

Large Language Models in SE

CS454 AI-Based Software Engineering

Shin Yoo

Large Language Models for SE

- 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:
 - They **seem to** get the semantics of the code
 - **Emergent behavior** leading to very attractive properties such as in-context learning, Chain-of-Thoughts, or PAL
 - **Low technical barrier** compared to tailored analysis and techniques

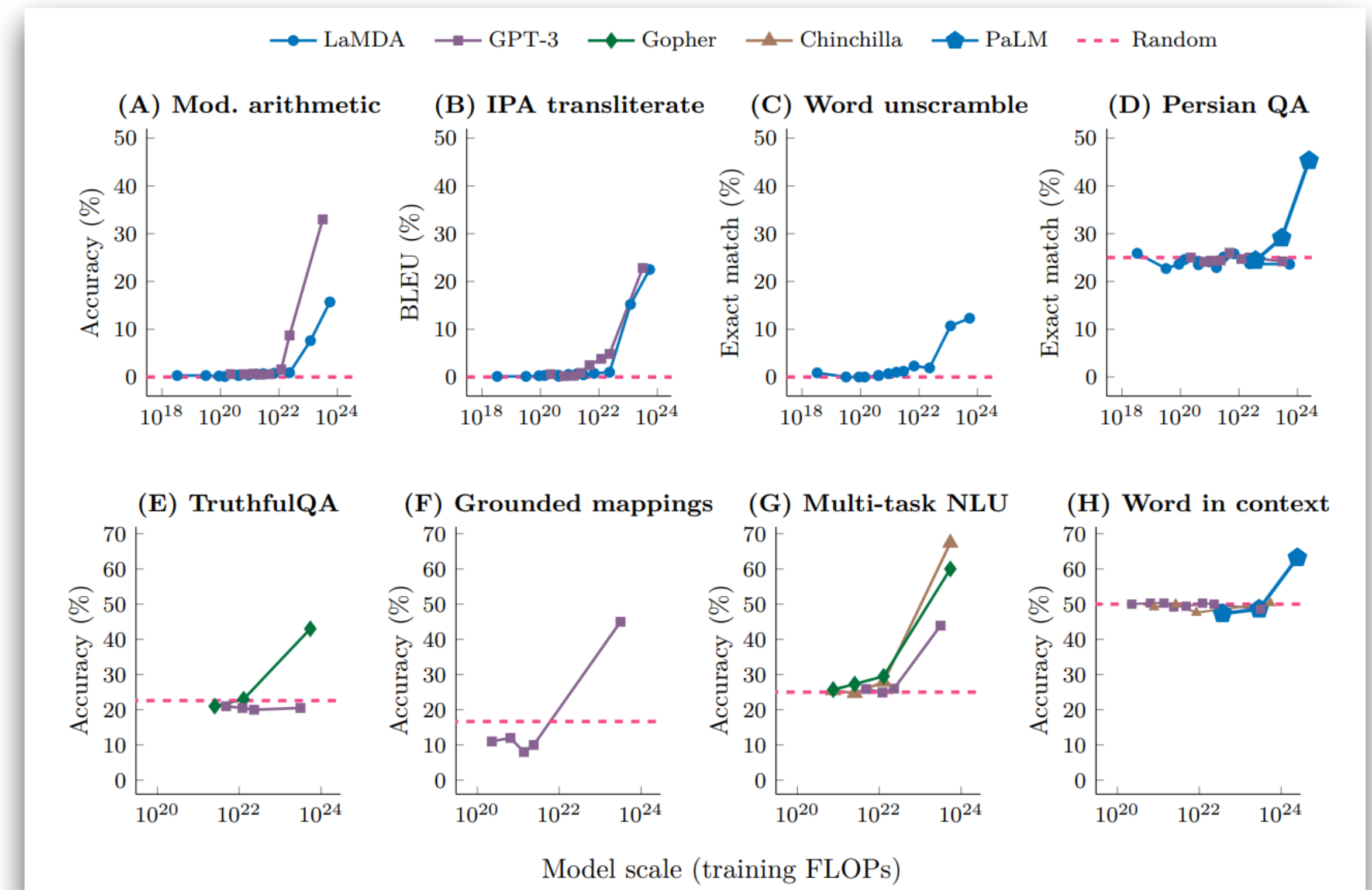
Further Guides

- 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.

