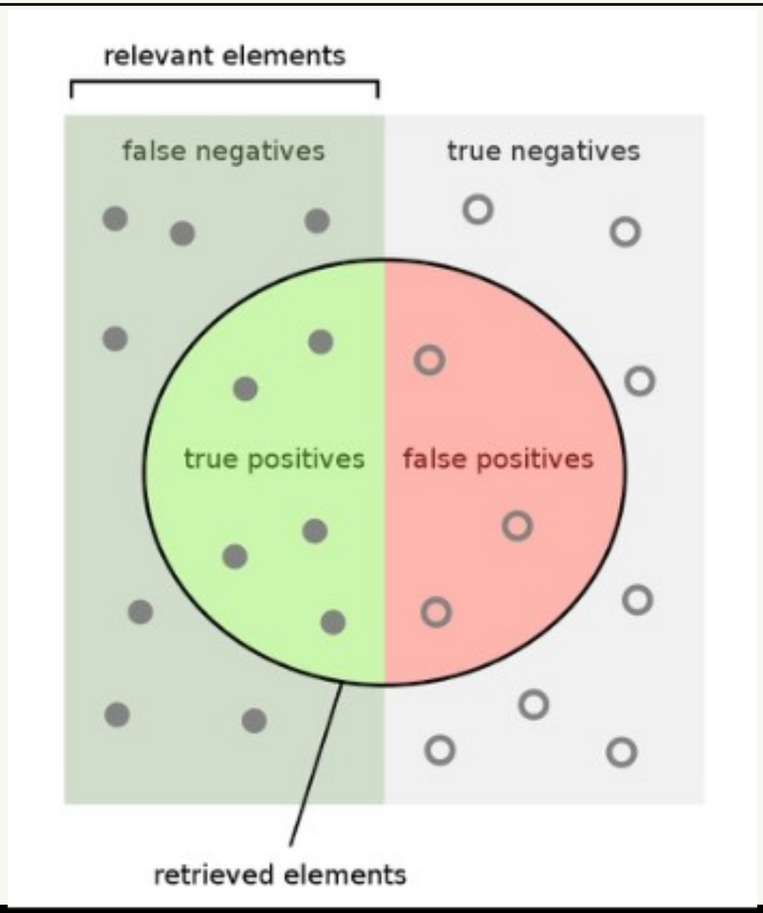
Simple example of test generation from GitHub Copilot page

“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

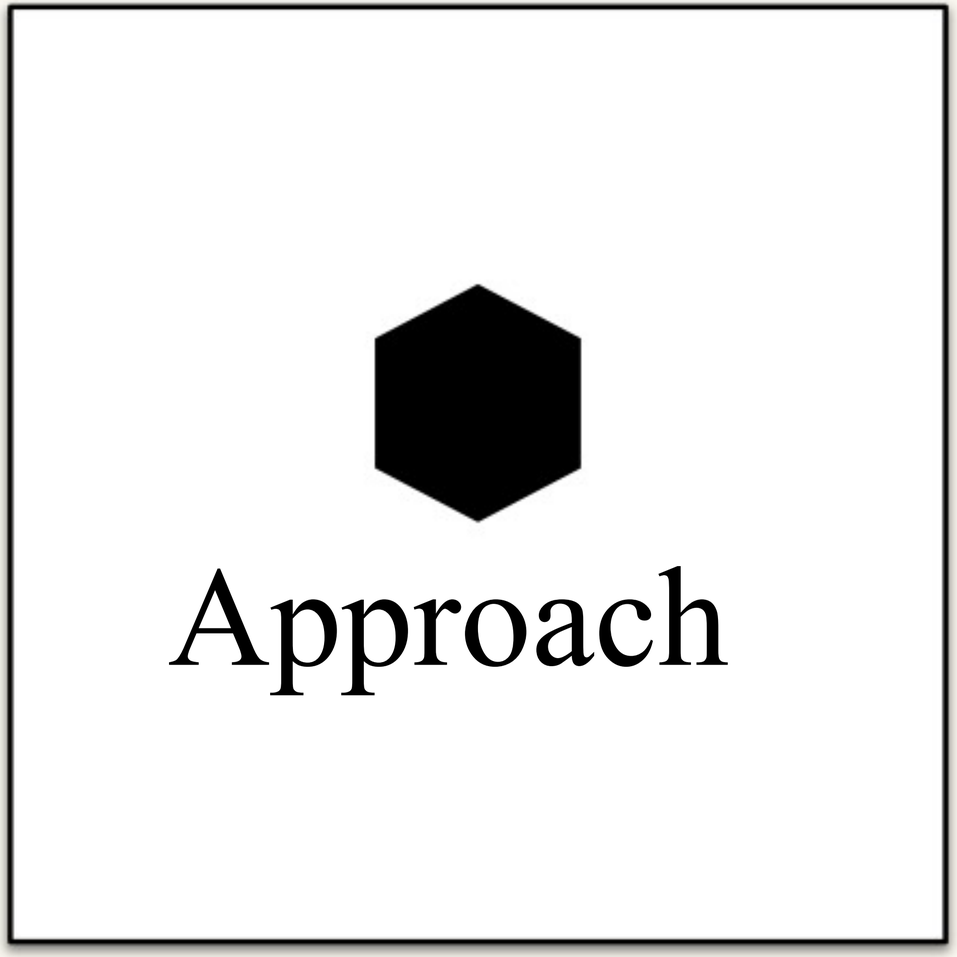
Overall:

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

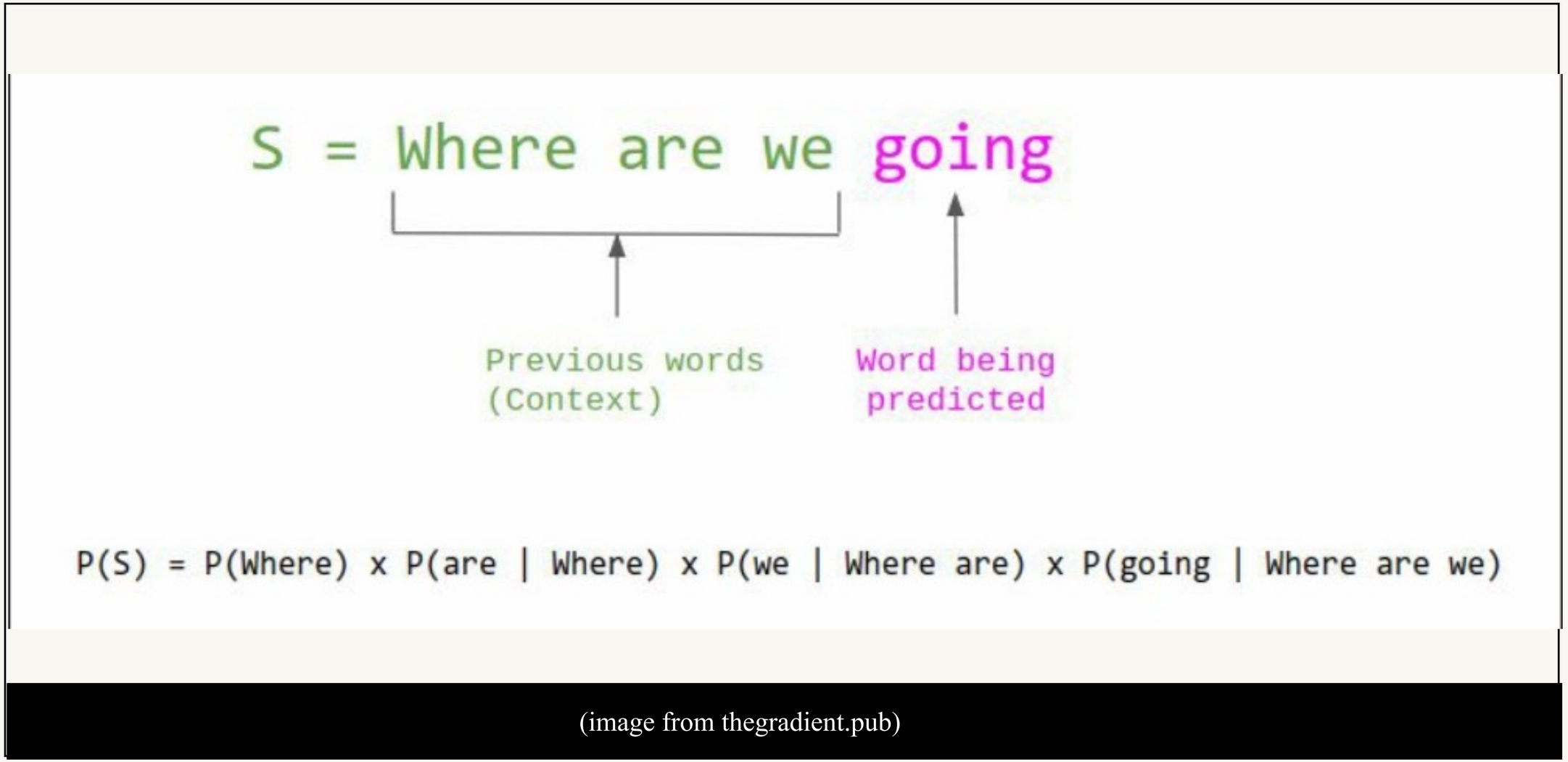
We propose **LIBRO**, which

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



(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.

