

CSIT5100 – Engineering Reliable Software Systems

Is LLM a Dependable Coding Copilot?



Shing-Chi Cheung

Computer Science and Engineering

HKUST

Can somebody detect, locate, and repair bugs for me?



Dream of a developer...



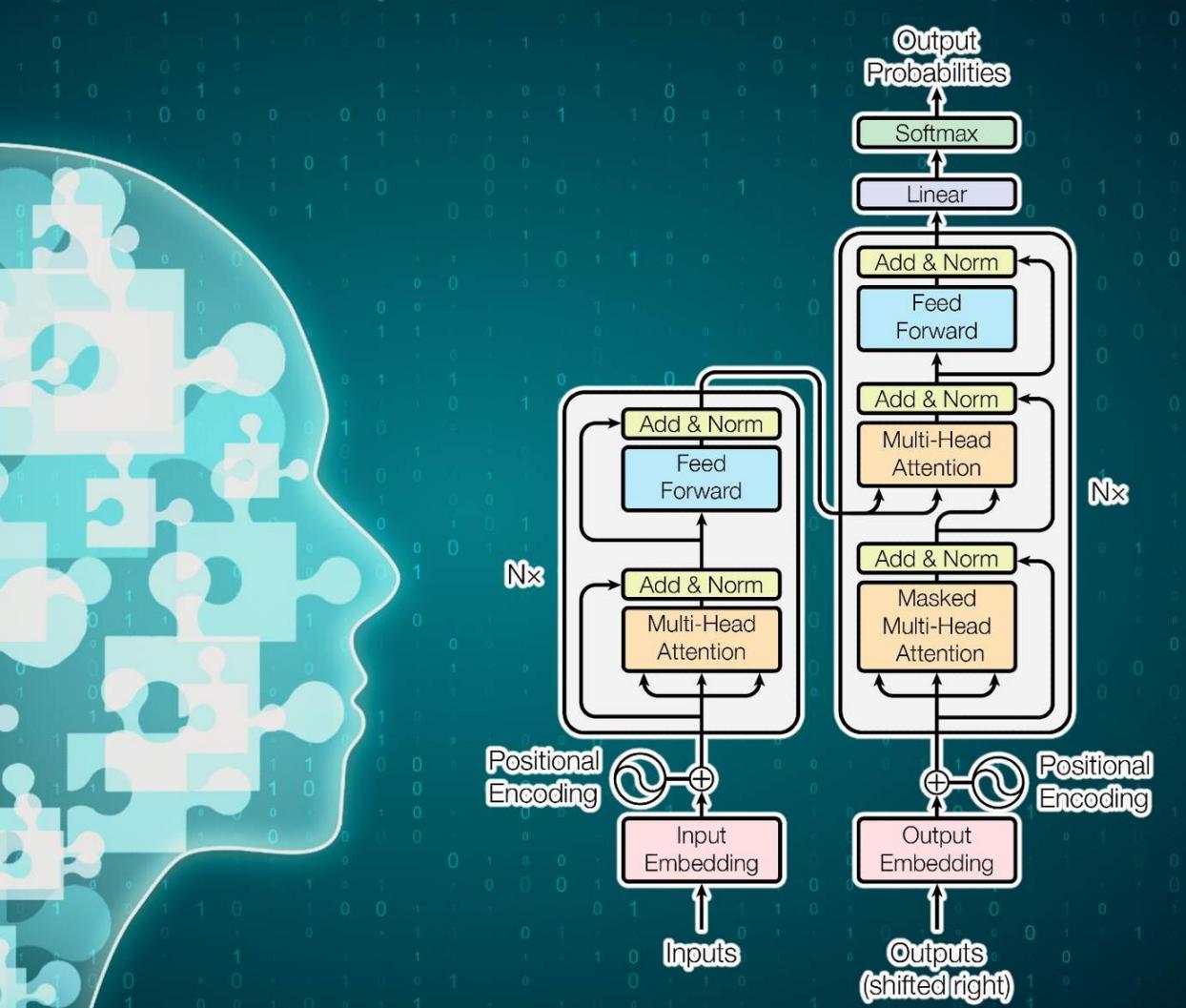
**Can somebody detect, locate, and repair
bugs for me?**

**Can somebody write, optimize, review, and
document code for me?**



Dream of a developer...

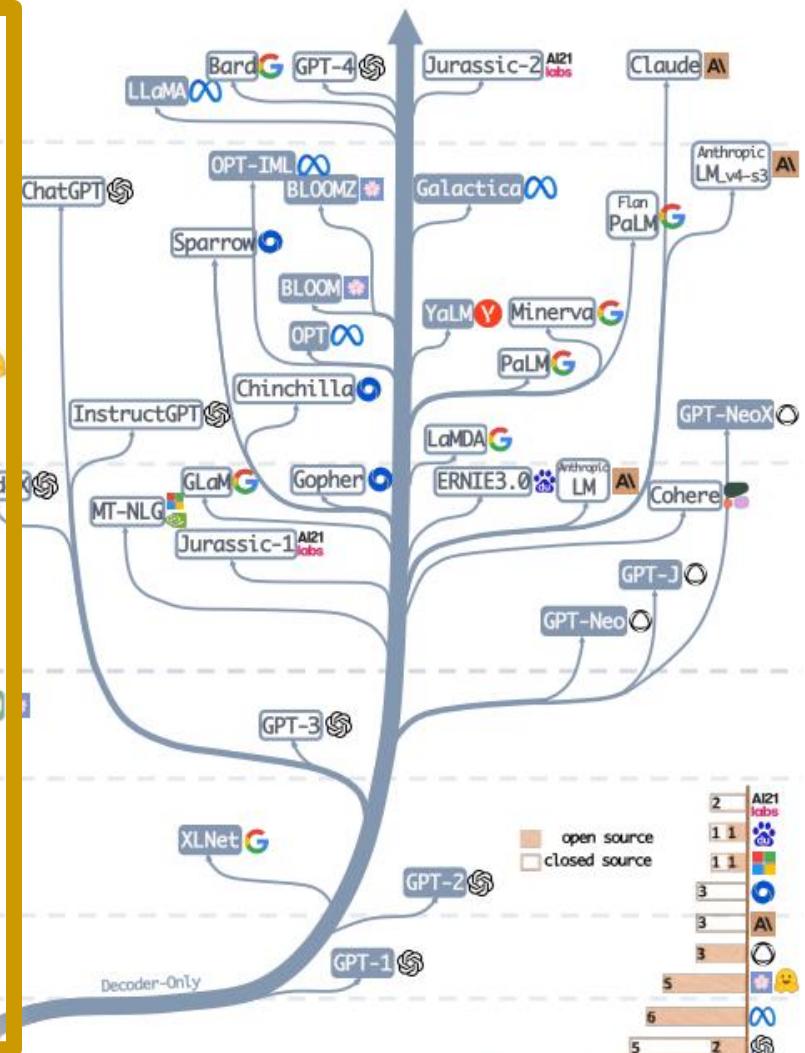
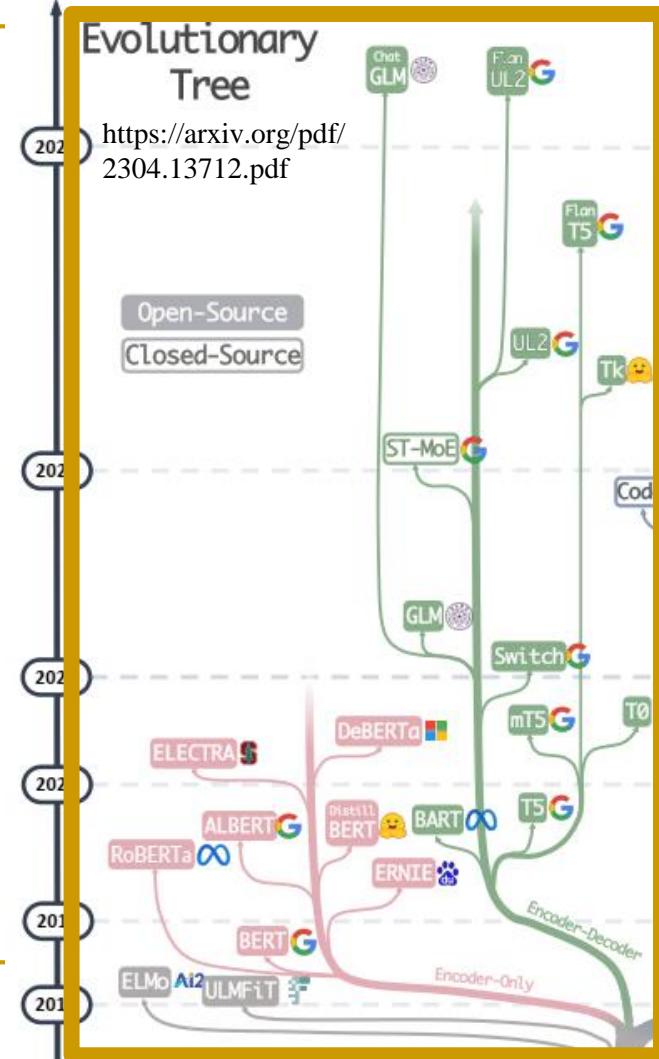




A bloom of powerful LLMs:

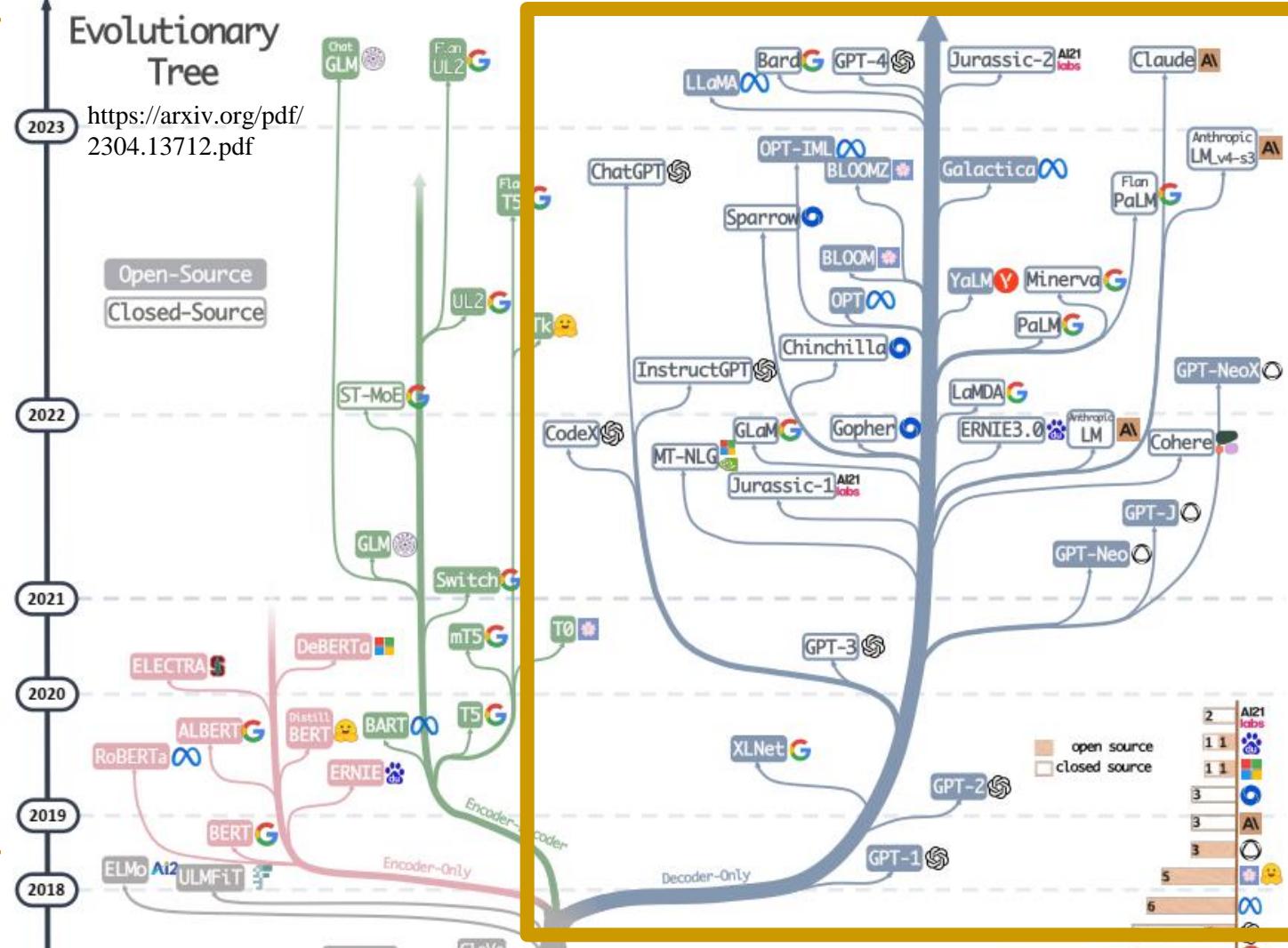
CodeT5, InCoder, Codex,
CodeGen, GPT-J, GPT-3,
ChatGPT, GPT-4, Bard,
Claude2, PaLM2, Vicuna,
GPT4All, Falcon, Aider,
StarCoder, WizardCoder,
Codewhisperer, Github
Copilot, Llama2, Freewilly2,
Orca, PanGu, PanGu-Coder,
GLM, Grok, CodeLamma,
TestGPT, Ernie, DeepSeek
Coder, Phind, ...