Few-shot Bug Reproduction

Kang et al., ICSE 2023

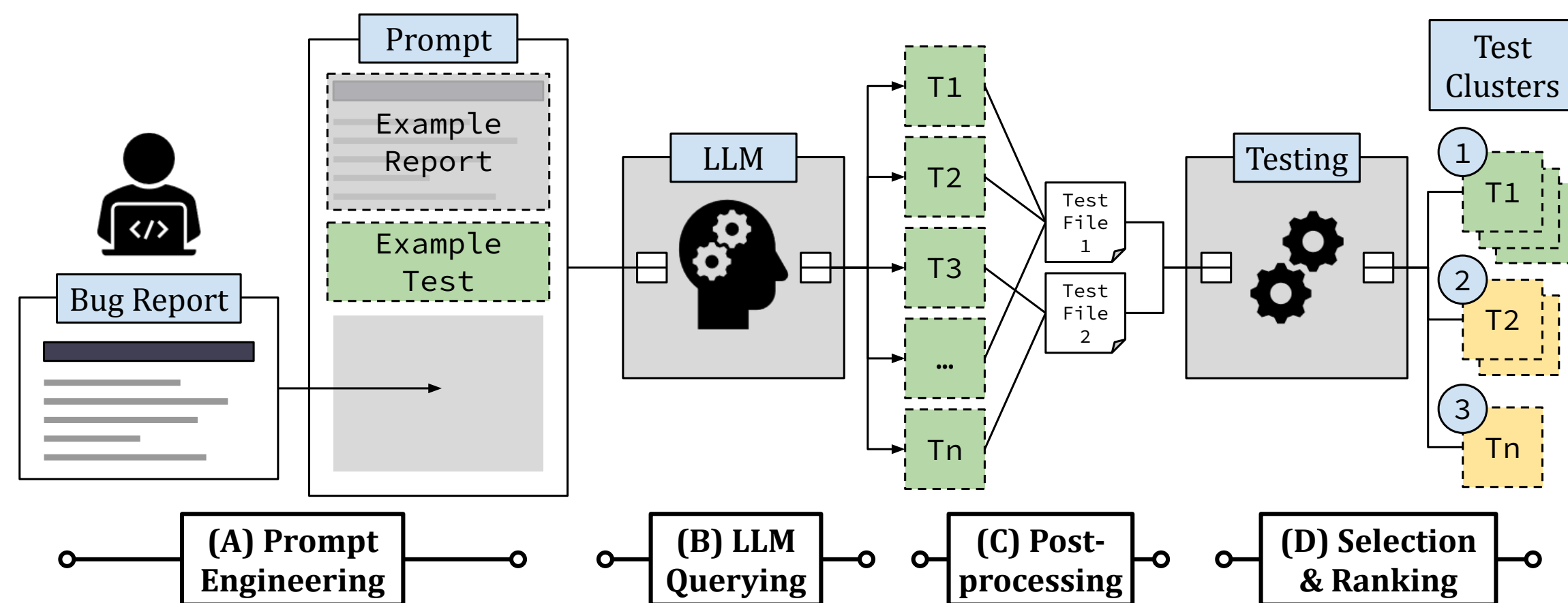
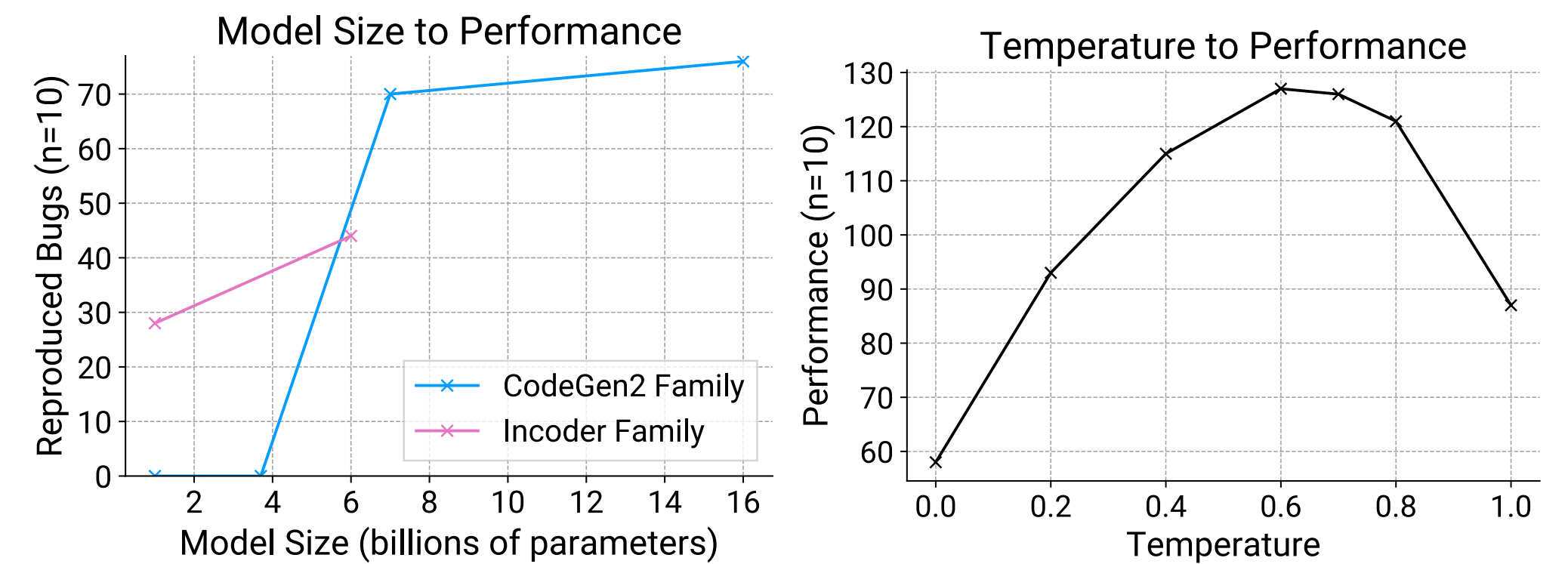