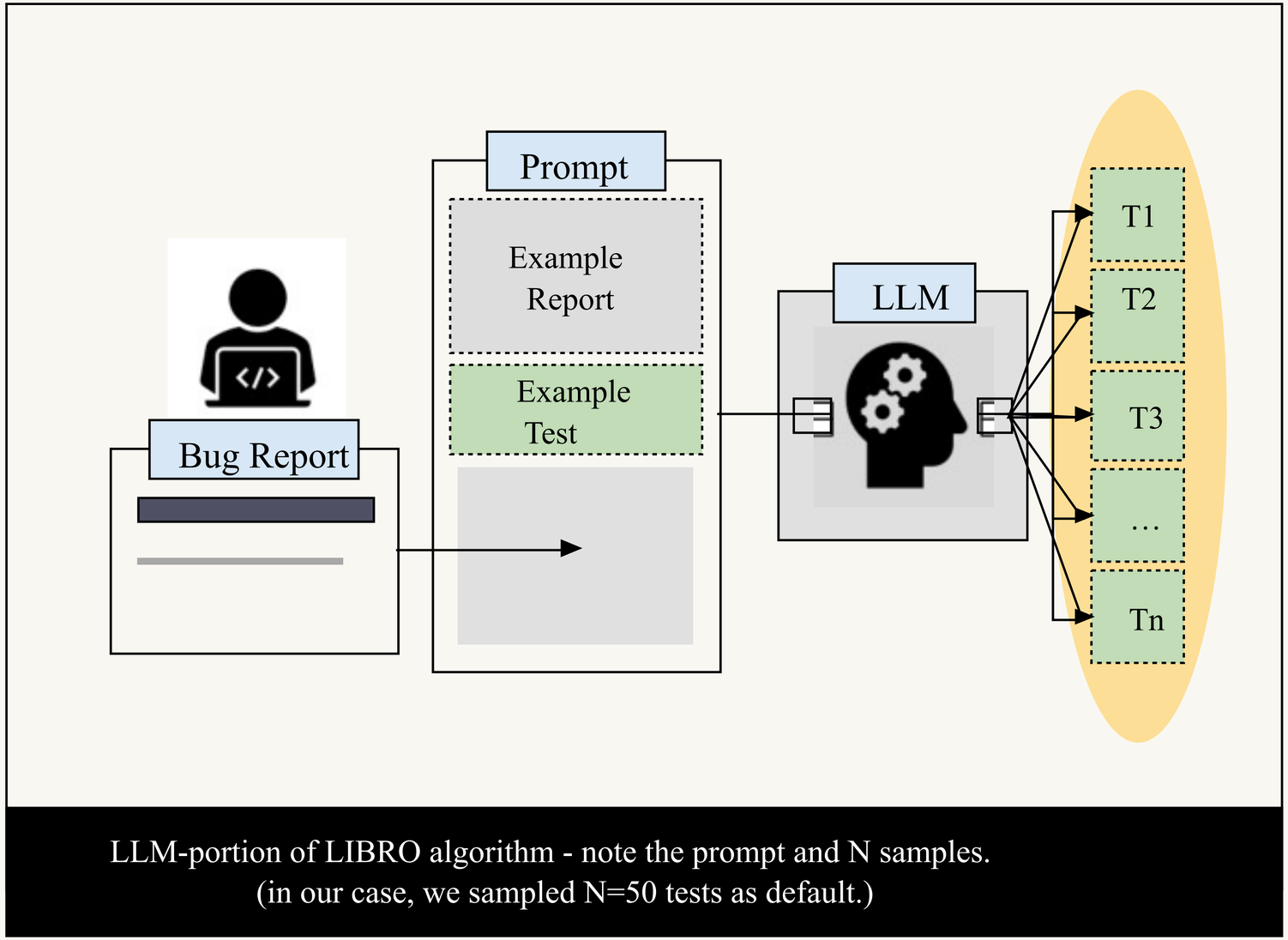
Please let me know if I am doing something wrong here

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.

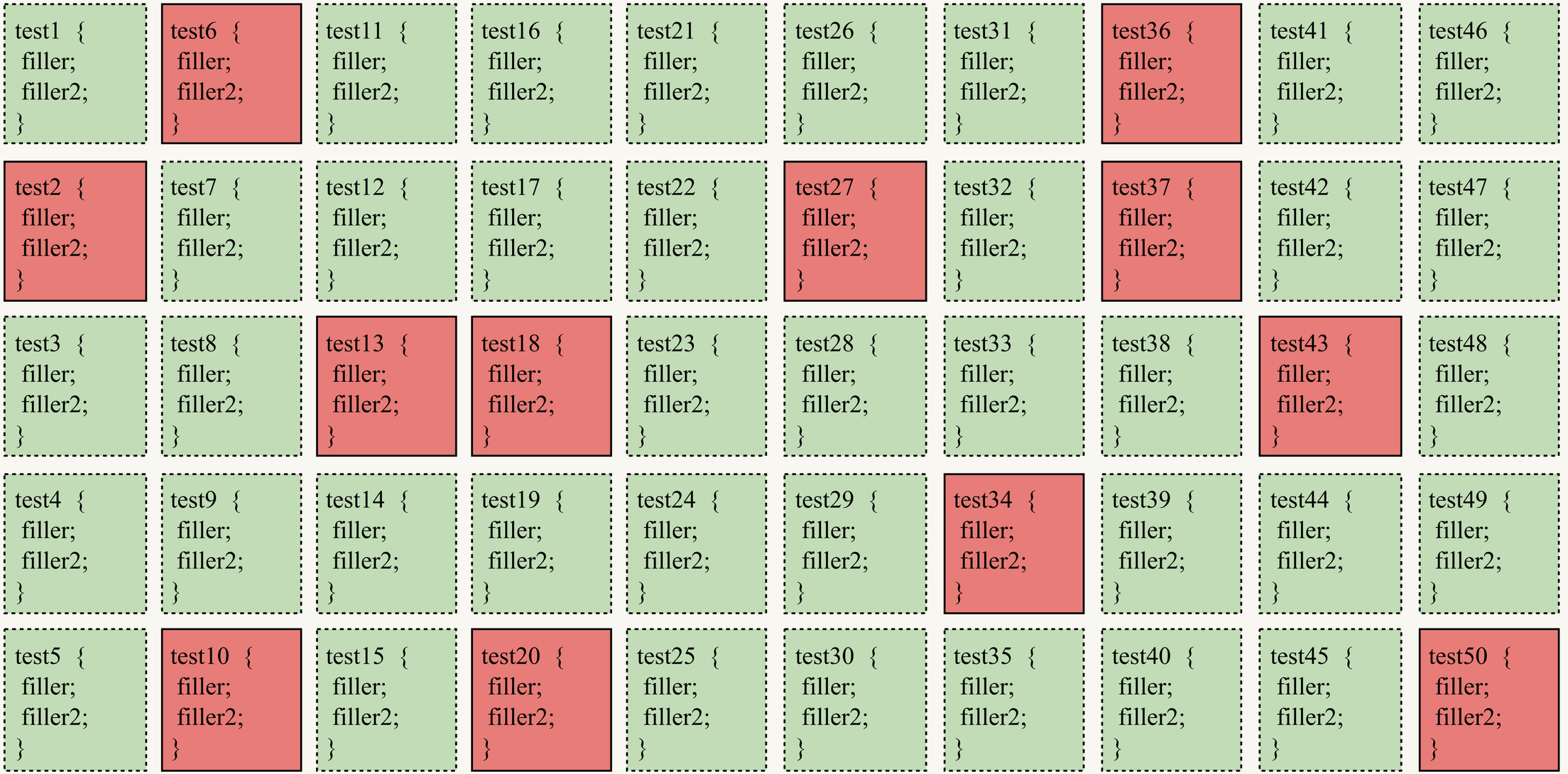


LLM-portion of LIBRO algorithm - note the prompt and N samples.
(in our case, we sampled N=50 tests as default.)