- ## Encoder-Decoder
- ### Encoder-only:
- BERT
(Bidirectional Encoder Representation Transformer)
 - Discriminative, predict masked tokens
 - Better for understanding



Decoder-only:

- GPT (Generative Pre-trained Transformer)
- Generative, predict next tokens
- Better for token generation



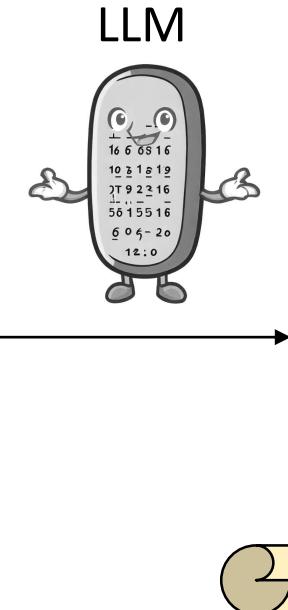
GPT-4-turbo 128K announced by OpenAI in Nov 2023

Model name	Discontinuation date	Replacement model
gpt-3.5-turbo-0613	Jun 13, 2024	gpt-3.5-turbo-1106
gpt-3.5-turbo-0301	Jun 13, 2024	gpt-3.5-turbo-0613
gpt-4-0314	Jun 13, 2024	gpt-4-0613
gpt-4-32k-0314	Jun 13, 2024	gpt-4-32k-0613



How will the
practice of
software
development
change in the era
of LLMs?

Implement a Python function that takes a list of integers `nums` as input and returns an integer that represents **the largest possible sum of any contiguous sublist** of `nums`.

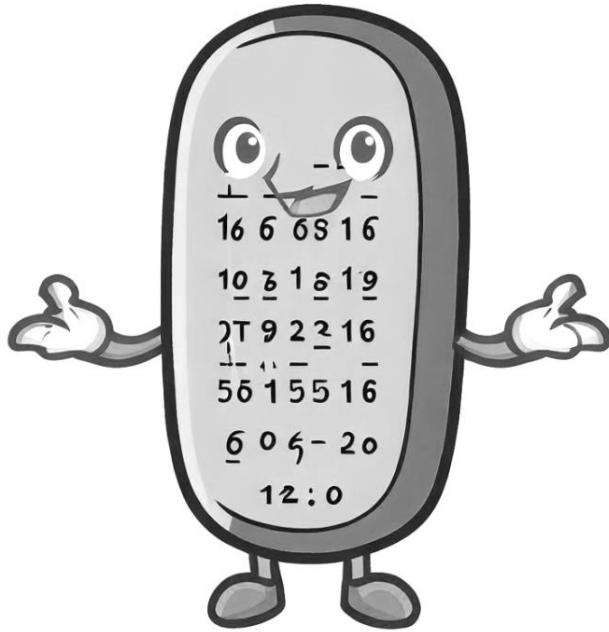


```
def max_sublist_sum(nums):
    if not nums:
        return 0
    max_sum = nums[0]
    for i in range(1, len(nums)):
        if nums[i] > nums[i-1]:
            max_sum += nums[i]
        else:
            max_sum = nums[i]
    return max_sum
```

[4, -5, 2, 1, -1, 3]



Software Engineer



A fundamental limitation of LLM is that it is weak in logical reasoning. Answers that require logical reasoning are not necessarily correct

... returns an integer that represents **the largest possible sum of any contiguous sublist** of nums.

Input: [4, -5, 2, 1, -1, 3]

Output: 2

Expected output: 5



```
def max_sublist_sum(nums):
    if not nums:
        return 0
    max_sum = nums[0]
    for i in range(1, len(nums)):
        if nums[i] > nums[i-1]:
            max_sum += nums[i]
        else:
            max_sum = nums[i]
    return max_sum
```

Conventional Software Development Practice

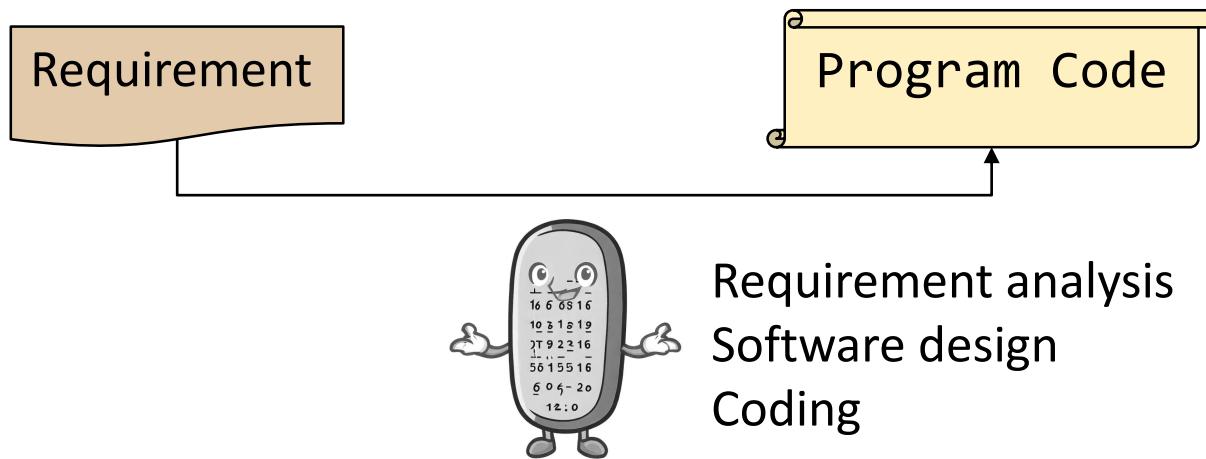


Requirement analysis
Software design
Coding



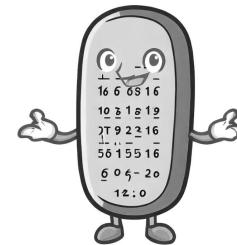
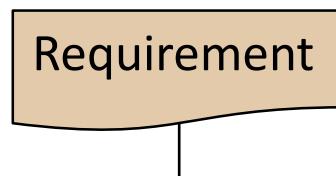
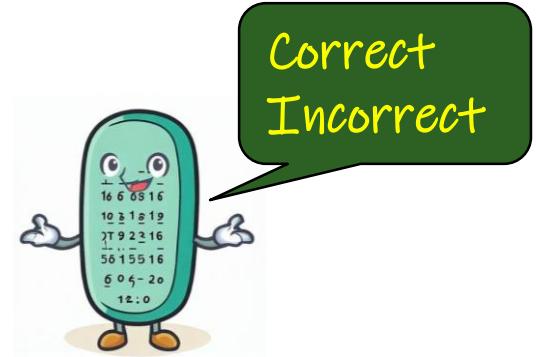
How does that affect the roles for software engineers?

LLM-based
Software
Development
Practice



LLM-based Software Development Practice

Can we go further to let
LLMs provide recommendation
on program correctness?



Requirement analysis
Software design
Coding

GPT-4 Doesn't Know It's Wrong: An Analysis of Iterative Prompting for Reasoning Problems

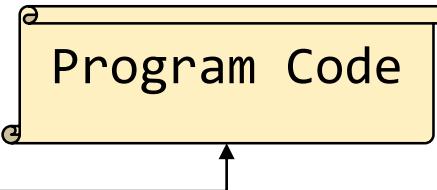
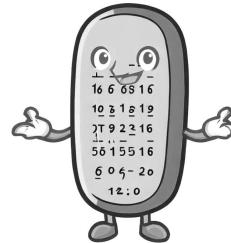
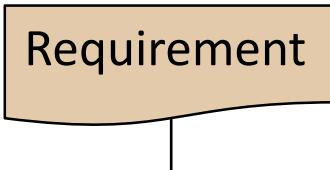
Kaya Stechly*

Matthew Marquez*

Subbarao Kambhampati*

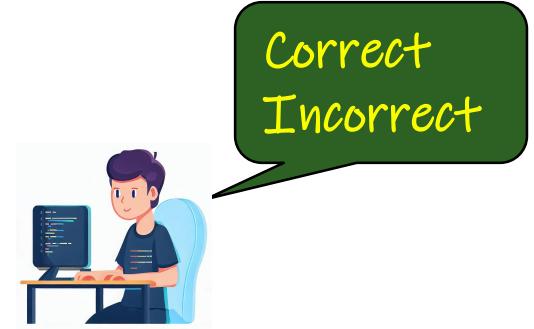
arXiv:2310.12397v1 [cs.AI] 19 Oct 2023

LLM-based
Software
Development
Practice

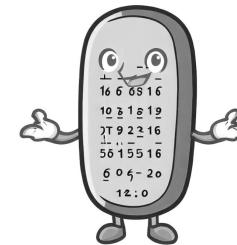
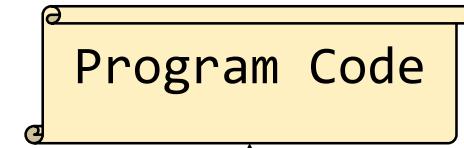
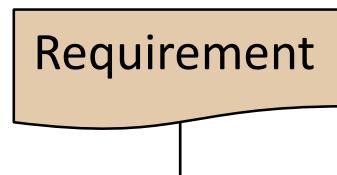


Cannot reliably reason about if a program is correct

In the foreseeable future, human will maintain a major role in LLM-based software development



LLM-based
Software
Development
Practice



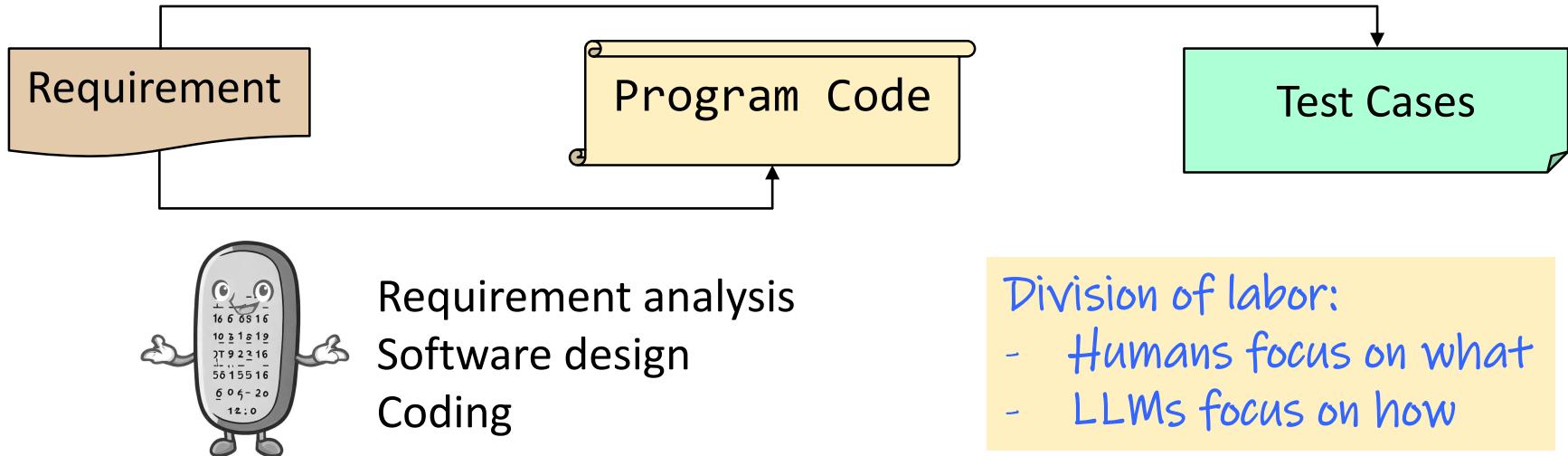
Requirement analysis
Software design
Coding

LLM-based Software Development Practice

But human's role will shift ...



Requirement analysis
Test case design



Test 1

Input: []

Expected Output: 0

Test 2

Input: [-1, 2]

Expected Output: 2

Test 3

Input: [4, -5, 2, 1, -1, 3]

Expected Output: 5

Test 4

Input: [-2, -4, -6, -5]

Expected Output: 0

...

Test Case Design

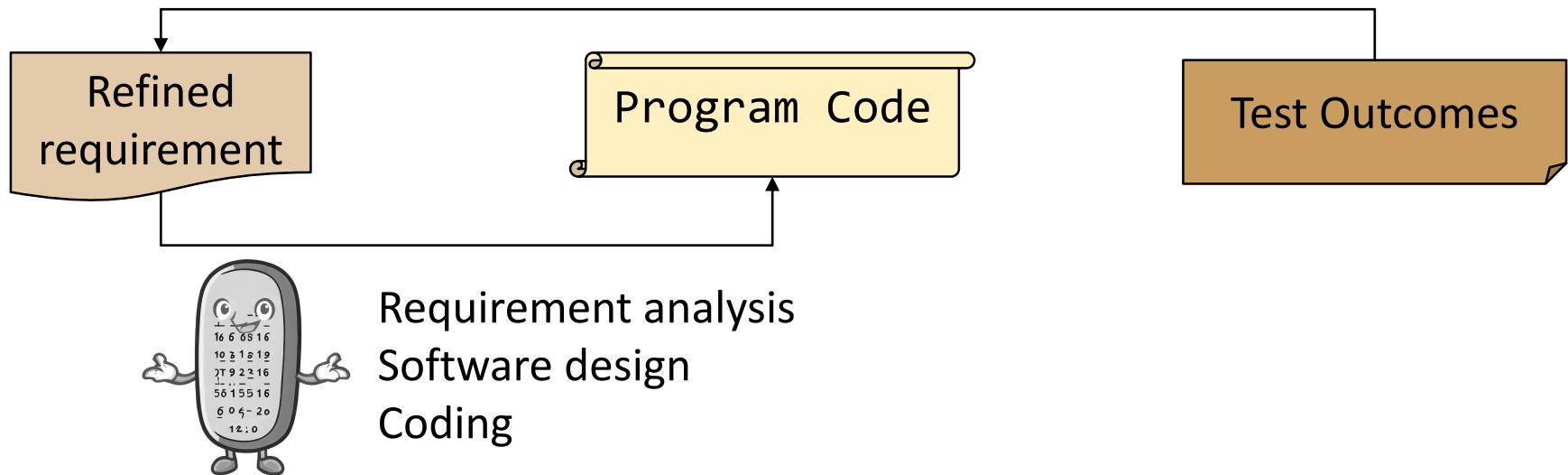


LLM-based Software Development Practice

But human's role will shift ...



Analyze test results
Refine requirements



Test 1

Input: []

Expected Output: 0

Test 2

Input: [-1, 2]

Expected Output: 2

Test 3

Input: [4, -5, 2, 1, -1, 3]



Expected Output: 5

Test 4



Input: [-2, -4, -6, -5]

Expected Output: 0

...