Fig. 1: Overview of LIBRO

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



(a) Model Size

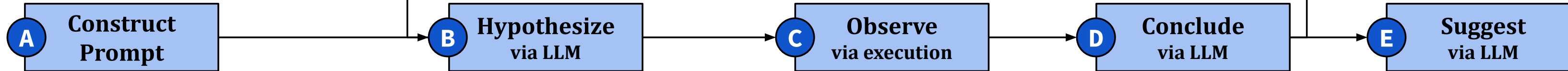
(b) Temperature

Fig. 8: Evaluation of the influence of LLM configuration to performance.

Zero-shot Automated Debugging

Kang et al., <https://arxiv.org/abs/2304.02195>

Pipeline (A-E)



Annotated Run (1-10)

Scientific Debugging Explanation

Debugging Problem Description

```

1 def f(n):
2   # Evaluate if n can be
3   # written as the sum of 4
4   # positive even numbers.
5   return n%2==0 and n>8

fails on the test
assert f(8) == True, f(8)

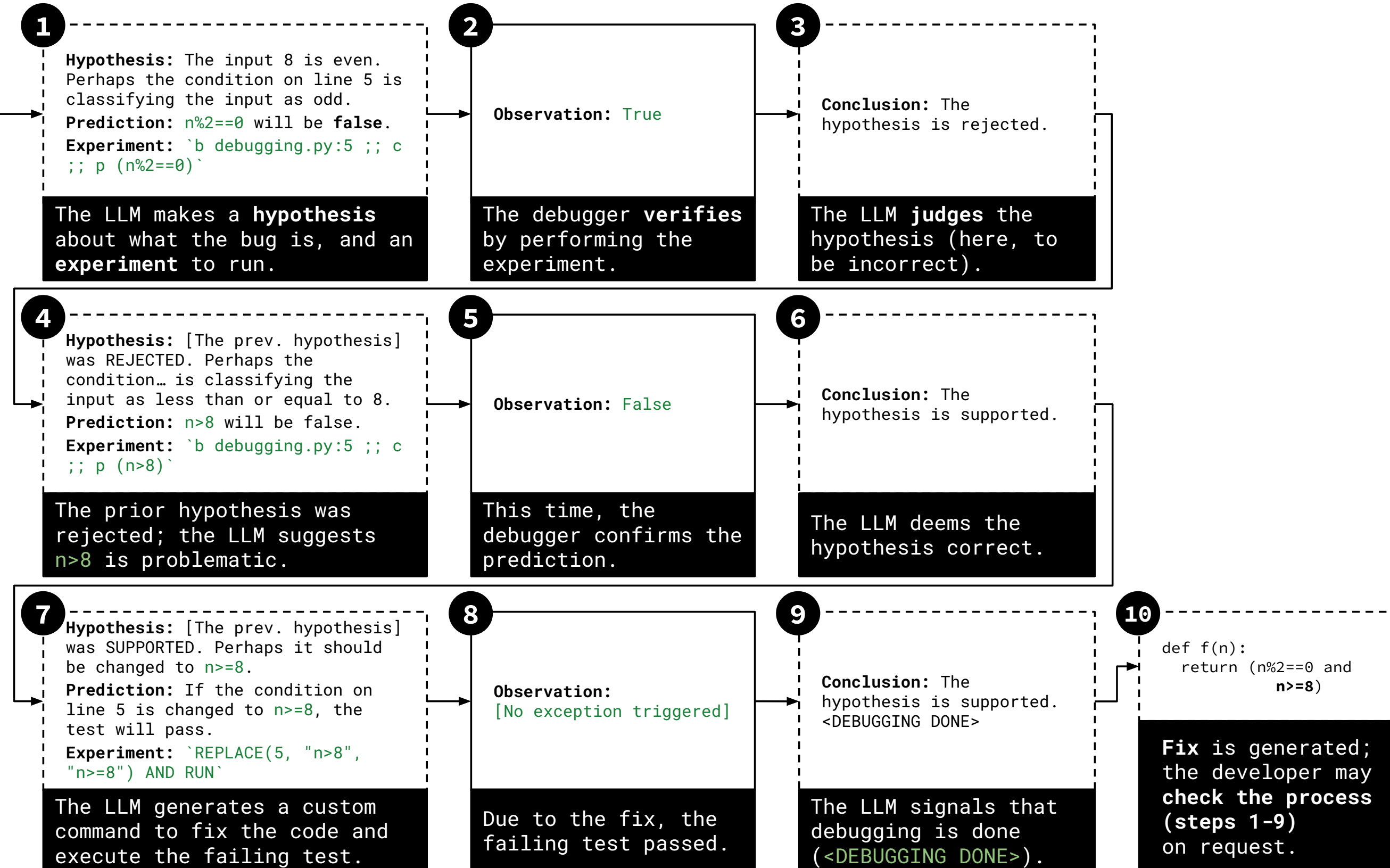
with the error message
...
AssertionError: False
  