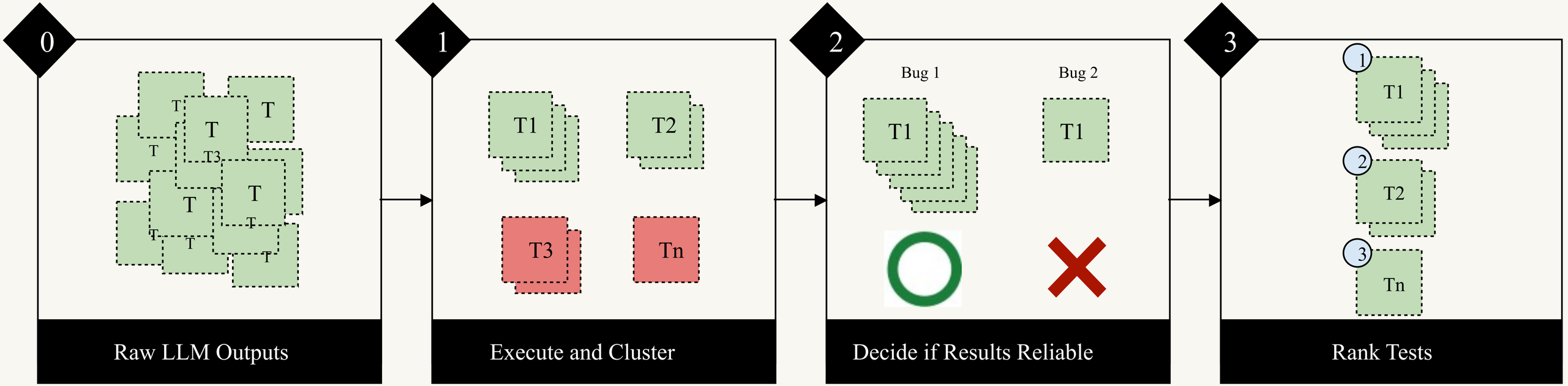
Showing 50 tests is infeasible

```
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

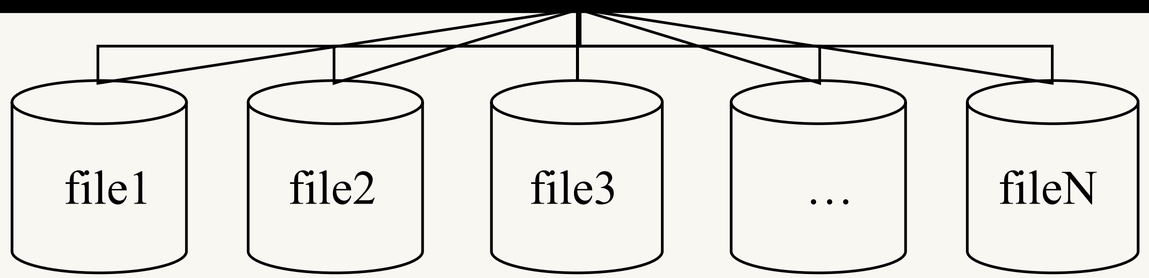


Injecting to target files

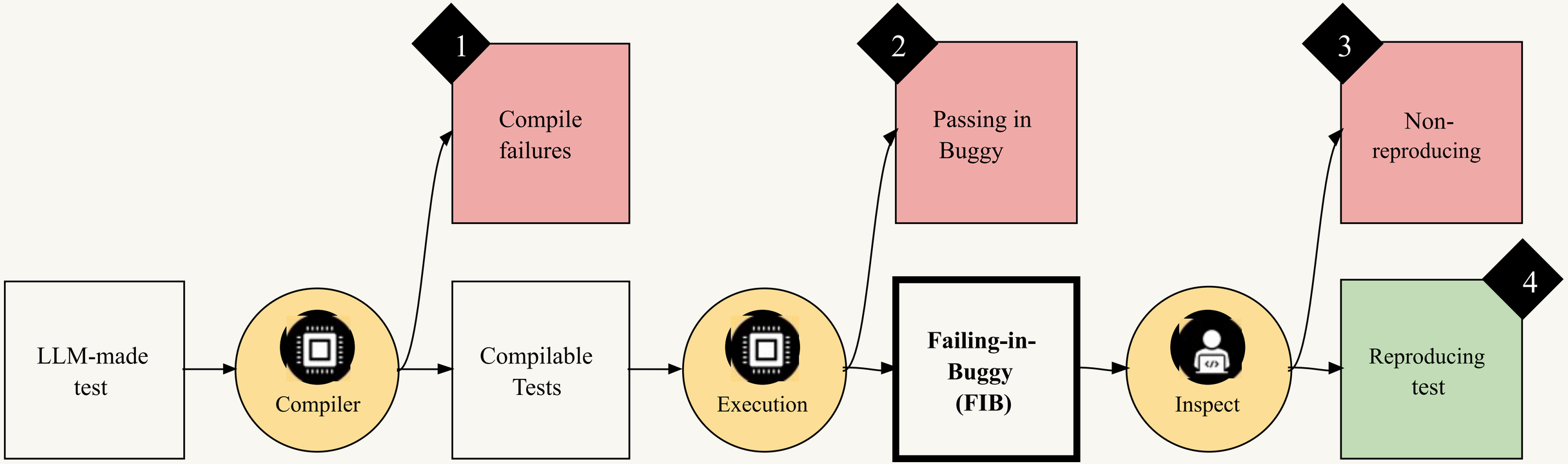
Listing 2: Example LLM result from the bug report described in Table II.

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

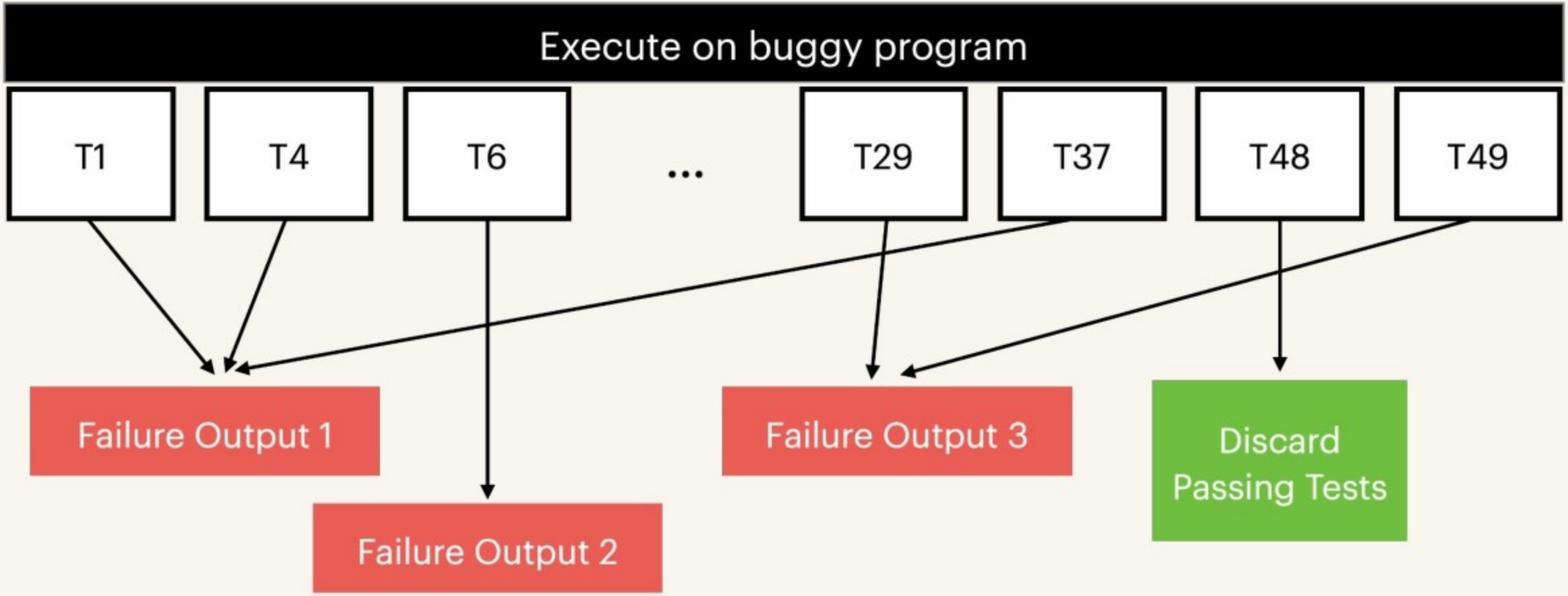
Select the file with greatest lexical similarity and inject the test; add import statements for unmet dependencies.