Refined requirement (prompt)

Test 1

Input: []

Expected Output: 0

Test 2

Input: [-1, 2]

Expected Output: 2

Test 3

Input: [4, -5, 2, 1, -1, 3]



Expected Output: 5

Test 4



Input: [-2, -4, -6, -5]

Expected Output: 0

...



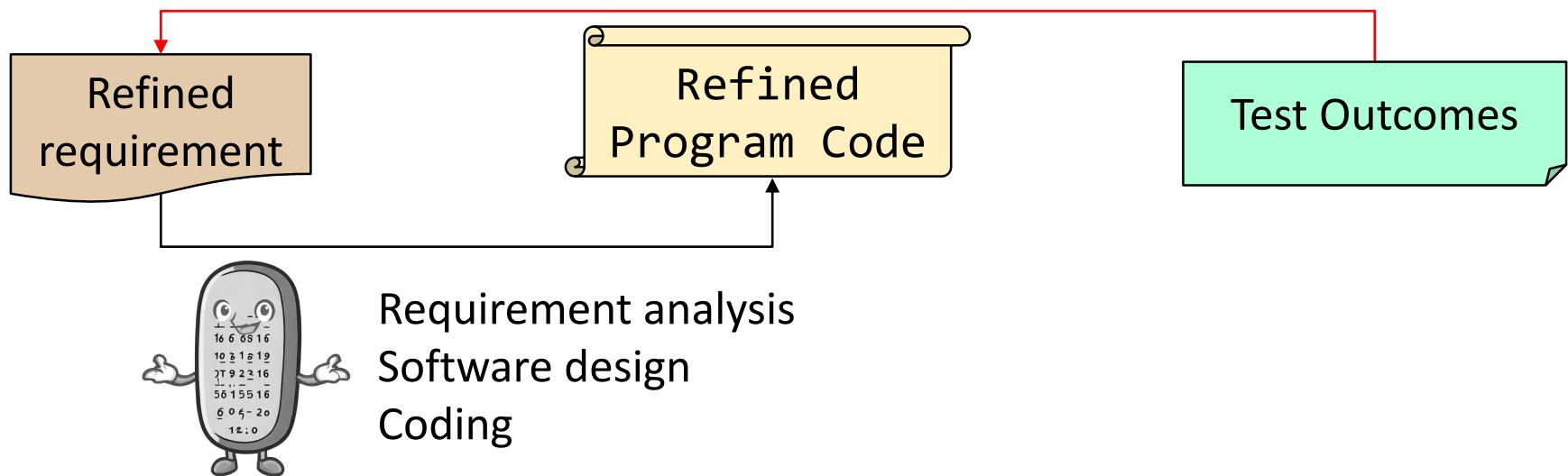
Implement a Python function that takes a list of integers `nums` as input and returns an integer that represents the largest possible sum of any contiguous sublist of `nums`. For example, given an input [4, -5, 2, 1, -1, 3], its maximum sum is given by the sublist [2, 1, -1, 3] with the largest sum being 5. If all the numbers in `nums` are negative, then its maximum sum is defined to be 0. In case that the maximum subsequence is not unique, output the one with the smallest indices.

LLM-based Software Development Practice

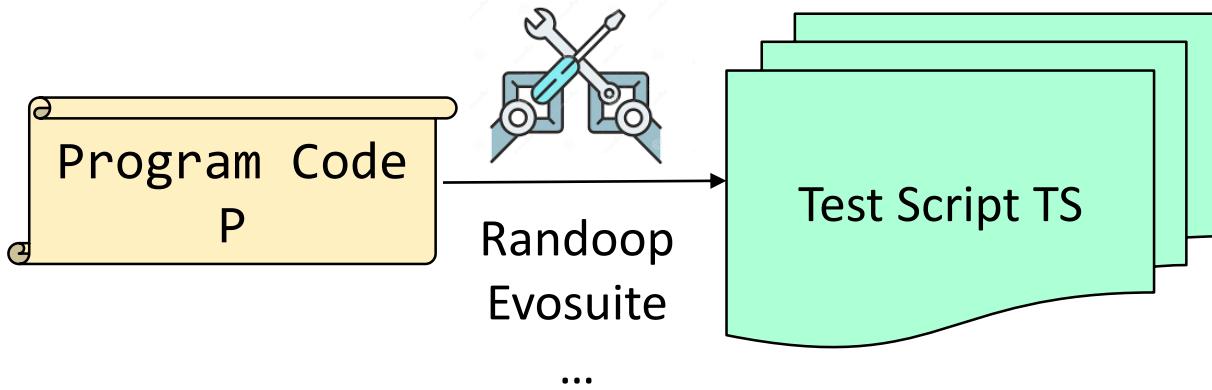
But human's role will shift ...



Analyze test results
Refine requirements



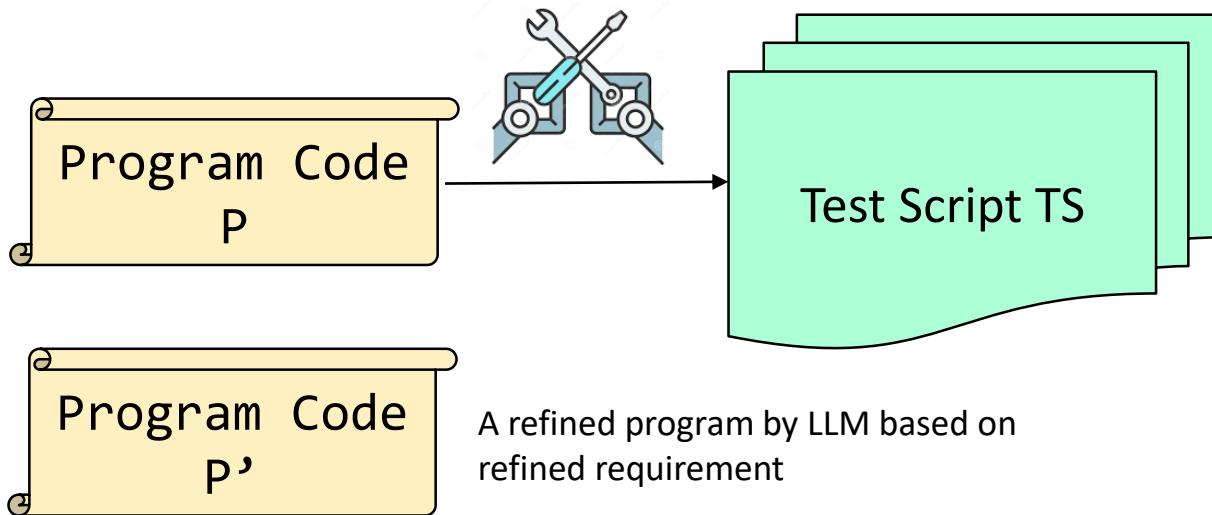
Test generation strategy also needs to be reviewed



Allocate non-trivial computation resources to generate test suite fulfilling specific criteria:

- Branch coverage
- Threshold coverage
- Test prioritization
- Test selection
- ...

Programs P and P' can have different structures. TS may no longer satisfy the same coverage criteria for program P'



Allocate non-trivial computation resources to generate test suite fulfilling specific criteria:

- Branch coverage
- Threshold coverage
- Test prioritization
- Test selection
- ...

Program P

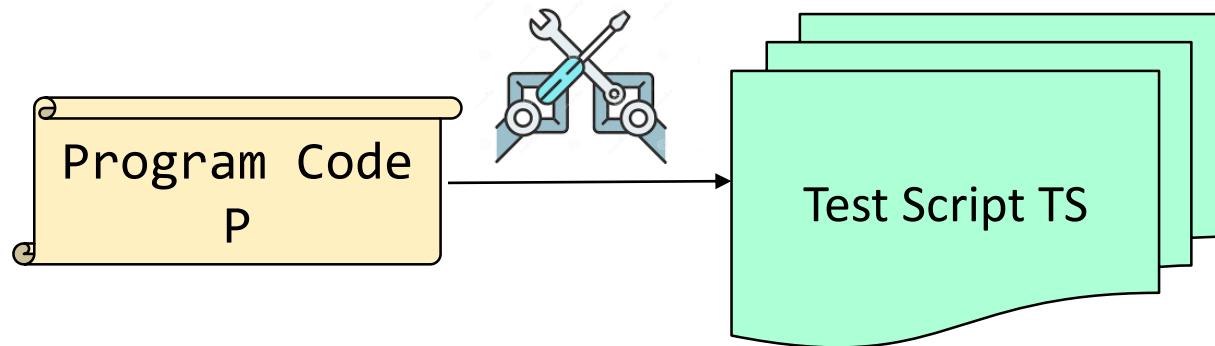
```
def max_sublist_sum(nums):
    if not nums:
        return 0
    max_sum = nums[0]
    for i in range(1, len(nums)):
        if nums[i] > nums[i-1]:
            max_sum += nums[i]
        else:
            max_sum = nums[i]
    return max_sum
```

Program P'

```
def max_sublist_sum(nums):
    cur_max = max_sum = 0
    for num in nums:
        cur_max = max(cur_max + num, num)
        max_sum = max(max_sum, cur_max)
    return max_sum
```

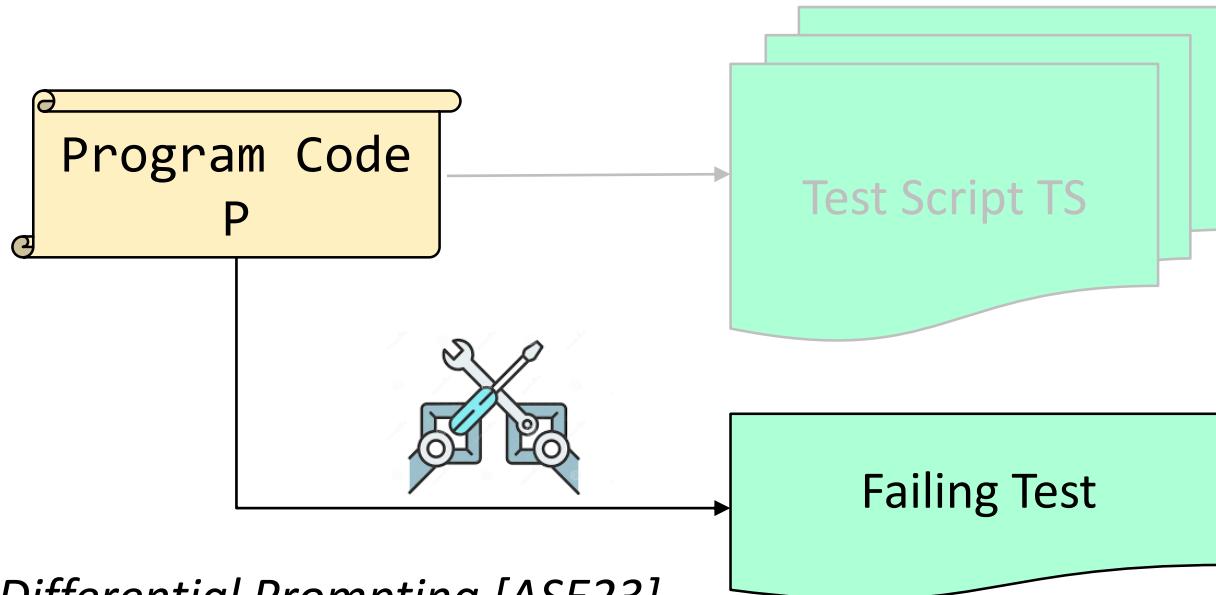
LLM can generate an alternate program with a significantly different structure for a revised requirement

May favor a failure-driven test generation strategy



Differential Prompting [ASE23]

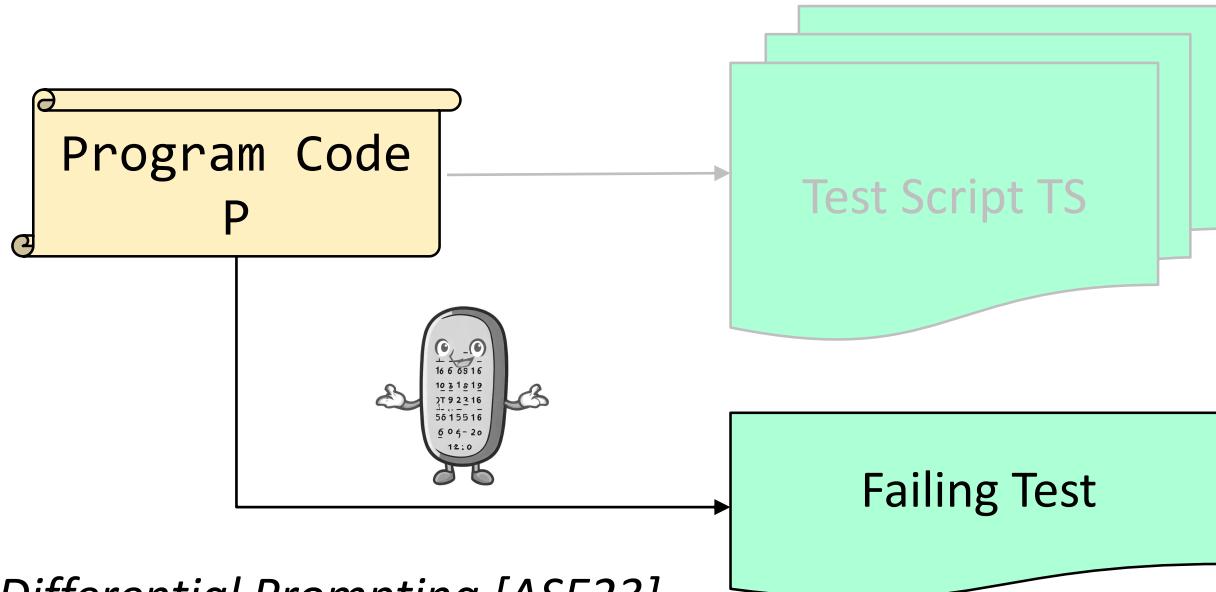
May favor a failure-driven test generation strategy



Differential Prompting [ASE23]

- Generate a failing test if the program has a bug
- Provide feedback based on the test result to LLM through a follow-up prompt
- Repeat until no failing test is found

Can we use LLM?