```

Legend

→ Append to prompt after generation

--- Dashed boxes: generated by LLM

▭ Solid boxes: generated by debugger / test execution



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


```
(1d) ReAct (Reason + Act)

Thought 1: I need to search Apple Remote and find the
program it was originally designed to interact with.
Act 1: Search[Apple Remote]
Obs 1: The Apple Remote is a remote control introduced in
October 2005 by Apple ... originally designed to control the
Front Row media center program ...

Thought 2: Apple Remote was originally designed to control
the Front Row media center program. I need to search Front
Row next and find what other device can control it.
Act 2: Search[Front Row]
Obs 2: Could not find [Front Row]. Similar: ['Front Row
Seat to Earth', 'Front Row Motorsports', 'Front Row
(software)', ...]

Thought 3: Front Row is not found. I need to search Front
Row (software) .
Act 3: Search[Front Row (software)]
Obs 3: Front Row is a discontinued media center software ...

Thought 4: Front Row (software) is controlled by an Apple
Remote or the keyboard function keys. So the answer is
keyboard function keys.
Act 4: Finish[keyboard function keys]
```



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-shot examples that are mixtures of NL and LP can enhance the reasoning capabilities of LLM

Program-aided Language models (this work)

Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 tennis balls.

```
tennis_balls = 5
```

2 cans of 3 tennis balls each is

```
bought_balls = 2 * 3
```

tennis balls. The answer is

```
answer = tennis_balls + bought_balls
```

Q: The bakers at the Beverly Hills Bakery baked 200 loaves of bread on Monday morning. They sold 93 loaves in the morning and 39 loaves in the afternoon. A grocery store returned 6 unsold loaves. How many loaves of bread did they have left?

Model Output

A: The bakers started with 200 loaves

```
loaves_baked = 200
```

They sold 93 in the morning and 39 in the afternoon

```
loaves_sold_morning = 93
```

```
loaves_sold_afternoon = 39
```

The grocery store returned 6 loaves.

```
loaves_returned = 6
```

The answer is

```
answer = loaves_baked - loaves_sold_morning  
- loaves_sold_afternoon + loaves_returned
```

```
>>> print(answer)
```

```
74
```



Hallucination

- LLM = (Statistical) Autocompletion = completion not necessarily because it is the right choice, but because it is the likely choice.
- How do we filter out hallucinations?
 - Automated testing should help a bit, but eventually we will hit the oracle problem.



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?

Low Technical Barrier

- No language specific pre-analysis: you just paste the target code and call the API...?
- Low entry cost, yes, but:
 - Real innovation and practical impact only possible when you really understand the problem domain
 - Post-processing to filter out hallucination heavily involves existing automated testing techniques.

Remainder of today:

- AutoFL: how to use ReAct like function-call ability to perform fault localization
- DroidAgent: how to harness the reasoning capabilities of LLMs so that they drive an autonomous agent