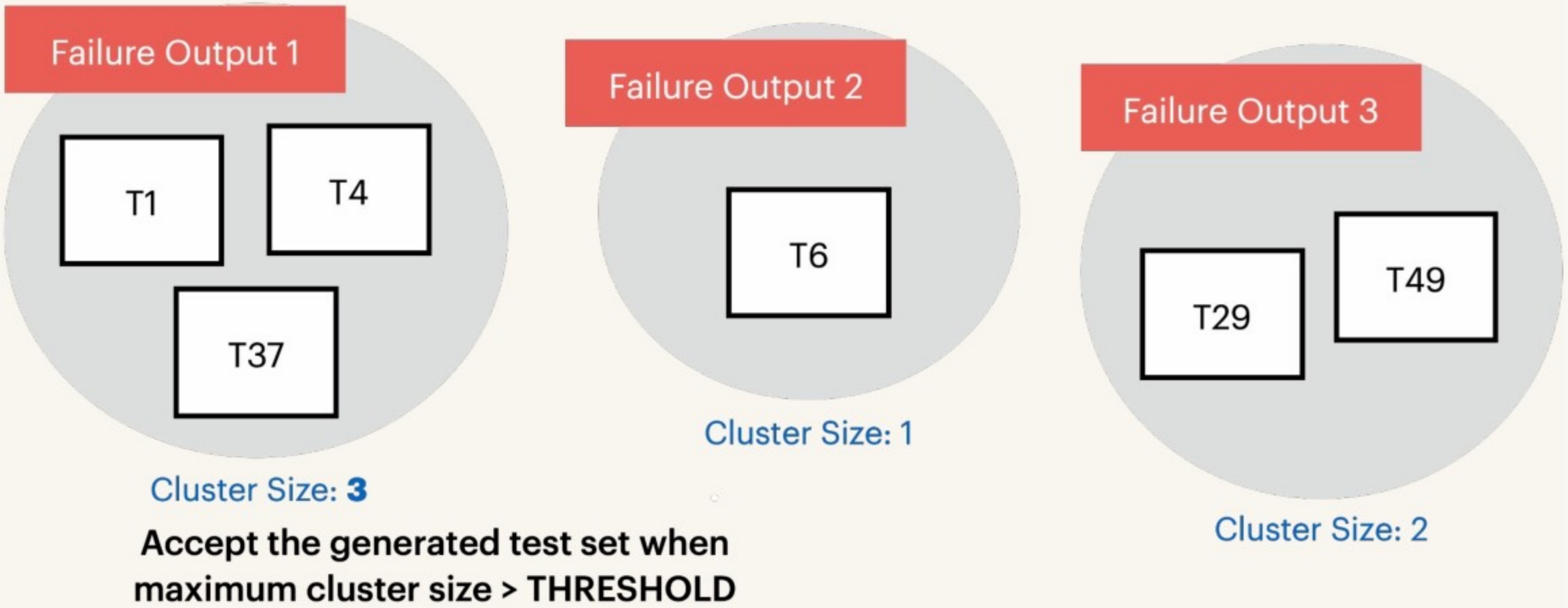
Execute Tests. Four results possible:



Cluster FIB tests with error message



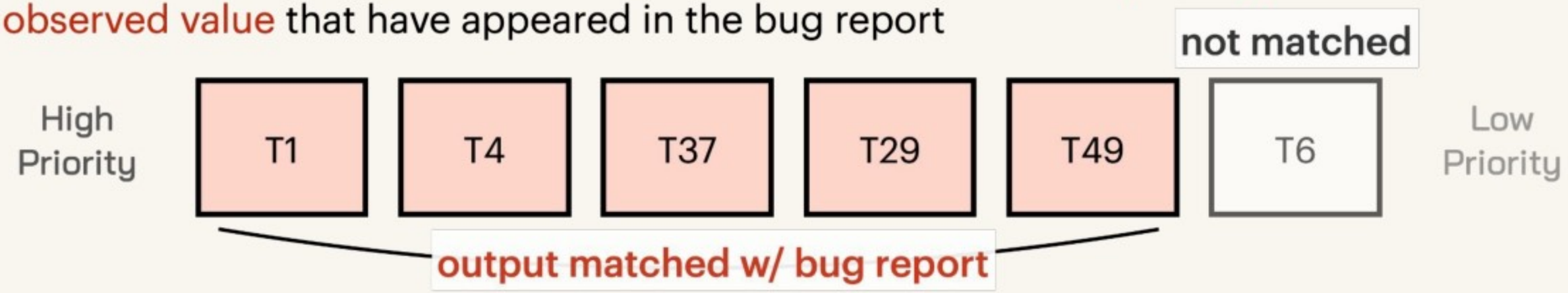
Show results only if cluster size large enough



Ranking tests with three heuristics (1)

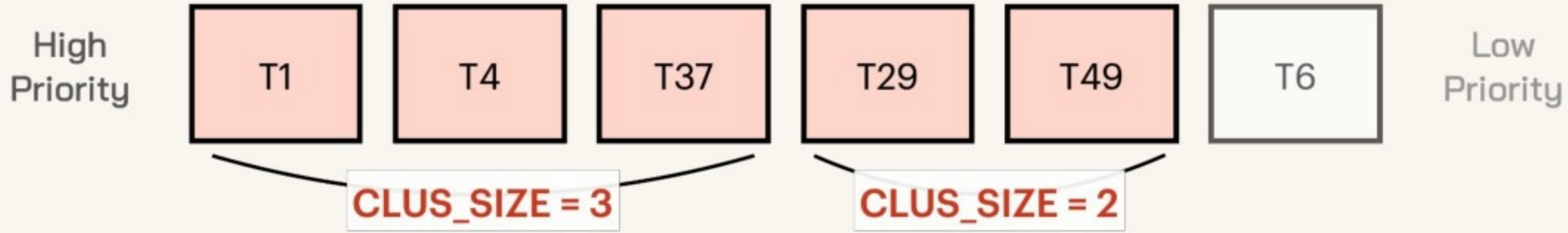
Q. Which test is more likely to be a correct bug reproducing test?

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

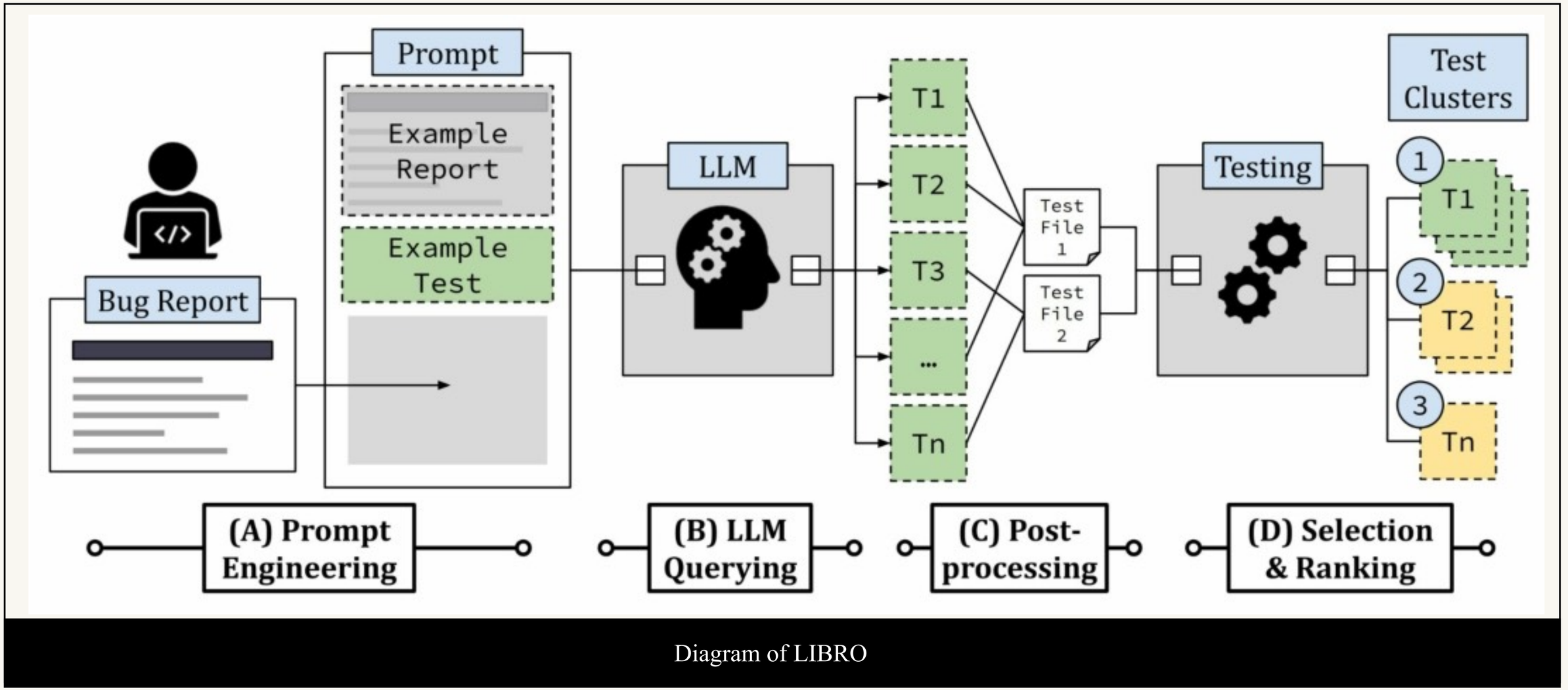


Diagram of LIBRO

Results

Evaluating the Technique

RQ1: Efficacy

How many bugs reproduced? (D4J)

RQ2: Efficiency

How much resources are required?