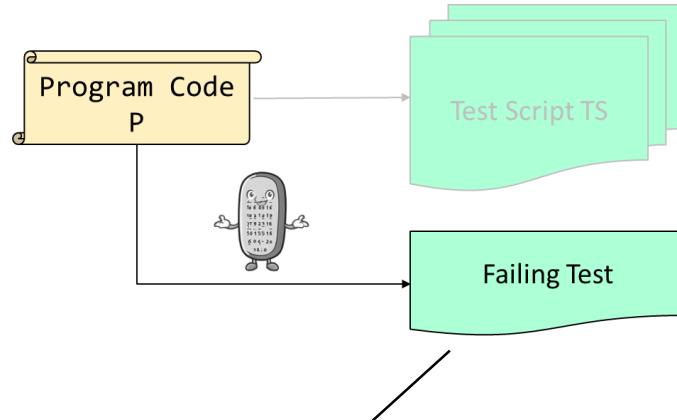


Differential Prompting [ASE23]

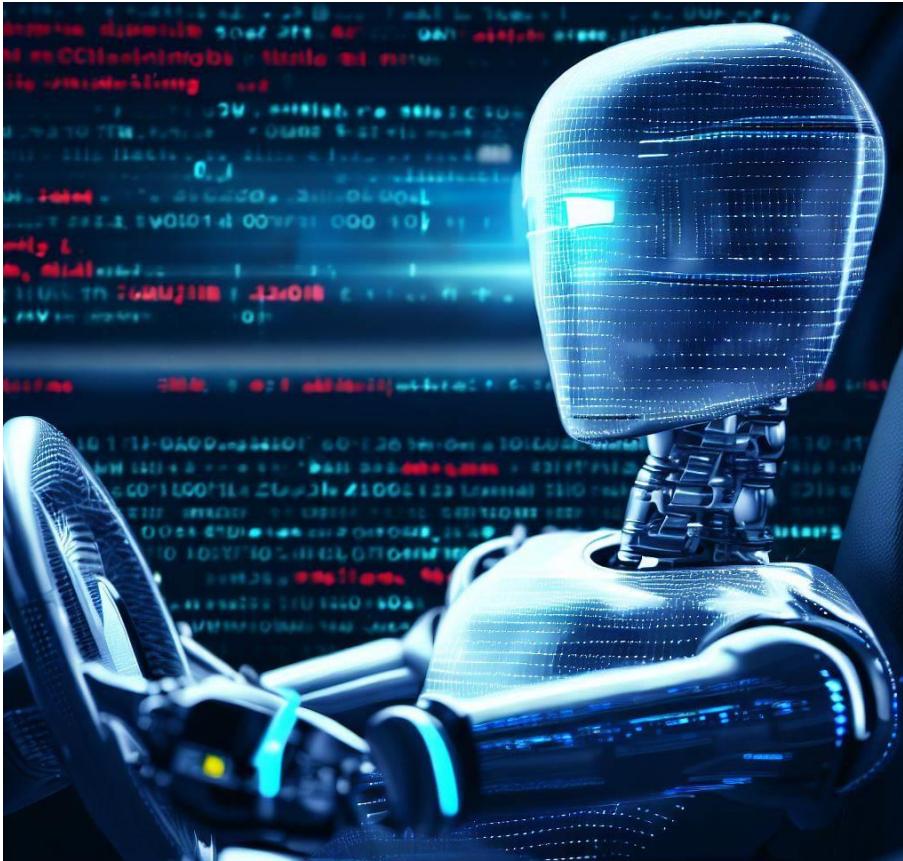
- Generate a failing test if the program has a bug
- Provide feedback based on the test result to LLM through a follow-up prompt
- Repeat until no failing test is found

Outstanding Challenges

Existing LLMs



- Poor in generating failure-revealing test inputs
- Cannot determine the correct (i.e., expected) output of a given test input
- Their logical explanation can be illogical



Can we make LLM a
dependable copilot to
generate failing tests?

This is a **buggy** program in the QuixBugs dataset, which is a popular benchmark for program repair [Lin *et al.* SPLASH17]

Please study the following program and examine if it is correct.

```
def mp1(arr):
    max_ending_here = 0
    max_so_far = 0
    for x in arr:
        max_ending_here = max(0, max_ending_here + x)
        max_so_far = max(max_so_far, max_ending_here)
    return max_so_far
```

The program finds the maximum sum of a contiguous subarray within a one-dimensional array of numbers which has at least one positive element. The program is correct and will work as expected.



↑
LLM Chat can infer the intention of this program even it is buggy

What is the output of $\text{mp1}([4, -5, 2, 1, -1, 3])$?

The output of $\text{mp1}([4, -5, 2, 1, -1, 3])$ is 6.



Facts:

The actual output of $\text{mp1}([4, -5, 2, 1, -1, 3])$ is 4

The expected output is 5, which is given by the sublist $[2, 1, -1, 3]$

It is a fundamental weakness with LLMs. They are weak in answering facts. We commonly observe this test oracle problem across LLMs, including ChatGPT, GPT-4 and Vicuna.

The output returned by mp1 is actually 4. Is mp1 correct?

- ✓ Searching for: what is the correct output of mp1([4, -5, 2, 1, -1, 3])
- ✓ Generating answers for you...



I apologize for the mistake. The correct output of mp1([4, -5, 2, 1, -1, 3]) is actually 4. The program is correct and will work as expected ¹.

Facts:

The actual output of mp1([4, -5, 2, 1, -1, 3]) is 4

The expected output is 5, which is given by the sublist [2, 1, -1, 3]

It is a fundamental weakness with LLMs. They are weak in answering facts. We commonly observe this test oracle problem across LLMs, including ChatGPT, GPT-4 and Vicuna.

What is the output of $\text{mp1}([4, -5, 2, 1, -1, 3])$?

The output of $\text{mp1}([4, -5, 2, 1, -1, 3])$ is 6.



Facts:

The actual output of $\text{mp1}([4, -5, 2, 1, -1, 3])$ is 4

The expected output is 5, which is given by the sublist $[2, 1, -1, 3]$

The test cases generated by LLMs can result in many false positives, even they generate many test cases achieving high code coverage.

What is the output of $\text{mp1}([4, -5, 2, 1, -1, 3])$?

The output of $\text{mp1}([4, -5, 2, 1, -1, 3])$ is 6.



Facts:

The actual output of $\text{mp1}([4, -5, 2, 1, -1, 3])$ is 4

The expected output is 5, which is given by the sublist $[2, 1, -1, 3]$

Therefore, we study if LLMs can generate **failing test cases**, which addresses the primary objective of testing. We focus on the detection of functional (i.e., non-crashing) bugs.

Three Major Challenges

Goal: To prompt
LLM Chat so that
it correctly
generates failing
test cases for mp1()

- LLM Chat cannot tell whether a program is correct or not
- LLM Chat cannot tell the actual output for a given input
- LLM Chat cannot tell the expected output for a given input
- **BUT, LLM Chat can infer the program intention**

What do we want LLM Chat to do?

- Find a failing input so that
 - It results in an execution that reaches the faulty statement
 - The faulty statement is so executed that it leads the program to an incorrect state
 - The incorrect state should result in an incorrect output
- Deduce the expected output for the input

The program mp1 is incorrect. Suggest a failure-inducing test case that gives an incorrect output.
Consider it step by step.



A failure-inducing test case for mp1 is [1, -1, 2]. The expected output is 2 but the program will return 1. Here are the steps to reproduce the error:

- The maximum sum ending at the first position is 1.
- The maximum sum ending at the second position is 0.
- The maximum sum ending at the third position is 2.
- The maximum sum seen so far is 2.
- The program returns 1 instead of 2.

The first attempt – asking for a failing test case directly

[1, -1, 2] is not a failing input. The input results in an execution reaching the faulty statement, but it cannot lead to an incorrect output. The actual output is 2, which is the same as the expected output.

Please study the following program and examine if it is correct.

```
def mp1(arr):
    max_ending_here = 0
    max_so_far = 0

    for x in arr:
        max_ending_here = max_ending_here + x
        max_so_far = max(max_so_far, max_ending_here)

    return max_so_far
```

Recall

The program finds the maximum sum of a contiguous subarray within a one-dimensional array of numbers which has at least one positive element. The program is correct and will work as expected.



LLM Chat can infer the intention of this program even it is buggy

Write a program and call it mr1 so that it performs the following. The program finds the maximum sum of a contiguous subarray within a one-dimensional array of numbers which has at least one positive element.

Here is the program you requested:

```
def mr1(arr):
    max_ending_here = 0
    max_so_far = 0

    for x in arr:
        max_ending_here = max(0, max_ending_here + x)
        max_so_far = max(max_so_far, max_ending_here)

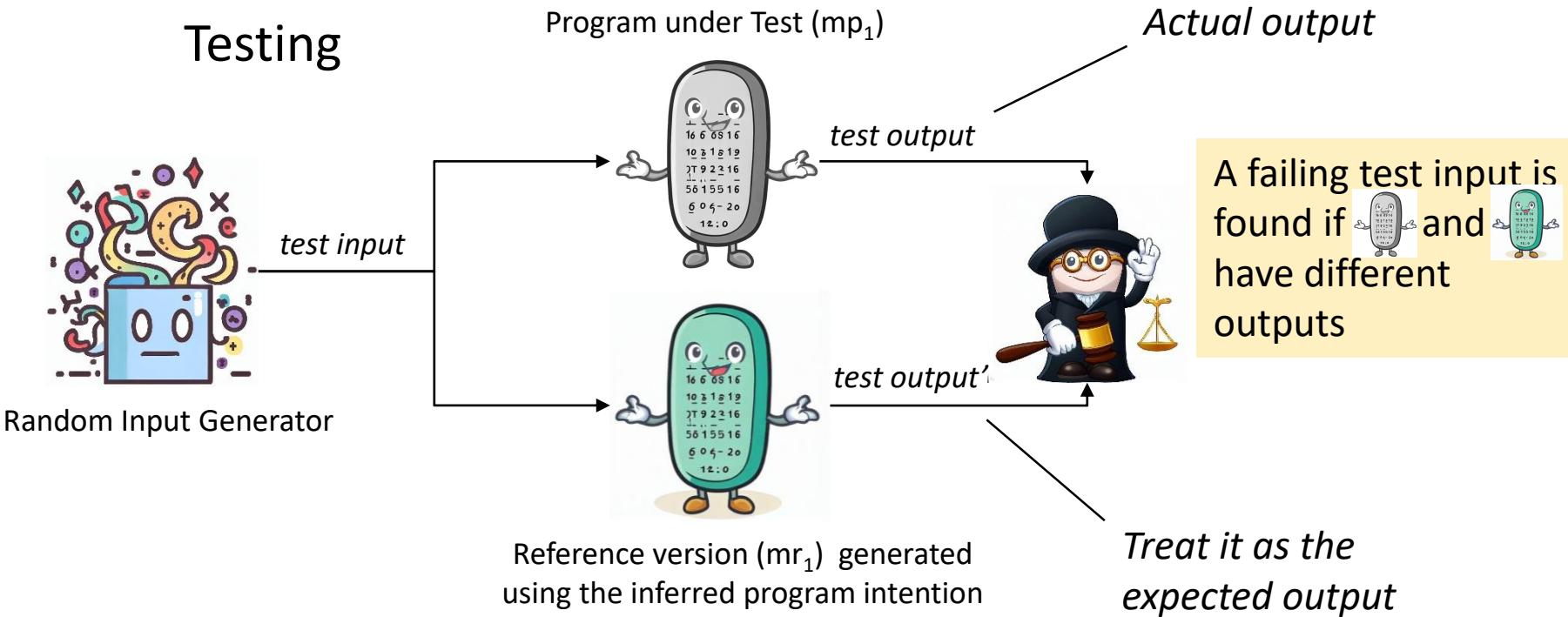
    return max_so_far
```

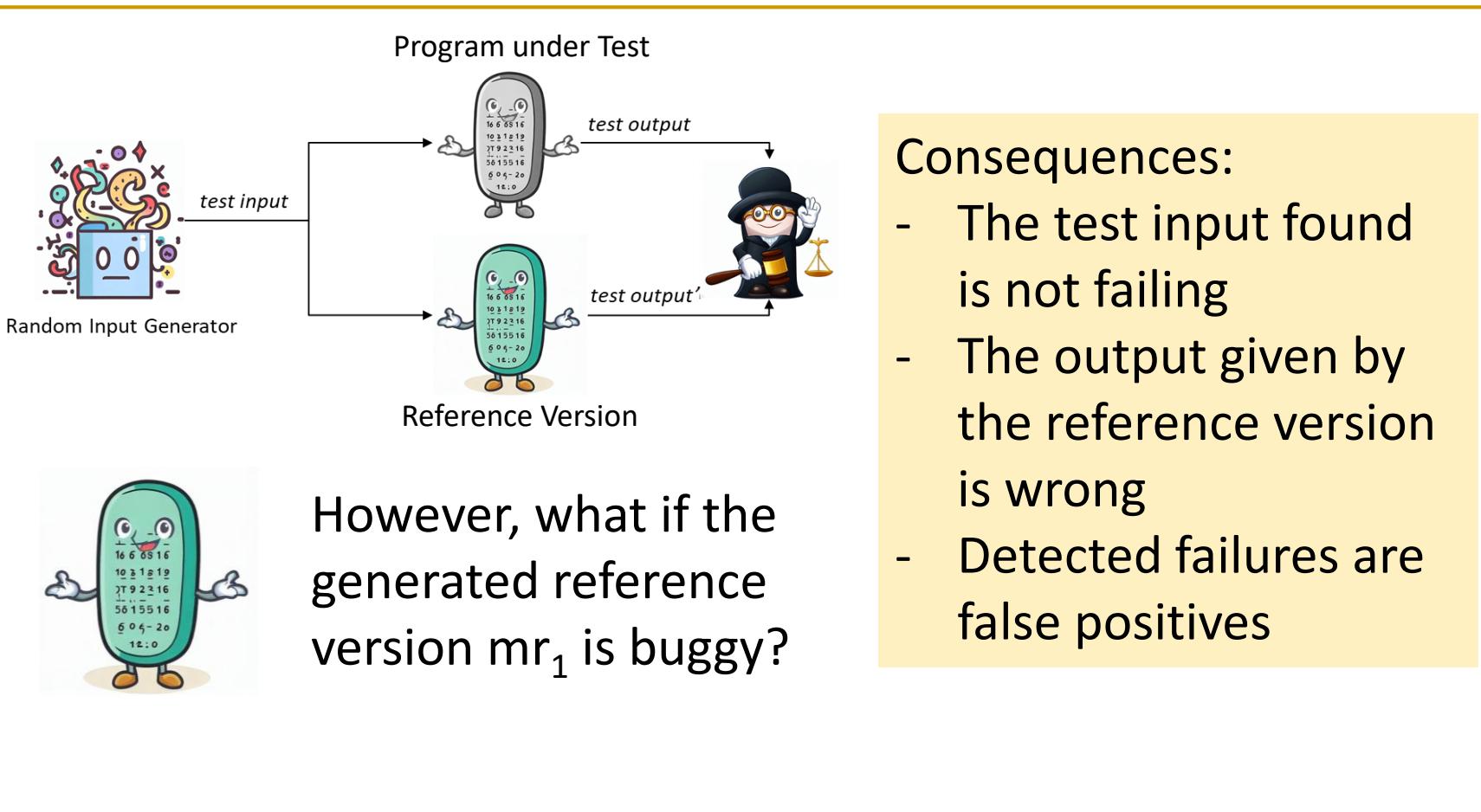
Second attempt –
Generate a program
using the inferred
program intention.