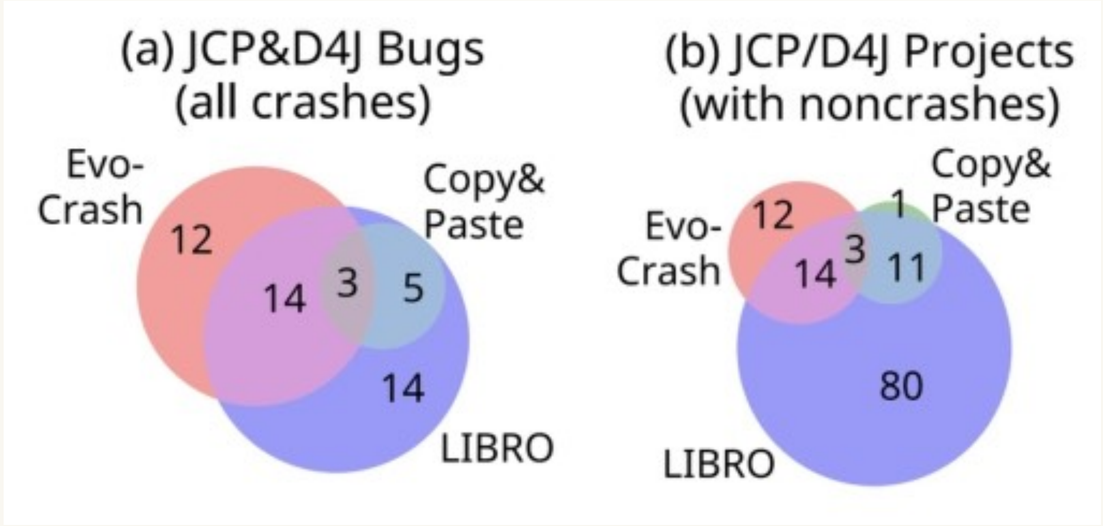
RQ3: Generality

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

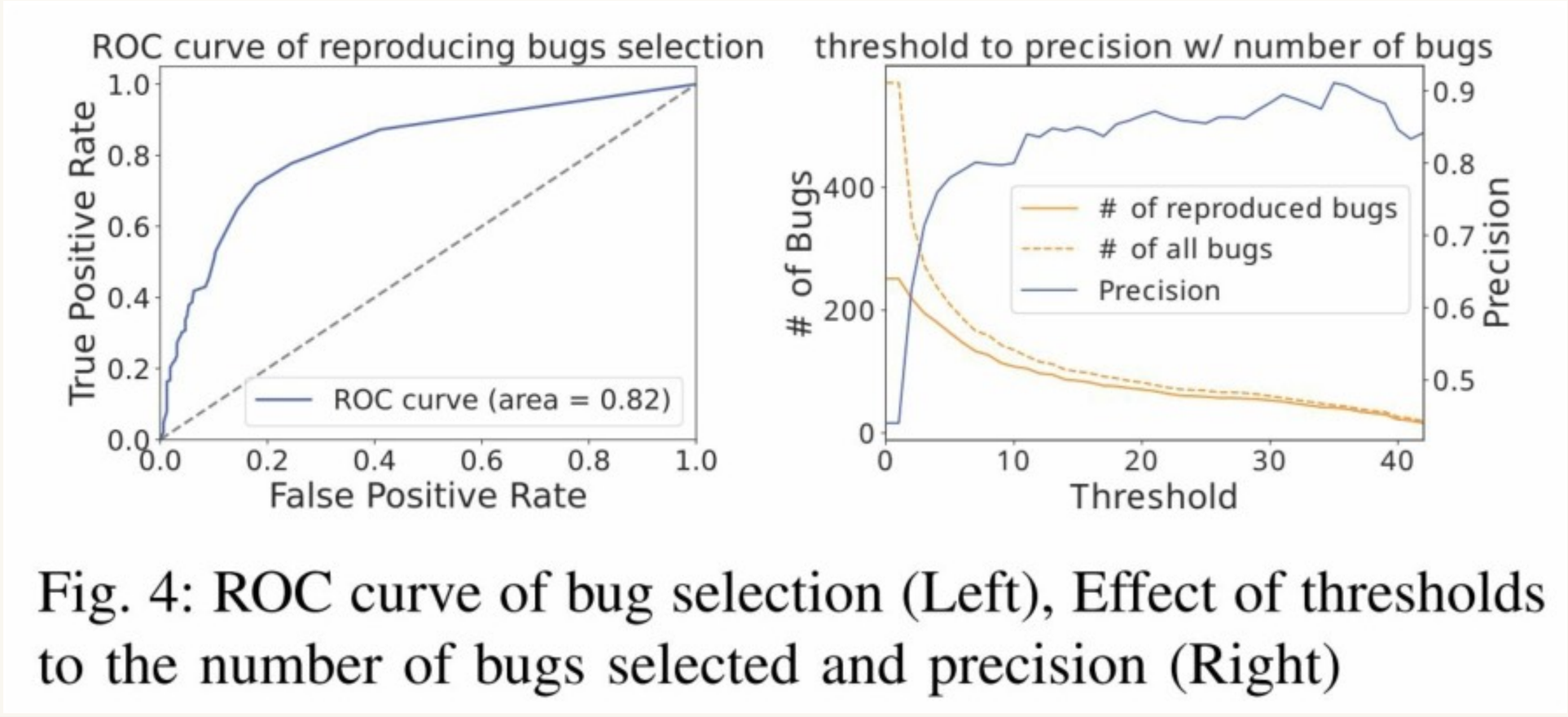


Fig. 4: ROC curve of bug selection (Left), Effect of thresholds to the number of bugs selected and precision (Right)

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

RQ2-3: Ranking performance

TABLE VI: Ranking Performance Comparison between LIBRO and Random Baseline

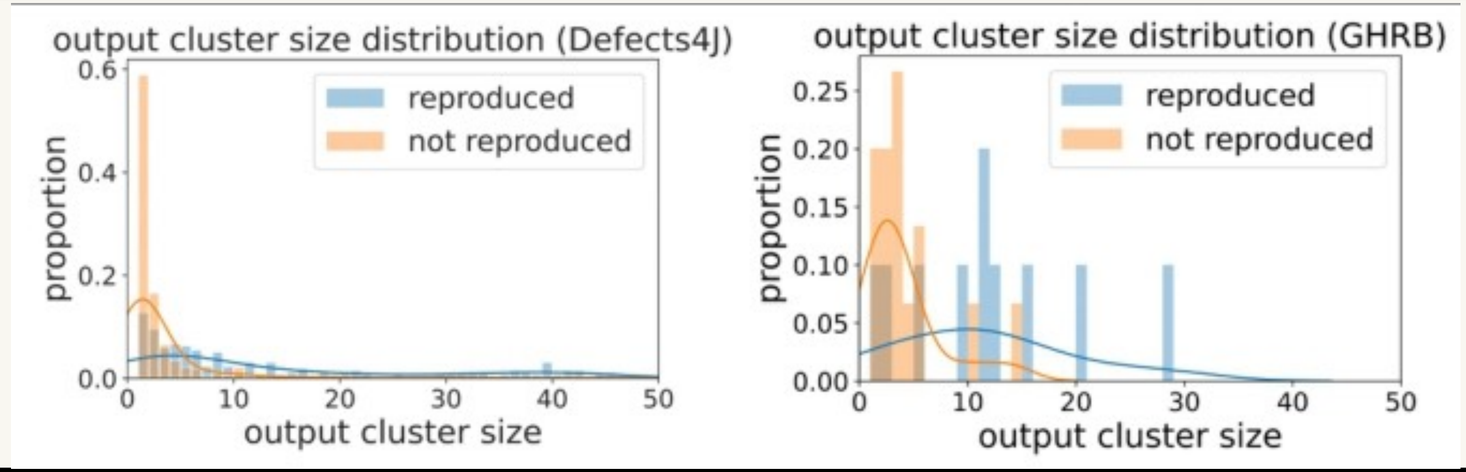
	Defects4J				GHRB			
	$acc@n$ (*precision*)		$wef@n_{agg}$		$acc@n$ (*precision*)		$wef@n_{agg}$	
n	LIBRO	random	LIBRO	random	LIBRO	random	LIBRO	random
1	**149** (**0.43**)	116 (0.33)	**201** (**0.57**)	234 (0.67)	**6** (**0.29**)	4.8 (0.23)	**15** (**0.71**)	16.2 (0.77)
3	**184** (**0.53**)	172 (0.49)	**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);
}
```

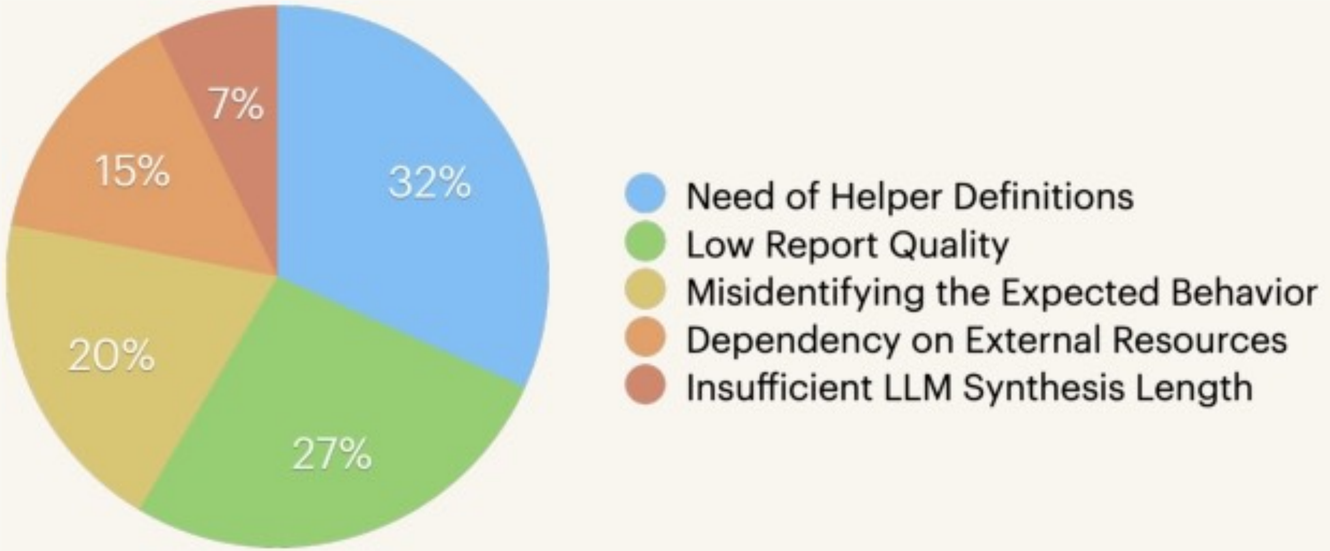
A test generated by LIBRO

```
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.

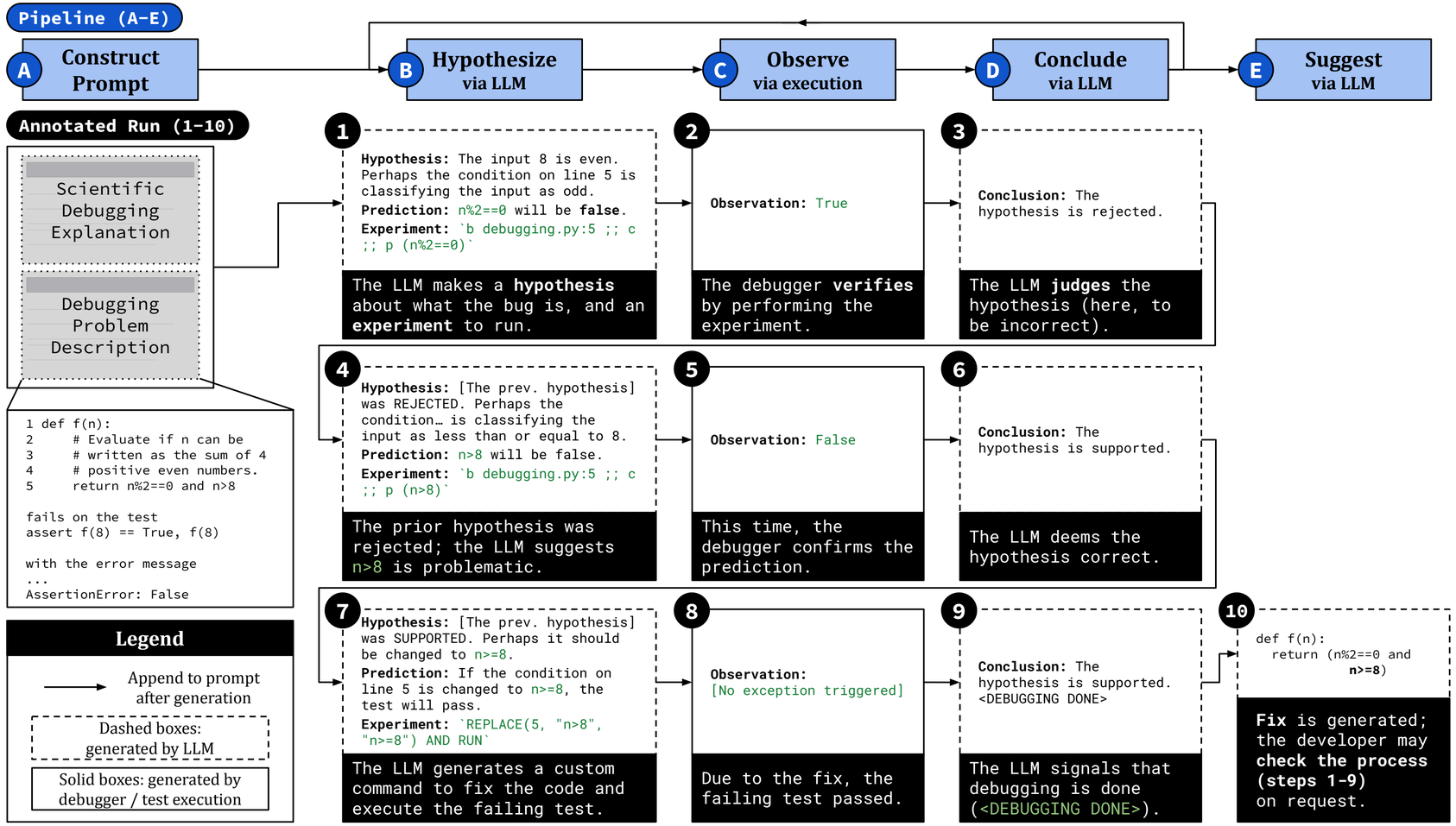
3

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”

Zero-shot Automated Debugging

Kang et al., EMSE 2025 (to appear)



Behind the scene

Chain-of-Thoughts

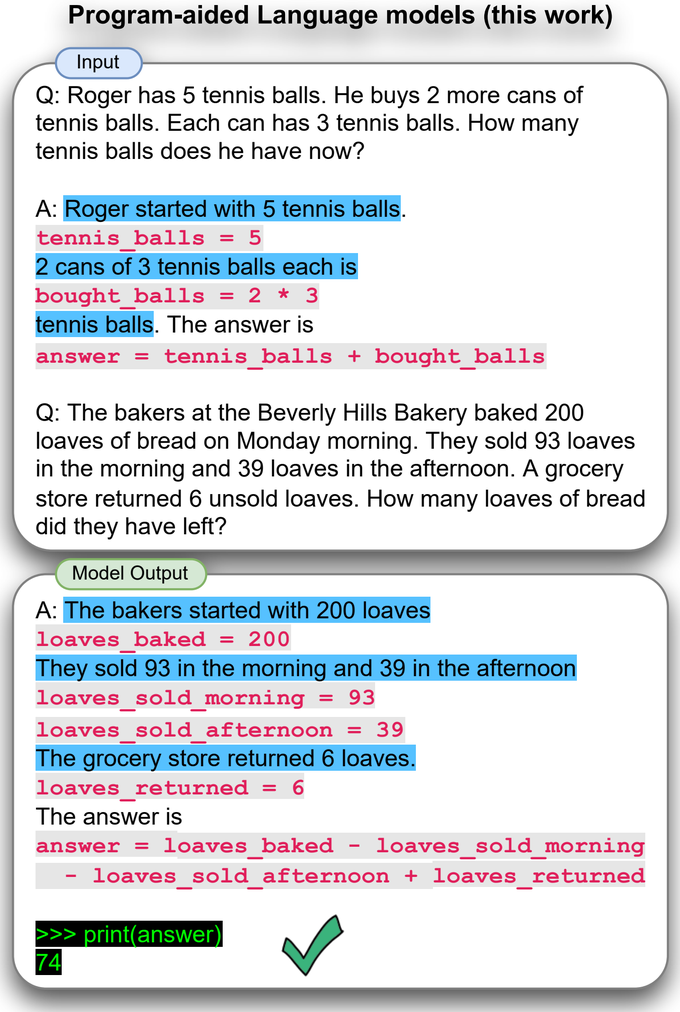
Wei et al., https://arxiv.org/abs/2201.11903

- Underneath, LLMs are doing autocompletion, not any other type of reasoning: they appear to be capable of rational inference because the corpus they are trained include traces of logical reasoning.