The generated program is correct!

Differential Testing







Context cleared

Write a program and call it mr1 so that it performs the following.

The program finds the maximum sum of a contiguous subarray within a one-dimensional array of numbers which has at least one positive element.

Here is a [Python program](#) that finds the maximum sum of a contiguous subarray within a one-dimensional array of numbers which has at least one positive element:

```
python
def mr1(arr):
    max_sum = 0
    cur_sum = 0
    has_positive = False
    for num in arr:
        if num > 0:
            has_positive = True
        cur_sum += num
        if cur_sum < 0:
            cur_sum = 0
        if cur_sum > max_sum:
            max_sum = cur_sum
    if has_positive:
        return max_sum
    else:
        return 0
```



Let's ask ChatGPT to generate a reference version

It generates a buggy reference version (mr_1)



Write a different program and call it mr2 so that it performs the following. The program finds the maximum sum of a contiguous subarray within a one-dimensional array of numbers which has at least one positive element.

Certainly! Here is another program "mr2" in Python that finds the maximum sum of a contiguous subarray within a one-dimensional array of numbers which has at least one positive element:

```
python
def mr2(arr):
    max_sum = float('-inf')
    curr_sum = 0
    has_positive = False

    for num in arr:
        curr_sum += num
        if curr_sum > max_sum and has_positive:
            max_sum = curr_sum
        if curr_sum < 0:
            curr_sum = 0
        if num > 0:
            has_positive = True

    if not has_positive:
        return 0
    return max_sum
```



Let's ask ChatGPT to generate another reference version

It generates another buggy reference version
mr₂

Output of Various Versions

mp1

mr1

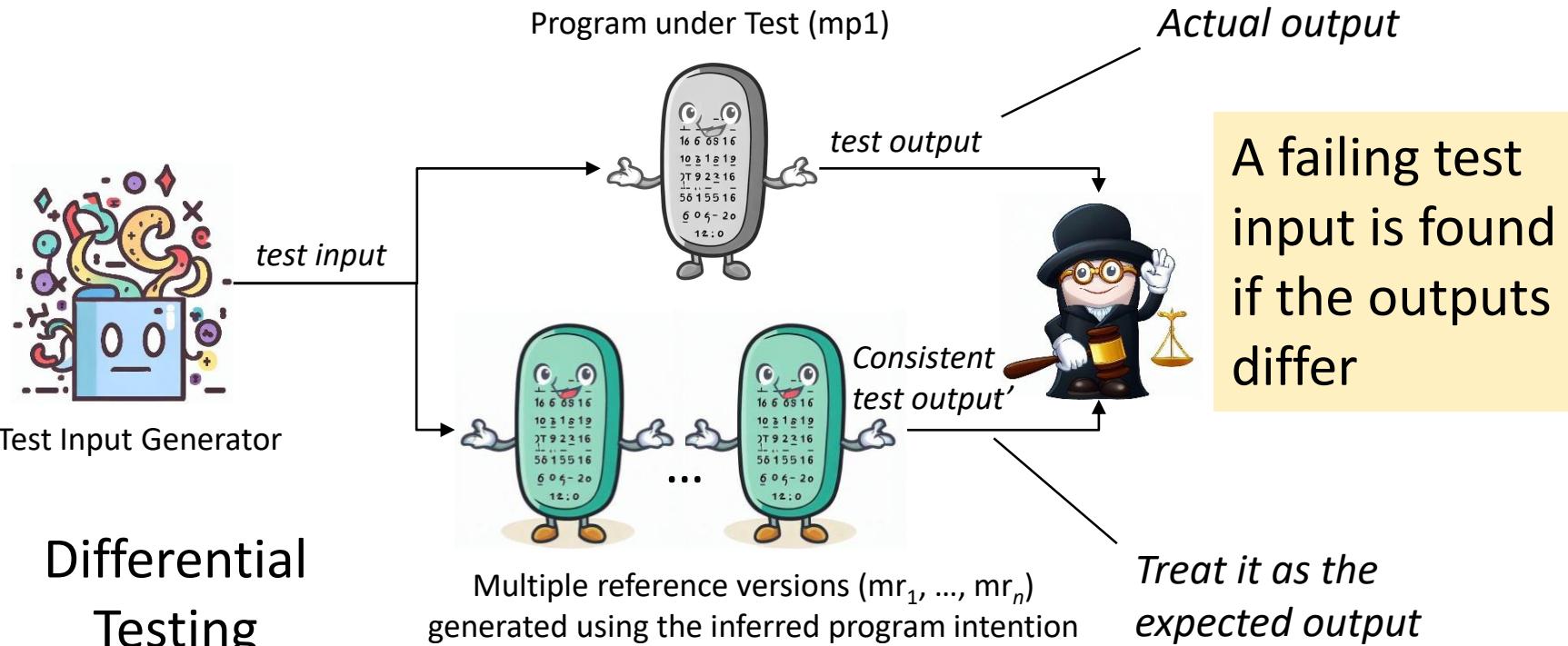
mr2

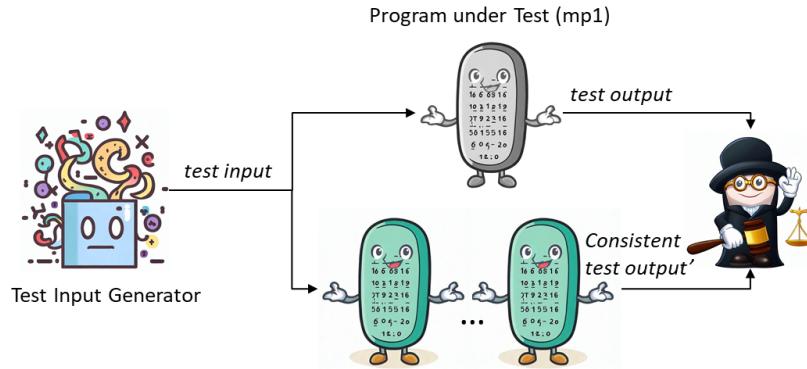
Test I/P	Prog Under Test O/P	Ref Program 1 O/P	Ref Program 2 O/P	Expected O/P
[-1, 2]	1	1	-inf	2
[-2, -4, -6, 0]	0	0	0	0
[1, 0, -2, 3, 4]	2	2	3	3
[4, -5, 2, 1, -1, 3]	4	5	5	5

They have different bugs and behaviors

Insight: Consistent outputs of the reference versions are likely to be correct

Differential Testing

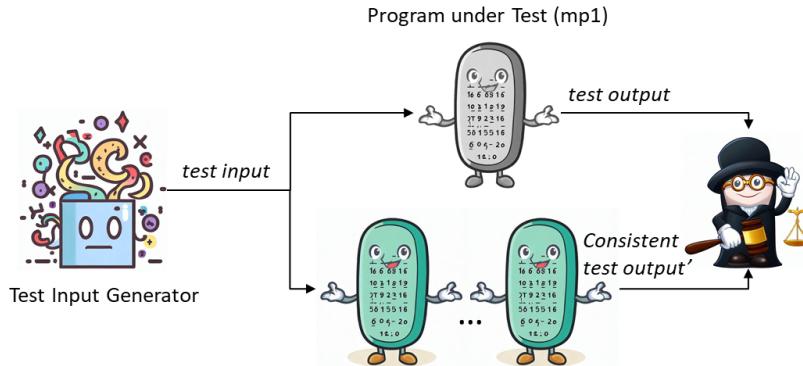




Differential Prompting:

A prompting framework that automates the process

Why are we excited?
It can be a new direction to approach the test oracle problem



Differential Prompting:

RQ1: How often can failing test cases be found?

RQ2: How often can program intention be inferred?

RQ3: How often can useful reference versions be generated?

Possible Outcome of a Test Case

Test Case Type	Test Outcome of PUT	failing I/P	Assertion
FailingTest-IA	Fail	✓	✓
FailingTest-la	Fail	✓	✗
FailingTest-ia	Fail	✗	✗
Passing / Invalid	Pass / Exception	?	?

The type of test cases that we are interested

A failing-inducing input is correct if it causes a failure in the correct version

An assertion input is correct if it does not reject executions of the correct version

RQ1: Failing-Inducing Test Cases (A)

	FT-IA	FT-la	FT-ia	Passing	Invalid	No Soln	Precision
Pynguin [ASE22]	8%	2%	0%	32%	56%	0%	NA
Direct Prompting	29%	19%	8%	30%	15%	0%	43.9%
Differential Prompting	75%	2%	0%	0%	0%	23%	97%

- Evaluation is based on all the 40 **buggy** Python programs in the QuixBugs dataset and gpt-turbo-3.5
- Experiments are repeated 10 times

Fail to find a test case where reference versions share an output different from that returned by the given program

Consider only the failing cases

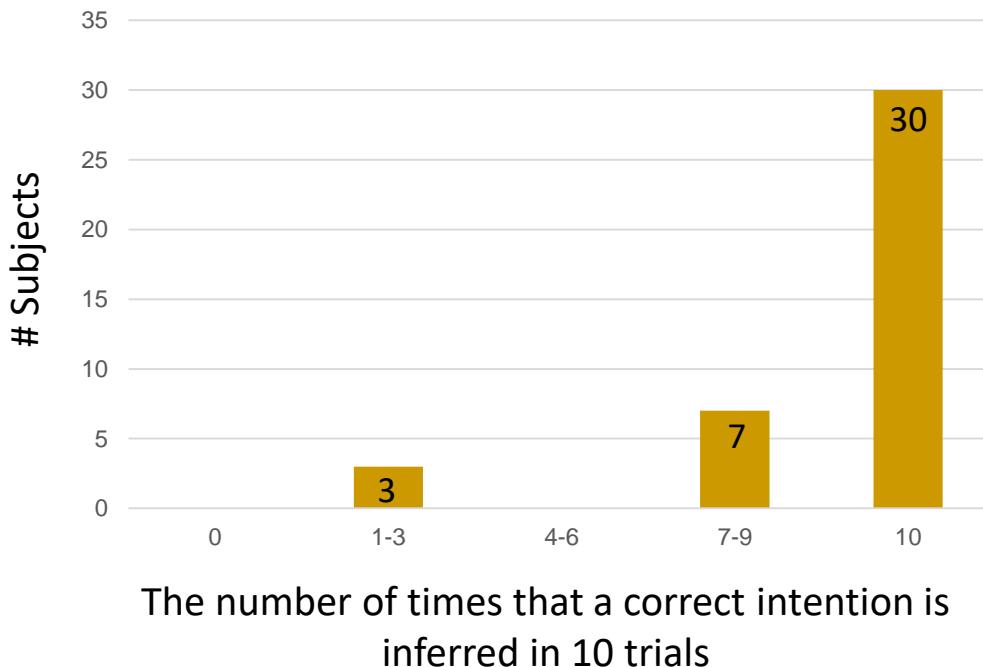
RQ1: Failing-Inducing Test Cases (B)

	FT-ia	Passing	Invalid	No Soln
Pynguin [ASE22]	0%	34%	67%	0%
Direct Prompting	22%	72%	4%	2%
Differential Prompting	3%	0%	0%	97%

- Evaluation is based on all the 40 **correct** Python programs in the QuixBugs dataset and gpt-turbo-3.5
- Experiments are repeated 10 times

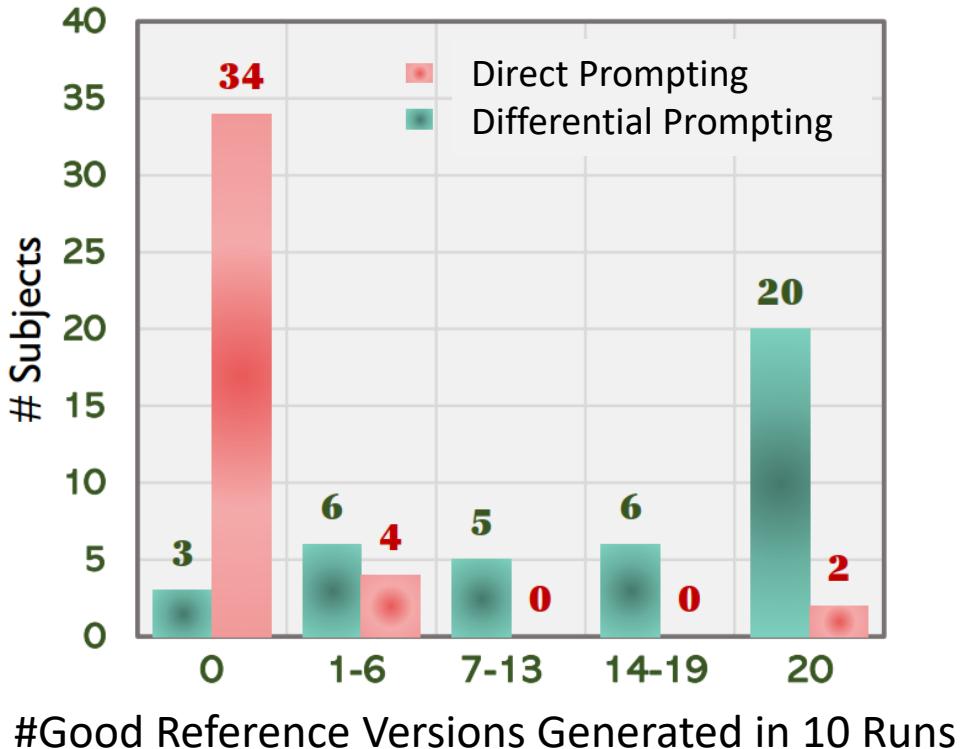
Fail to find a test case where reference versions share the same output that is different from that of the given program

RQ2: Inference of Program Intention



- There is a 100% (10/10) probability of inferring intentions correctly for 30 (out of 40) buggy QuixBugs subjects
- There is a 70%~90% probability of inferring intentions correctly for 7 buggy QuixBugs subjects
- There is a 10%~30% probability of inferring intentions correctly for 3 buggy QuixBugs subjects

RQ3: Reference Version Generation



Each Quixbugs buggy program comes with a set of bug revealing test cases. A reference version for a buggy program is **GOOD** if it passes these bug revealing test cases.