- So, conditioning the model (with the context) to be more precise about the reasoning steps can result in generation of more accurate reasoning steps.
  - Add “Let’s think in step by step” at the end of every prompt (https://arxiv.org/abs/2205.11916) 🙃 😐 🫠

Program-Aided Language Models (PAL)

Gao et al., ICML 2023

- What is even more logical and step by step than natural language? Programming language :)
- Providing few-shop examples that are mixtures of NL and LP can enhance the reasoning capabilities of LLM

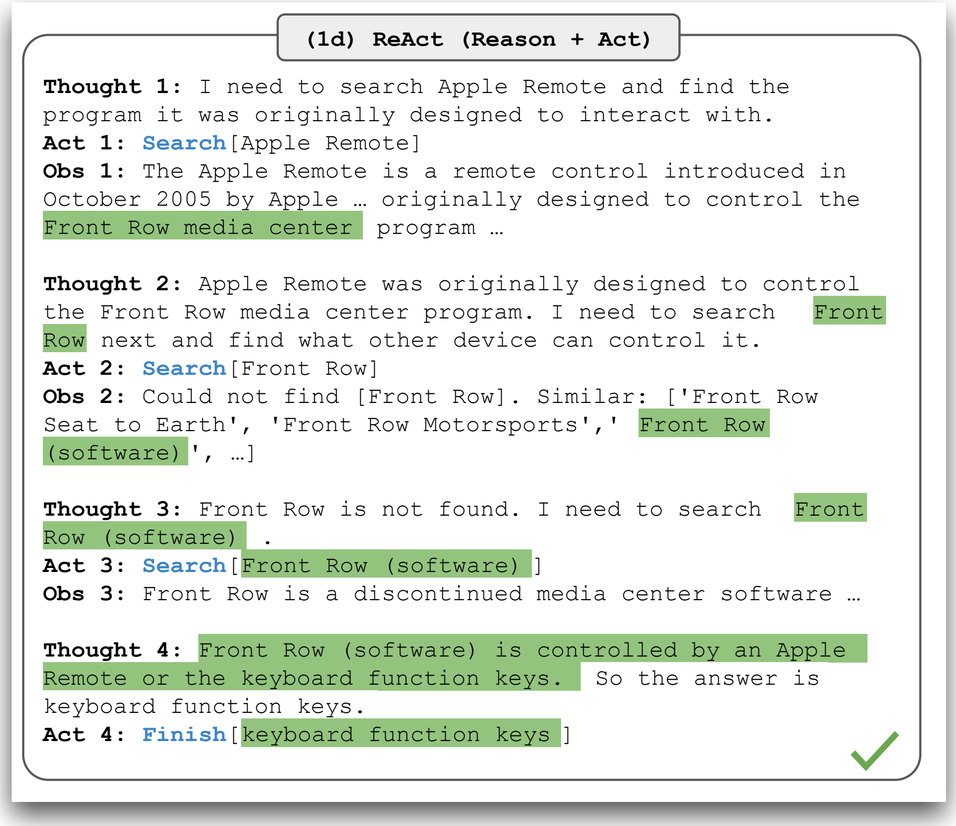


PAL: Program-aided Language Models, Gao et al., ICML 2023
https://arxiv.org/abs/2211.10435

ReAct

Yao et al., ICLR 2023

- What if we need external information for the in-context learning? In other words, can LLMs be given tools?
- Remember that this is still autocompletion:
  - LLMs can be taught to signal the need to invoke tools
  - Whenever LLMs need tool invocation, we can do it ourselves and paste the outcome back into the context

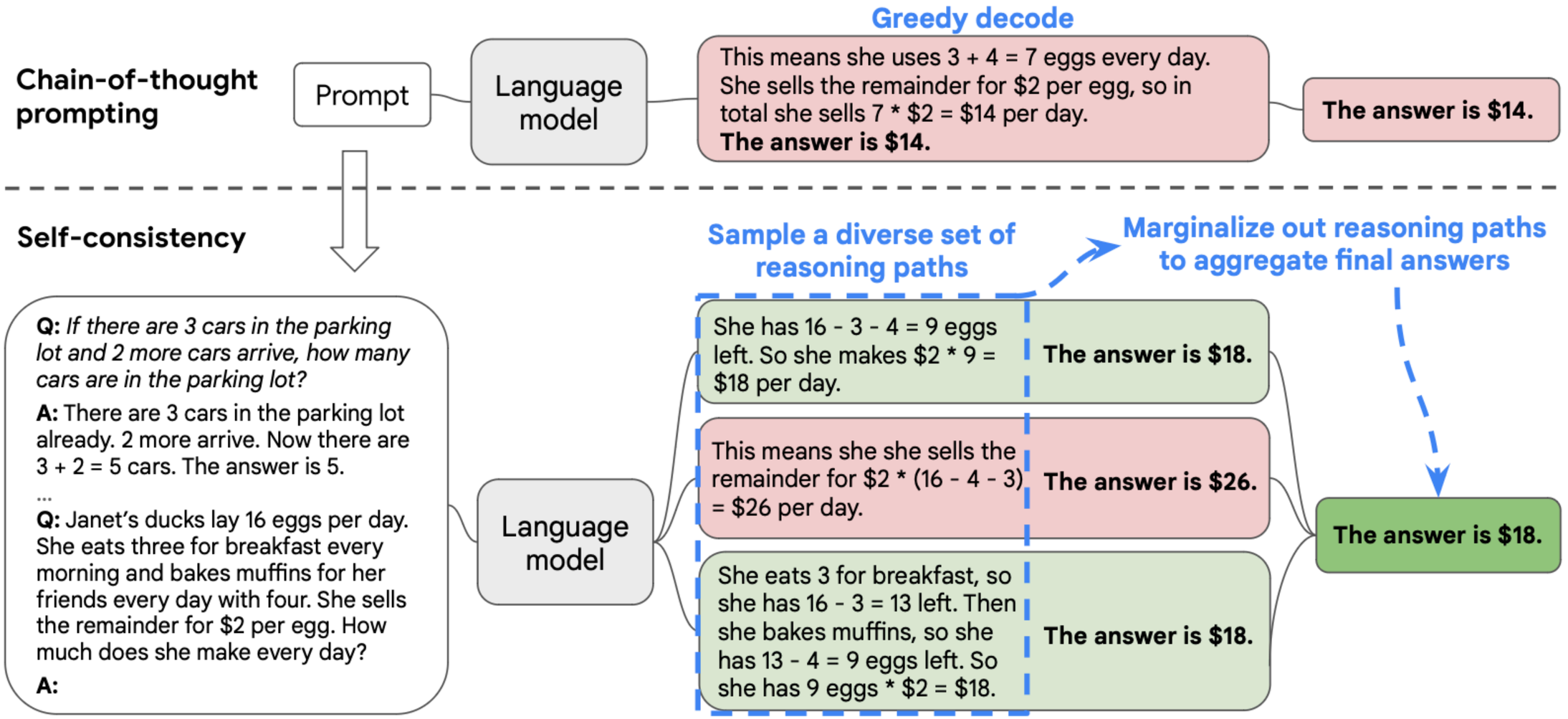


ReAct: Synergizing Reasoning and Acting in Language Models, Yao et al., ICLR 2023
https://arxiv.org/abs/2210.03629

Self Consistency

Wang et al., ICLR 2023 (https://arxiv.org/abs/2203.11171)

- Sample an LLM multiple times for the same question: the majority answer is the most likely to be the correct one!
- Intuitively because: *“we hypothesize that correct reasoning processes, even if they are diverse, tend to have greater agreement in their final answer than incorrect processes”,* i.e., there are multiple reasoning paths to arrive at the correct answer, but fewer ways to arrive at the incorrect one
- Still very early days but: can we connect this to the concept of landscape analysis? Is the correct answer the **highest** (=correct) and also the **biggest** (=the most accessible) hill?

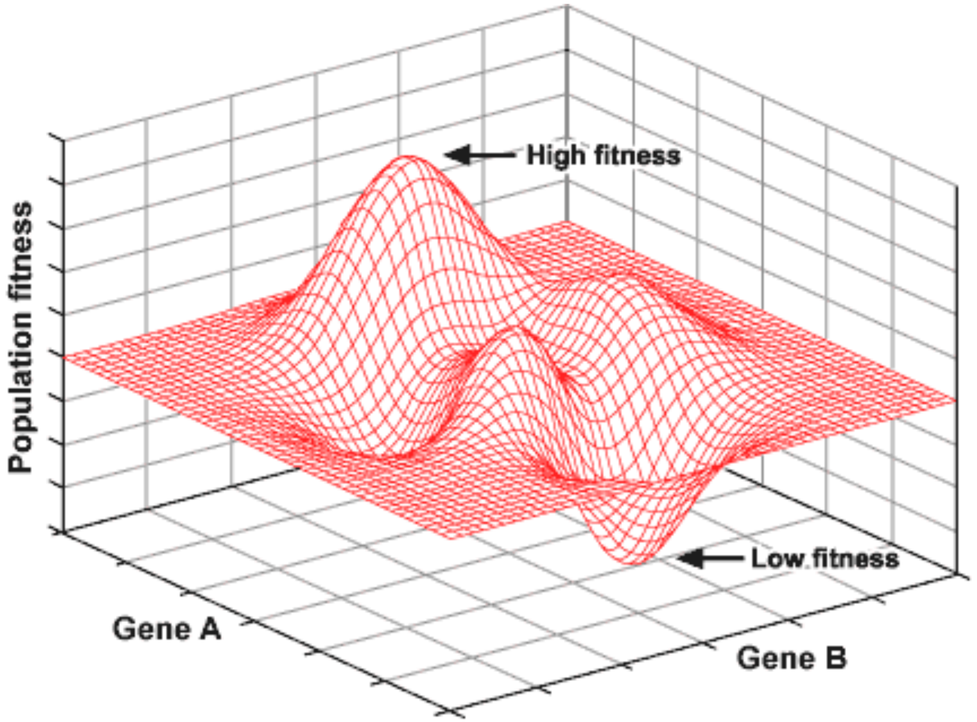


Wang et al., ICLR 2023

LLM Reasoning as Constructive Optimisation

Why does self-consistency work?

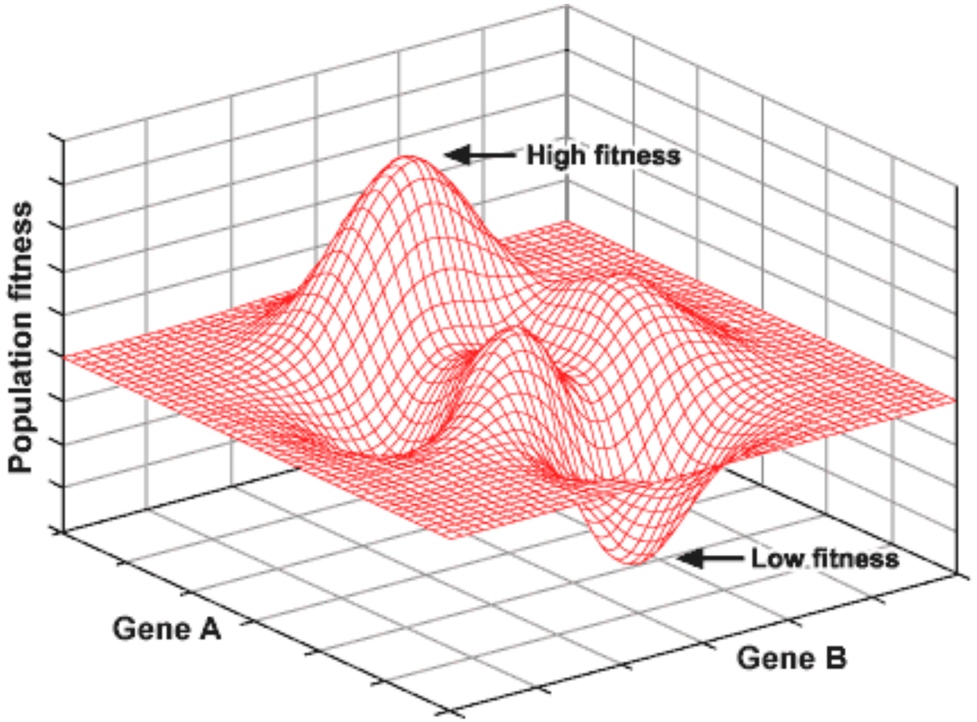
- Fitness Landscape = [solution space] × [fitness dimension]
- Optimisation is essentially climbing up hills to get higher fitness
- What if we see LLM-based solution generation as an optimisation process?
  - What would be the landscape that results in self-consistency?



LLM Reasoning as Constructive Optimisation

Why does self-consistency work?

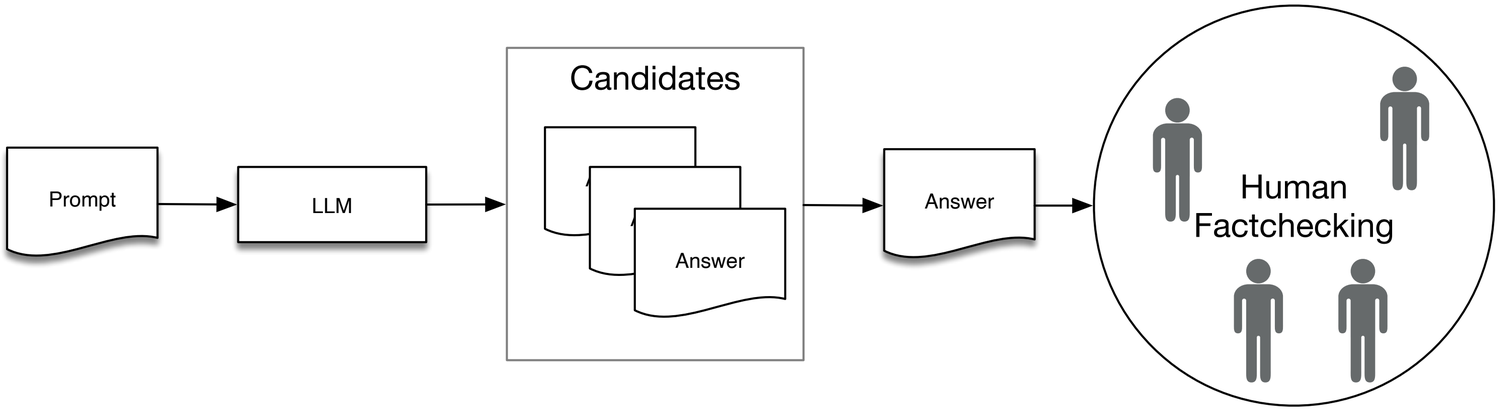
- With problems for which the self-consistency works, we may hypothesise that:
  - The tallest hill is also the largest; there are multiple starting points and pathways to the top
  - Smaller hills (=incorrect solutions) have smaller base area, resulting in fewer pathways to their top



Code is a unique artifact because it executes.

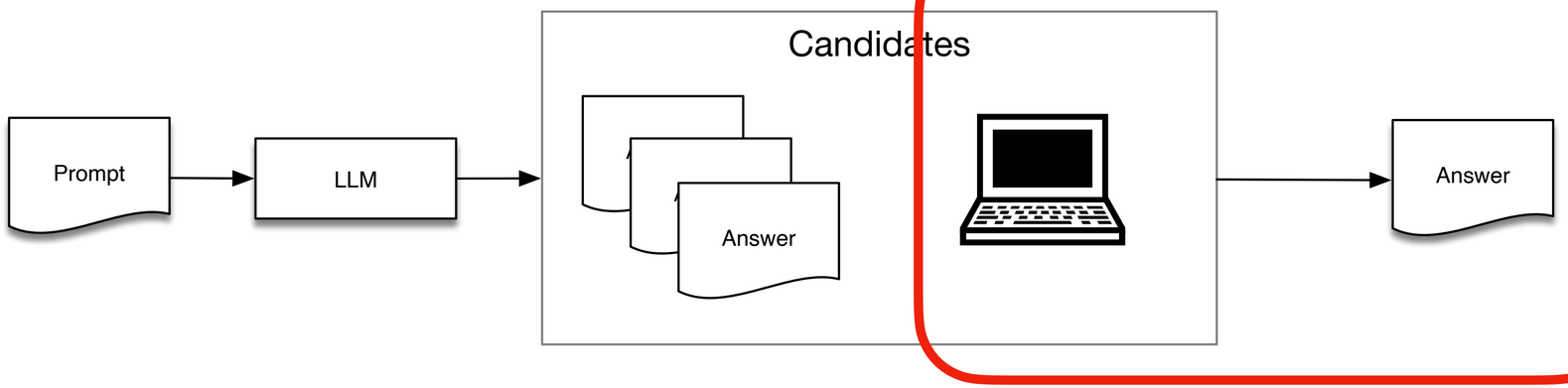
(And we've been doing dynamic analysis for a long time)

NL + LLM Pipeline



Isn't this testing? :)

PL/NL + LLM Pipeline



Cross-cutting Concerns

- **Architecture**: is asking a single LLM instance for answers sufficient? There are views that agents, and even multi-agents, are the future.
- **Energy**: closed-source LLMs are huge and come with massive carbon footprints. What is the trade-off between performance and energy consumption?
- **Openness**: organisations will NOT send their internal data over the network to query commercial models. What is the right provenance?
- **Trajectory**: have we already seen the performance peak? Or will they keep improving over time?