RQ4: Failing-Inducing Test Cases (A)

	FT-IA	FT-la	FT-ia	Passing	Invalid	No Soln	Precision
Pynguin [ASE22]	0%	0%	0%	30%	70%	0%	NA
Direct Prompting	7%	30%	44%	7%	11%	0%	9%
Differential Prompting	41%	0%	6%	0%	0%	53%	87%

- Evaluation is based on all the 7 **buggy** CodeForces subjects (in round #835 11/2022) by gpt-turbo-3.5
- Experiments are repeated 10 times

Fail to find a test case where reference versions share an output different from that returned by the given program

Comparison on Buggy Subjects

	FT-IA	FT-la	FT-ia	Passing	Invalid	No Soln	Precision
Pynguin [ASE22]	8%	2%	0%	32%	56%	0%	NA
Direct Prompting	29%	19%	8%	30%	15%	0%	43.9%
Differential Prompting	75%	2%	0%	0%	0%	23%	97%

Based on 40
Buggy QuixBugs
(before training
cutoff)

	FT-IA	FT-la	FT-ia	Passing	Invalid	No Soln	Precision
Pynguin [ASE22]	0%	0%	0%	30%	70%	0%	NA
Direct Prompting	7%	30%	44%	7%	11%	0%	9%
Differential Prompting	41%	0%	6%	0%	0%	53%	87%

Based on 7 buggy
CodeForces (after
training cutoff)

Comparison on Correct Subjects

	FT-ia	Passing	Invalid	No Soln
Pynguin [ASE22]	0%	34%	67%	0%
Direct Prompting	22%	72%	4%	2%
Differential Prompting	3%	0%	0%	97%

Correct QuixBugs
(before training
cutoff)

	FT-ia	Passing	Invalid	No Soln
Pynguin [ASE22]	0%	23%	77%	0%
Direct prompting	43%	20%	34%	3%
Differential Prompting	4%	0%	0%	96%

Correct CodeForces
(after training
cutoff)

Anatomy of Differential Prompting

- Common coding task assumption
- For those cases that work well, we find similar code at public repositories
- Otherwise, it may not work well
 - Inferred intention is either incomplete or contains errors
 - Fail to generate multiple good reference versions



A reference version is good if it delivers a correct output for fault-revealing tests

What's next?



```
def func(x):  
    ...  
    s(y) #common coding task  
    ...  
    return r
```

Diagram illustrating the analysis of the code snippet:

- An upward arrow labeled "backward analysis" points to the call to `s(y)`.
- A downward arrow labeled "forward analysis" points from the `s(y)` call to the `return r` statement.

- `s(y)` is a function that abstracts a code snippet implementing a common coding task
- Generate a failing test `s(v)` with an expected output `w`
- Find a value of `x` that results in calling `s(v)` – backward analysis
- Find the returning value due to `w` – forward analysis

What's next?

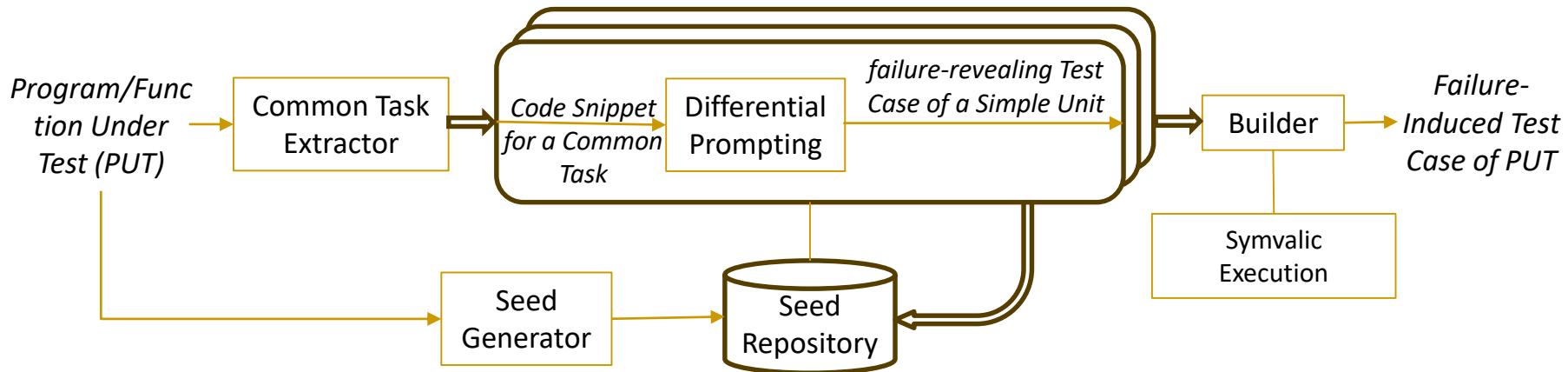


```
def func(x):  
    ...  
    s(y) #common coding task  
    ...  
    return r
```

backward analysis ↑
 ↓ *forward analysis*

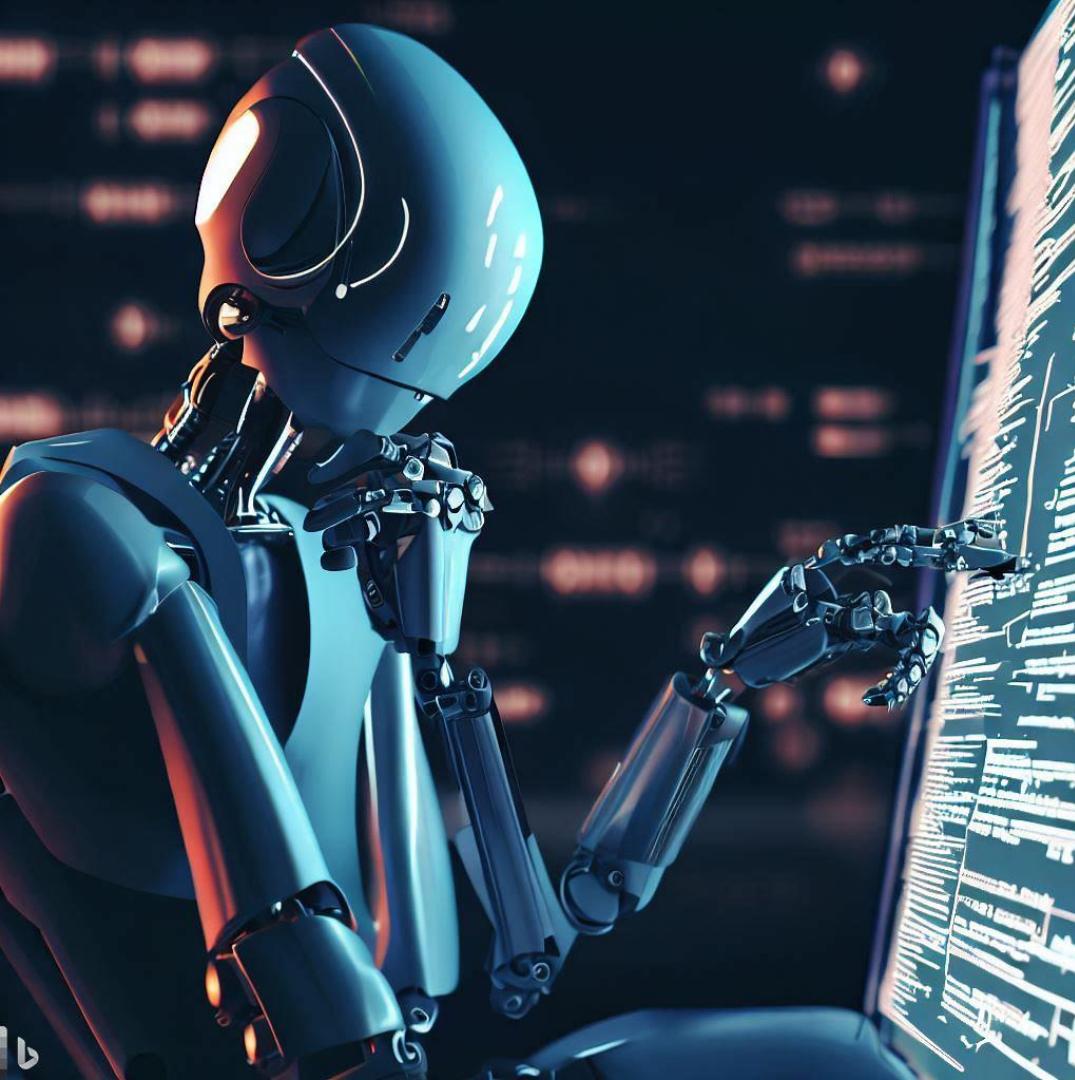
- So, we have a failing test case for `func()`
- It can be a big step towards addressing the test oracle problem
- Real-world software often contains small common code snippets [Weimer et al., ICSE09]

Failing Test Case Generation Framework



Summary

- A naïve use of LLMs is not effective in generating failing test cases
- LLMs have been pre-trained by common coding tasks
- LLMs are relatively insensitive to small perturbation in the code that they have often seen
- LLMs can infer the intention of a code snippet that implements common coding tasks
- We can apply differential testing to find failing test cases effectively



Can LLM be a
dependable copilot
for other common
coding tasks such as
program generation,
debugging and
repair?



Generate a Python program to compute the greatest common divisor recursively step-by-step.

- ✓ Searching for: python program to compute the greatest common divisor recursively step-by-step
- ✓ Generating answers for you...

Here is a Python program that computes the greatest common divisor recursively step-by-step:

```
def gcd(x, y):
    if y == 0:
        return x
    else:
        return gcd(y, x % y)

num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))

print("The GCD of", num1, "and", num2, "is", gcd(num1, num2))
```

```
def gcd(x, y):
    if y == 0:
        return x
    else:
        return gcd(y, x % y)
```

This program uses a recursive function to calculate the greatest common divisor (GCD) of two numbers. The GCD is the largest possible number that divides both numbers without leaving a remainder. The function takes two arguments `x` and `y`, and returns the GCD of those two numbers.

The function first checks if `y` is equal to 0. If it is, then the GCD is `x`. Otherwise, it calls itself recursively with the arguments `y` and `x % y`. This continues until `y` is equal to 0.

I hope this helps you! Let me know if you have any questions.

An attempt to generate a Python program to compute the greatest common divisor using Bing Chat

Explanation



Here is another Python program that computes the greatest common divisor recursively step-by-step:

```
def gcd(a,b):  
    if a == b:  
        return a  
    elif a > b:  
        return gcd(a-b,b)  
    else:  
        return gcd(a,b-a)  
  
num1 = int(input("Enter first number: "))  
num2 = int(input("Enter second number: "))  
  
print("The GCD of", num1,"and", num2,"is", gcd(num1, num2))
```



Asking Bing Chat for an alternative



This program also uses a recursive function to calculate the greatest common divisor of two integers `a` and `b`. If `a` is equal to `b` then `gcd(a,b)` is `a`; otherwise if `a > b`, then `gcd(a,b)` is `gcd(a-b,b)`; otherwise if `a < b`, then `gcd(a,b)` is `gcd(a,b-a)`.

LOOKS AMAZING, BUT ...



Please summarize p1

```
def p1(a, b):  
    if b == 0:  
        return a  
    else:  
        return p1(a, a % b)
```

What if seeking assistance for a buggy program?

The function `p1(a,b)` is a recursive function that takes two arguments `a` and `b`. If `b` is equal to 0, the function returns `a`. Otherwise, it calls itself with arguments `a` and `a % b`. This process continues until `b` is equal to 0. The function then returns the value of `a`.



Here is the code for the function:

```
def p1(a, b):  
    if b == 0:  
        return a
```

#correct version

```
def gcd(a, b):  
    if b == 0:  
        return a  
    else:  
        return gcd(b, a % b)
```

The `p1(a,b)` function ultimately returns the greatest common divisor (GCD) of two numbers `a` and `b`. The function uses the Euclidean algorithm to compute the GCD recursively. The Euclidean algorithm is based on the principle that the GCD of two numbers does not change if the larger number is replaced by its remainder when divided by the smaller number. The algorithm continues until the remainder is 0, at which point the GCD has been found.



The intention of the `p1(a,b)` function is to find the greatest common divisor (GCD) of two numbers `a` and `b`. The function uses the Euclidean algorithm to compute the GCD recursively. The Euclidean algorithm is based on the principle that the GCD of two numbers does not change if the larger number is replaced by its remainder when divided by the smaller number. The algorithm continues until the remainder is 0, at which point the GCD has been found .



Yes, the `p1(a,b)` function is correct and will return the greatest common divisor (GCD) of two numbers `a` and `b`. The function uses the Euclidean algorithm to compute the GCD recursively. The Euclidean algorithm is a well-known and widely used algorithm for computing the GCD of two numbers.

Can we trust LLM's assessment on our code?



Suggest a test case for p1. What value will be output by p1? What should be the expected output?

Sure! Here is a test case for p1:

Test case: a = 10 b = 25

Expected output: 5

The output of p1 when called with arguments a = 10 and b = 25 will be 5.

```
IDLE Shell 3.11.3
File Edit Shell Debug Options Window Help
Python 3.11.3 (tags/v3.11.3:f3909b8, Apr  4 2023,
Type "help", "copyright", "credits" or "license()"
>>> def p1(a, b):
...     if b == 0:
...         return a
...     else:
...         return p1(a, a % b)
...
>>> p1(10,25)
10
>>> |
```

Can we trust the evidence given by LLM?



The actual output
of p1(10, 25) is 10.

When $p1(0, 0)$ is executed, the function will enter the if statement and return a . The statement coverage of the test case is 100% because both the if and else statements are executed at least once.

The branch coverage of the test case is 50% because only one branch of the if-else statement is executed.

contradictory

```
def p1(a, b):  
    if b == 0:  
        return a  
    else:  
        return p1(a, a % b)
```

A 100% statement coverage of $p1$ implies 100% branch coverage for this program!



Suggest a failure-inducing test case for p1, indicating its expected output.

Sure! A failure-inducing test case for `p1` would be to call the function with arguments `a = 0` and `b = 0`. The expected output for this test case is undefined because the GCD of 0 and 0 is not well-defined.

Another failure-inducing test case would be to call the function with arguments `a = 10` and `b = -25`. The expected output for this test case is also undefined because the GCD of two negative numbers is not well-defined.

inconsistency



I apologize for the confusion. I just realized that I made a mistake in my previous response. A failure-inducing test case for `p1` would be to call the function with arguments `a = 10` and `b = 0`. The expected output for this test case is 10 because the GCD of any number and 0 is the number itself.

```
def p1(a, b):  
    if b == 0:  
        return a  
    else:  
        return p1(a, a % b)
```

Actually, `p1(10, 0)` is not a failure-inducing test case. It returns 10, which is the same as the expected output.



This implementation is incorrect. Please suggest a correct implementation of GCD.

I apologize for the confusion. Here's a correct implementation of the GCD function:

```
def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)
```



This implementation uses the Euclidean algorithm to calculate the GCD of two numbers. The algorithm works by repeatedly taking the remainder of the larger number divided by the smaller number until the remainder is 0. The GCD is then the last non-zero remainder.

Complete the following program that computes the greatest common divisor recursively step-by-step.

```
def gcd(a: int, b: int) -> int:  
    if a == 0:  
        return b  
    elif b == 0:  
        return a
```

Here is the completed Python program that computes the greatest common divisor recursively step-by-step:

```
def gcd(a: int, b: int) -> int:  
    if a == 0:  
        return b  
    elif b == 0:  
        return a  
    else:  
        return gcd(b, a % b)  
  
num1 = int(input("Enter first number: "))  
num2 = int(input("Enter second number: "))  
  
print("The GCD of", num1, "and", num2, "is", gcd(num1, num2))
```



Asking Bing Chat to complete the code for another implementation

```
def gcd(a: int, b: int) ->  
int:  
    if a == 0:  
        return b  
    elif b == 0:  
        return a  
    else:  
        return gcd(b, a % b)
```

This program uses a recursive function to calculate the greatest common divisor of two integers `a` and `b`. If `a` is equal to 0 then `gcd(a,b)` is `b`; if `b` is equal to 0 then `gcd(a,b)` is `a`; otherwise `gcd(a,b)` is `gcd(b,a%b)`.

Complete the following program that computes the greatest common divisor step-by-step.

```
def gcd(a: int, b: int) -> int:
```

```
    if a == 0:  
        return b
```

```
# fill in additional code here
```

```
    return gcd(b, a % b)
```

Here is the completed Python program that computes the greatest common divisor step-by-step:

```
def gcd(a: int, b: int) -> int:
```

```
    if a == 0:
```

```
        return b
```

```
    else:
```

```
        while b != 0:
```

```
            temp = b
```

```
            b = a % b
```

```
            a = temp
```

```
    return a
```

```
num1 = int(input("Enter first number: "))
```

```
num2 = int(input("Enter second number: "))
```

```
print("The GCD of", num1, "and", num2, "is", gcd(num1, num2))
```

May ignore the given constraint



```
def gcd(a: int, b: int) ->
```

```
int:
```

```
if a == 0:
```

```
    return b
```

```
elif b == 0:
```

```
    return a
```

```
else:
```

```
    return gcd(b, a % b)
```

Asking Bing Chat to
fill in code for a
partial
implementation

Complete the following program that recursively computes the greatest common divisor step-by-step.

```
def gcd(a: int, b: int) -> int:  
    if a == 0:  
        return b  
  
    # fill in additional code here  
  
    return gcd(b, a % b)
```

Another code filling attempt

Here is the completed Python program that recursively computes the greatest common divisor step-by-step:

```
def gcd(a: int, b: int) -> int:  
    if a == 0:  
        return b  
    else:  
        return gcd(b % a, a)  
  
num1 = int(input("Enter first number: "))  
num2 = int(input("Enter second number: "))  
  
print("The GCD of", num1, "and", num2, "is", gcd(num1, num2))
```

May ignore the given constraint



```
def gcd(a: int, b: int) ->  
int:  
    if a == 0:  
        return b  
    elif b == 0:  
        return a  
    else:  
        return gcd(b, a % b)
```

Preliminary Study

- RQ-1: Can LLMs generate correct code for common coding tasks?
- RQ-2: Can LLMs repair buggy programs for common coding tasks?
- RQ-3: Can LLMs summarize the intention of code appropriately?

Why Focusing on Common Coding Tasks?

- Weimer et al. [ICSE 2009] reported that **copying** of small code snippets is common among developers
- It means programs generally consist of many small but common tasks
 - Code clones are common
 - Developers are largely reluctant in coding from scratch!
 - Code copying/adaptation from public forums is common
- Code implementing common tasks is pervasive
- **LLMs have been well trained on these tasks**

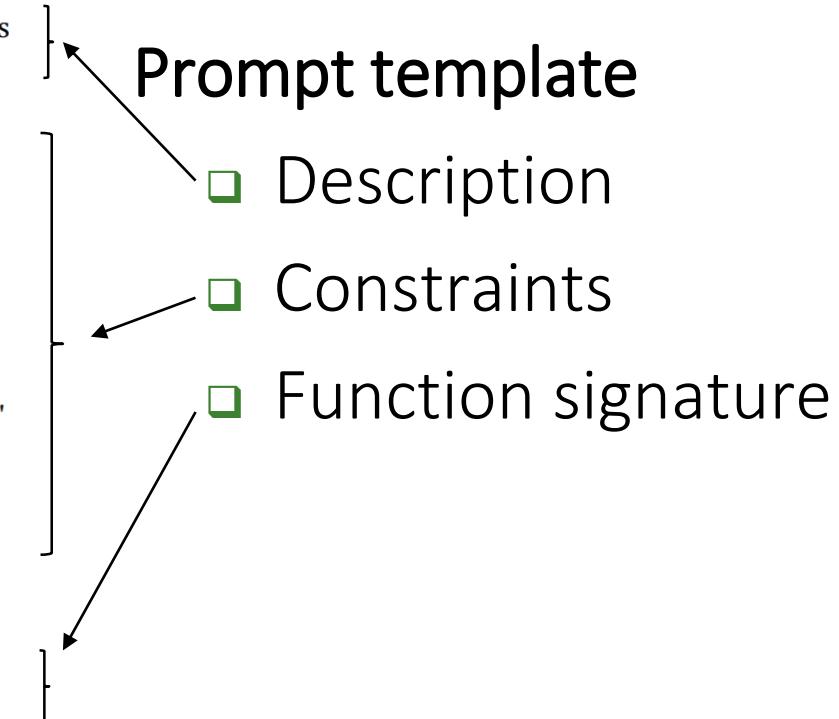


Two Benchmarks

- RQ-1: 240 ($=2*4*3*10$) programs from LeetCode
 - 2 periods (2016-2020, 2022)
 - 4 common programming tasks (Array, String, Hash table, Sorting)
 - 3 difficult levels (simple, medium, and hard)
 - 10 samples for each combination ($2*4*3$)
- RQ-2 & 3: 2442 correct +1783 buggy programs from Refactory (2019)
 - 5 common programming tasks (Sequential search, Unique dates, Duplicate elimination, Sorting, Top_k elements)
 - A benchmark used for program repair

RQ-1: Code Generation Performance

```
1 Convert a non-negative integer `num` to its English words
2 representation.
3
4 **Example 1:** 
5   Input: num = 123   Output: "One Hundred Twenty Three"
6 **Example 2:** 
7   Input: num = 12345  Output: "Twelve Thousand Three
8 Hundred Forty Five"
9 **Example 3:** 
10  Input: num = 1234567  Output: "One Million Two
11 Hundred Thirty Four Thousand Five Hundred Sixty Seven"
12
13 **Constraints:** 
14 * `0 <= num <= 2^31 - 1`
15
16 ...
17 class Solution:
18     def numberToWords(self, num: int) -> str:
19 ...
20 Implement the above task in Python.
```



Three LLMs

- ChatGPT (GPT)
 - Released in November 2022
- Codex (Dex)
 - Released in June 2021
- CodeGen (Gen)
 - Released in March 2022
 - Open-source local model

RQ-1: Code Generation Performance

Level	Type	Correctness			Correctness		
		GPT	Dex	Gen	GPT	Dex	Gen
Easy	Array	10 (1.00)	10 (0.28)	2 (0.04)	8 (0.66)	4 (0.12)	1 (0.02)
	String	10 (0.98)	7 (0.24)	2 (0.04)	9 (0.48)	2 (0.10)	0 (0.00)
	HashTable	10 (1.00)	7 (0.36)	3 (0.14)	9 (0.74)	3 (0.08)	1 (0.02)
	Sorting	10 (0.94)	8 (0.42)	2 (0.06)	7 (0.56)	2 (0.10)	1 (0.04)
Total		40 (0.98)	32 (0.33)	9 (0.07)	33 (0.61)	11 (0.10)	3 (0.02)
Medium	Array	9 (0.80)	5 (0.18)	0 (0.00)	1 (0.08)	1 (0.02)	0 (0.00)
	String	9 (0.66)	5 (0.16)	1 (0.02)	1 (0.10)	0 (0.00)	0 (0.00)
	HashTable	9 (0.80)	5 (0.20)	1 (0.02)	5 (0.38)	2 (0.08)	0 (0.00)
	Sorting	10 (0.98)	5 (0.14)	1 (0.02)	3 (0.18)	0 (0.00)	0 (0.00)
Total		37 (0.81)	20 (0.17)	3 (0.02)	10 (0.18)	3 (0.03)	0 (0.00)
Hard	Array	5 (0.38)	3 (0.08)	0 (0.00)	1 (0.06)	0 (0.00)	0 (0.00)
	String	10 (0.50)	2 (0.04)	0 (0.00)	2 (0.08)	0 (0.00)	0 (0.00)
	HashTable	7 (0.54)	3 (0.10)	0 (0.00)	0 (0.00)	0 (0.00)	0 (0.00)
	Sorting	8 (0.46)	1 (0.02)	0 (0.00)	0 (0.00)	0 (0.00)	0 (0.00)
Total		30 (0.47)	9 (0.06)	0 (0.00)	3 (0.03)	0 (0.00)	0 (0.00)

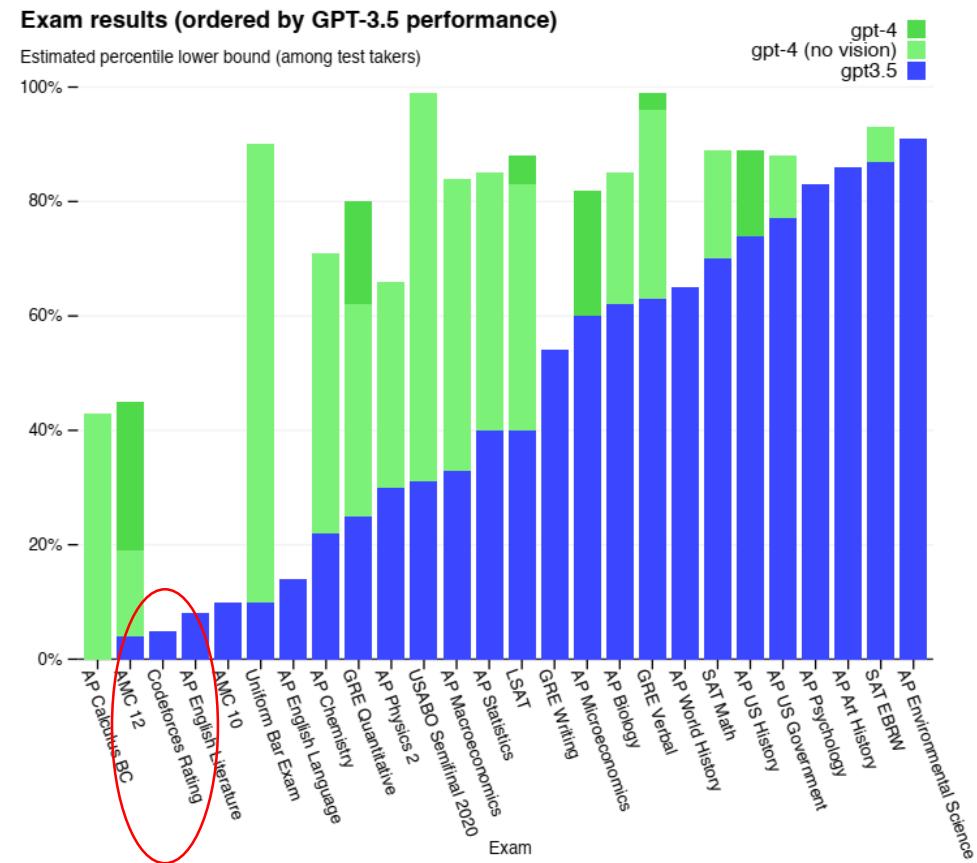
- Left: LeetCode (2016-20)
- Right: LeetCode (2022)
- Each entry represents the experimental result on 10 LeetCode problems
- Attempt 5 times for each problem. Consider correct if a correct program is generated
- **8 (0.66) means**
 - A correct program is generated in some attempt for 8/10 problems
 - 0.66 means 66% of the programs generated on average are correct

OpenAI Report

Exam	GPT-4	GPT-3.5
Leetcode (easy)	31/41	12/41
Leetcode (medium)	21/80	8/80
Leetcode (hard)	3/45	0/45

In line with the finding
by OpenAI

Source: <https://cdn.openai.com/papers/gpt-4.pdf>



RQ-2: Program Repair on Refactory Benchmark

Problems	#LoC	#Buggy	GPT (Top-5)	GPT (AVG-5)	Dex (Top-5)	Dex (AVG-5)
Seq Search	10	575	89.0%	17.8%	63.3%	14.3%
Uniq Dates	28	435	74.0%	14.9%	61.4%	12.5%
Dup Elim	7	308	97.7%	19.5%	83.8%	17.9%
Sorting Tup	9	357	87.7%	17.5%	58.3%	12.0%
Top-k Elm	11	108	97.2%	19.3%	50.0%	15.7%
<i>Overall</i>	<i>14</i>	<i>1783</i>	<i>87.1%</i>	<i>17.4%</i>	<i>64.6%</i>	<i>14.1%</i>

- Slight performance improvement if the prompts include problem description
- Refactory achieves an overall repair rate of 90.8% (1 min time budget) on common coding tasks

RQ-2: Program Repair on Refactory Benchmark

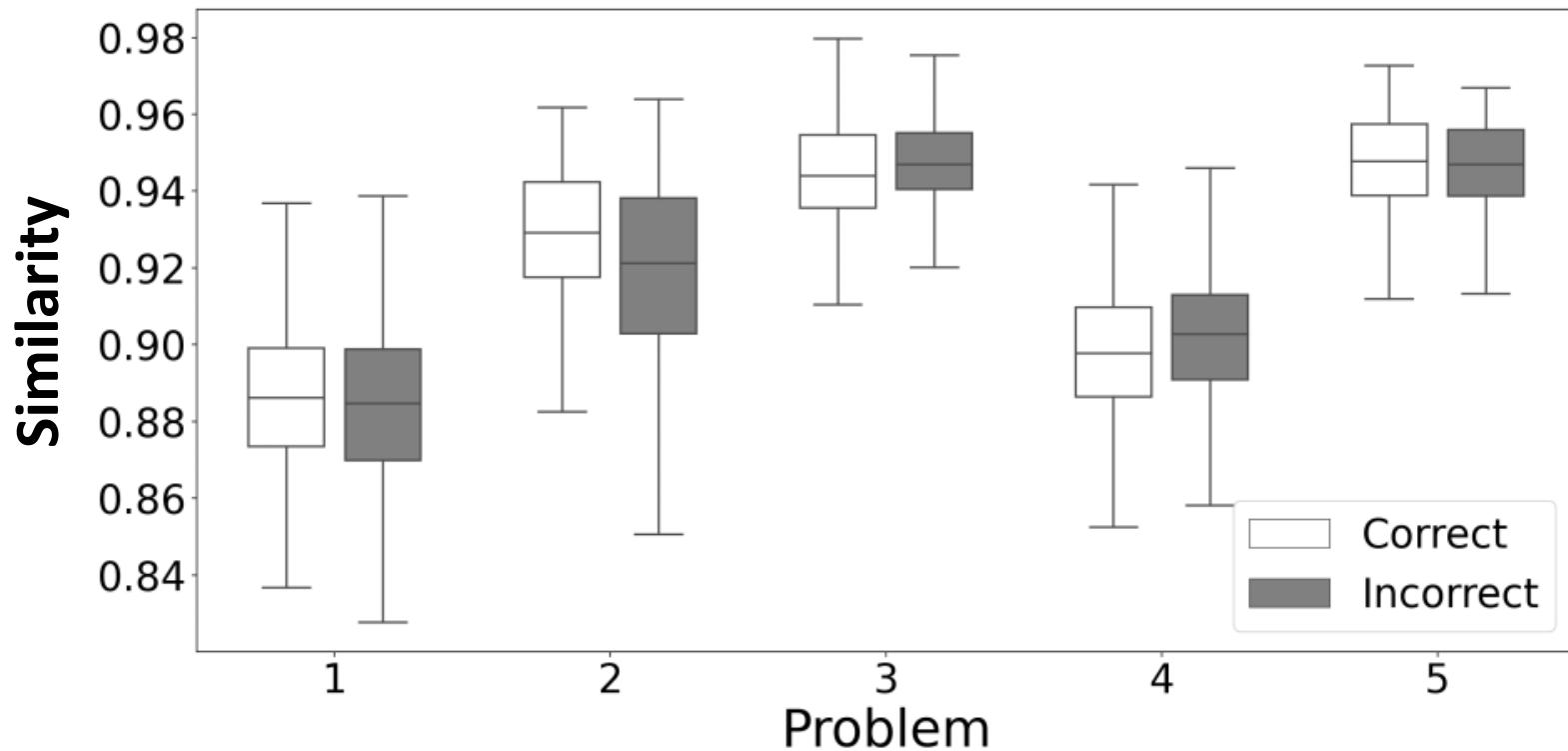
Problems	#LoC	#Buggy	GPT (Top-5)	GPT (AVG-5)	Dex (Top-5)	Dex (AVG-5)
Seq Search	10	575	89.0%	17.8%	63.3%	14.3%
Uniq Dates	28	435	74.0%	14.9%	61.4%	12.5%
Dup Elim	7	308	97.7%	19.5%	83.8%	17.9%
Sorting Tup	9	357	87.7%	17.5%	58.3%	12.0%
Top-k Elm	11	108	97.2%	19.3%	50.0%	15.7%
<i>Overall</i>	<i>14</i>	<i>1783</i>	<i>87.1%</i>	<i>17.4%</i>	<i>64.6%</i>	<i>14.1%</i>

- LLMs apparently capable of repairing common coding tasks
- Multiple attempts are often needed for finding a correct repair
- High quality test suites play an important role

RQ-3: Code Summarization

- Intuitively,
 - S_c : Similarity between the code summary of a correct implementation and that of the original problem description
 - S_i : Similarity between the code summary of an incorrect implementation and that of the original problem description
- Compute similarity S_c and S_i based on the embeddings generated by Bert
- If the intuition holds, $\text{distribution}(S_c) \neq \text{distribution}(S_i)$

RQ-3: Similarity Distribution



RQ-3: Summarization of Incorrect Code

```
def remove_extras(lst):
    t=[]
    for i in lst:
        if i not in t:
            t.append(i)
```



**Missing return statement
Does not return anything**



The function removes duplicate elements from a list and returns a new list containing only the unique elements.

RQ-3: Incorrect Code Summary Example

```
def remove_extras(lst):
    t=[]
    for i in lst:
        if i not in t:
            t.append(i)
```



- Our experimentation finds that ChatGPT is able to generate a code summary similar to the **original problem description** for buggy implementations.
- It suggests that ChatGPT may make up a code summary based on training samples similar to the given buggy code for common tasks



The function removes duplicate elements from a list and returns a new list containing only the unique elements.

RQ1: Summary on Code Generation

- ChatGPT outperforms state-of-the-art LLM approaches for common tasks
- ChatGPT is capable of generating (nearly-)correct code for simple common tasks
- Test suites play an important role in assessing the quality of generated code
- It helps if problem description is precise and concise

RQ2: Summary on Program Repair

- Comparable performance to state-of-the-arts
- Can repair many buggy programs for simple common tasks within five attempts
- But the repaired programs may not exercise the minimal changes necessary to repair the code
- ChatGPT is a helper to programmer, but not yet an autonomous, dependable coding assistant

RQ3: Summary on Code Summarization

- May not always offer full summarization of a given program
- Can generate summarization that hints the expected intention of a given buggy program for simple common tasks
- Research opportunity: Can we leverage the hint to generate test oracles for simple common tasks?

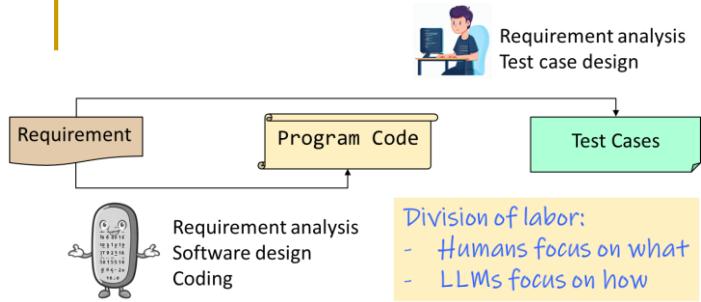
Summary

Findings

- LLMs can perform common coding tasks
- However, they often make up stories (or properties) based on the code that LLMs have learnt
- Using LLMs alone to solve coding tasks (even the common ones) is not effective

Advice

- Avoid using LLMs alone
- Use LLMs mainly on common tasks
- Always validate (test) LLMs' answers
- Augmenting LLMs with existing SE techniques can significantly boost its dependability



Reduced Coding Effort

Test 1
Input: []
Expected Output: 0
Test 2
Input: [-1, 2]
Expected Output: 2
Test 3
Input: [-2, -4, -6, -5]
Expected Output: 0
Test 4
Input: [4, -5, 2, 1, -1, 3]
Expected Output: 5
...



Implement a Python function that takes a list of integers nums as input and returns an integer that represents the largest possible sum of any contiguous sublist of nums. In case that the maximum subsequence is not unique, output the one with the smallest indices. If all the numbers in nums are negative, then its maximum sum is defined to be 0. For example, given an input [4, -5, 2, 1, -1, 3], its maximum sum is given by the sublist [2, 1, -1, 3] with the largest sum being 5.

Increased Requirement Specification (Prompting) Effort

Impact on Software Development Practice

Test 1
Input: []
Expected Output: 0
Test 2
Input: [-1, 2]
Expected Output: 2
Test 3
Input: [-2, -4, -6, -5]
Expected Output: 0
Test 4
Input: [4, -5, 2, 1, -1, 3]
Expected Output: 5
...



Increased Test Design Effort

```
def max_sublist_sum(nums):
    if not nums:
        return 0
    max_sum = nums[0]
    for num in nums:
        max_sum = max(max_sum + num, num)
    return max_sum
```

```
def max_sublist_sum(nums):
    cur_max = max_sum = 0
    for num in nums:
        cur_max = max(cur_max + num, num)
        max_sum = max(max_sum, cur_max)
    return max_sum
```

Failure-Driven Test Generation

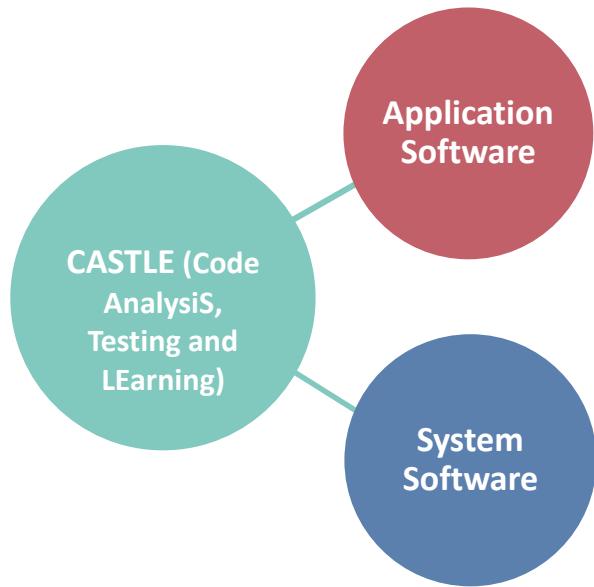
#	Model	pass@1
1	🥇 GPT-4-Turbo (Nov 2023) 🟡	⚡ 81.7
2	🥈 GPT-4 (May 2023) 🟡	⚡ 76.8
3	🥉 WizardCoder-33B-V1.1 🟡	⚡ 73.2
4	DeepSeek-Coder-33B-instruct 🟡	⚡ 72.6
5	Magicoder-S-DS-6.7B 🟡	⚡ 70.7
6	DeepSeek-Coder-6.7B-instruct 🟡	⚡ 70.1
7	speechless-codellama-34B-v2.0 🟡	⚡ 68.3
8	code-millenials-34B 🟡	⚡ 67.7
9	Phind-CodeLlama-34B-v2	⚡ 67.1
10	Magicoder-S-CL-7B 🟡	⚡ 66.5
11	GPT-3.5-Turbo (Nov 2023) 🟡	⚡ 65.9

<https://evalplus.github.io/leaderboard.html>

Food for Thoughts

- Does code generated by LLMs need to be comprehensible?
 - If so, what degree of comprehensibility is needed
 - If not, who will be responsible to review the generated code
- Does code generated by LLMs need to be maintainable?
- What can happen if future LLMs are trained using datasets collected from code repositories that are largely authored by LLMs?
- What can we do if LLMs refuse to further revise an earlier version after saying sorry or ask for a new task instead?
- Can LLMs assess the quality of a piece of code (or work) with justification explained by a chain of thought for management's reference?
- How likely will software engineers need to live with LLMs?

Prior Efforts on LLM-assisted Software Code Quality Management



- **Test Generation** (test input, stub, flakiness, test oracle, LLM synthesized code) [TOSEM23c, ASE23]
- **Test Migration**
- **Mobile Apps** [ASE21a, ICSE22a, TSE23a, ISSTA23a, ISSTA23c]
- **Code Synthesis** [TOSEM23b]
- **Fault Diagnosis** [TSE21]
- **Smart Contracts** [ASE21b, ESEC/FSE21a, TSE23c]
- **Deep Learning Systems** [EmSE21, ESEC/FSE21b, ICSE22b, TOSEM22, ESEC/FSE23b]

- **Compilers / Deep Learning Compilers** [ESEC/FSE21c, ISSTA23b]
- **Deep Learning Libraries** [TOSEM23a]
- **Software Supply Chain** [ICSE21a, ICSE22c, ASE22, TSE22, TSE23b, ESEC/FSE23a]
- **Others** [TSE21a, TSE21b, TSC21a, TSC21b, ICSE21b, USENIX21, USENIX22, JSS23, ESEC/FSE23c]