

# An Analysis of the Automatic Bug Fixing Performance of ChatGPT

Seminar on Artificial Intelligence 2

Maroš Bratko 14.3.2023

# An Analysis of the Automatic Bug Fixing Performance of ChatGPT

Dominik Sobania

Johannes Gutenberg University Mainz

Email: dsobania@uni-mainz.de

Martin Briesch

Johannes Gutenberg University Mainz

Email: briesch@uni-mainz.de

Carol Hanna

University College London

Email: carol.hanna.21@ucl.ac.uk

Justyna Petke

University College London

Email: j.petke@ucl.ac.uk

**Abstract**—To support software developers in finding and fixing software bugs, several automated program repair techniques have been introduced. Given a test suite, standard methods usually either synthesize a repair, or navigate a search space of software edits to find test-suite passing variants. Recent program repair methods are based on deep learning approaches. One of these novel methods, which is not primarily intended for automated program repair, but is still suitable for it, is ChatGPT. The bug fixing performance of ChatGPT, however, is so far unclear. Therefore, in this paper we evaluate ChatGPT on the standard bug fixing benchmark set, QuixBugs, and compare the perfor-

straint solver to synthesize repairs [3]. The generate-and-validate ones have first seen industrial uptake [4]. One of the key disadvantage of standard approaches to APR is their running cost. The generate-and-validate ones usually rely on test suites to verify program correctness, while synthesis-based ones on calls to a constraint solver. Both validation strategies are costly, making typical APR tools hours to run before a viable patch is presented to the developer.

Most recently, program repair tools based on deep learn-

SE] 20 Jan 2023

# Table of contents

- Overview of methods for *Automated Program Repair (APR)*
  - standard methods
  - deep-learning based methods
- Comparison of chatGPT with other models
  - Benchmark
  - Methodology
  - Results
  - Dialogue hints

# Standard methods for APR

- *semantics-driven (or synthesis-based)*
  - use a constraint solver to synthesize repairs
- *generate-and-validate*
  - mutate software guided by a search strategy (Genetic Improvement - GI)
  - have first seen industrial uptake [1]
  - in [1], authors suggest a system, that tracks bugs during a workday, and searches for potential fixes overnight
- Both validation strategies are costly (run for hours)

---

[1] Haraldsson, Sæmundur & Woodward, John & Brownlee, Alexander & Siggeirsdottir, Kristin. (2017). Fixing Bugs in Your Sleep: How Genetic Improvement Became an Overnight Success. [10.1145/3067695.3082517](https://doi.org/10.1145/3067695.3082517).

# Genetic Improvement (GI)

- uses computational search methods to improve existing software [1]

Description	Operations
Numerical constants	Can increment by $\pm 1$
Arithmetic operators	$+$ , $-$ , $*$ , $/$ , $//$ , $\%$ , $**$
Arithmetic assignments	$+ =$ , $- =$ , $* =$ , $/ =$ ,
Relational operators	$<$ , $>$ , $< =$ , $> =$ , $==$ , $! =$ , <i>is</i> , <i>is not</i> , <i>not</i>
Logical operators	<i>and</i> , <i>or</i>
Logical constants	<i>True</i> , <i>False</i>

**Table 1: Sets of single operators available to the GI. One member of a given set can be changed to another member of the same set. [1]**

---

[1] Haraldsson, Sæmundur & Woodward, John & Brownlee, Alexander & Siggeirsdottir, Kristin. (2017). Fixing Bugs in Your Sleep: How Genetic Improvement Became an Overnight Success. 10.1145/3067695.3082517.

# Genetic Improvement (GI)

```
if (a && b) {  
    c = 1;  
} else {  
    c = 0;  
}
```



```
if (a || b) {  
    c *= 2;  
} else {  
    c /= -1;  
}
```

# Genetic Improvement (GI)

## Genetic Algorithm - Individual [1]

(Operation, [Location], "Code\_out", "Code\_in")

(Replace, [13,26], "<", "<=")

(Replace, [10,12], "False", "True")

# GI Community

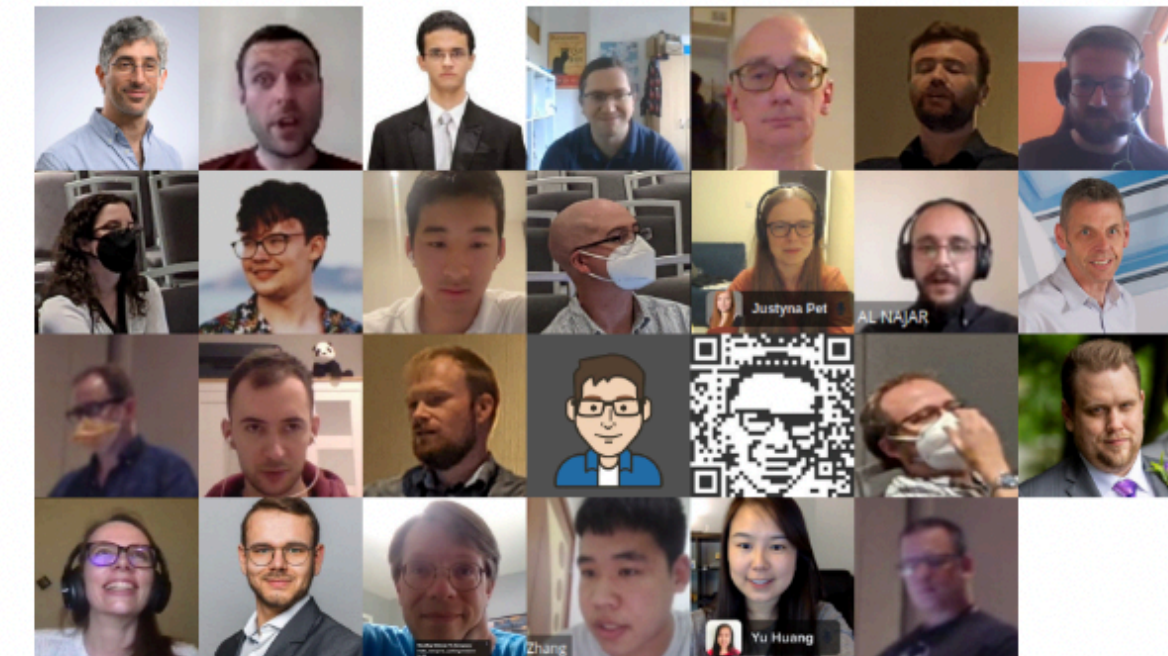
**HOT** 20 May 2023 (Sat): 12th GI workshop ([event page](#)) at ICSE 2023

## Genetic Improvement of Software *.com*

Welcome to the community website on Genetic Improvement (GI).

GI focuses on the repair and optimisation of software using computational search and evolutionary processes, often [outperforming](#) human developers at these tasks. ([read more](#))

We as a community run a workshop and frequent [events](#) year-round, feel free to meet [us](#) to discuss the future of software engineering!



GI@GECCO 2022 (hybrid event)

## News

- The [11th International Workshop on Genetic Improvement](#) took place at [GECCO 2022](#), together with a [GI tutorial](#).
- The [10th International Workshop on Genetic Improvement](#) took place at [ICSE 2021](#).
- There was a [GI tutorial at ASE 2020](#).
- There was a [tutorial on Genetic Improvement](#) at PPSN 2020. [Slides can be obtained here](#).
- Version 2 of the Genetic Improvement in No time (Gin) tool has been released. It is available at <https://github.com/gintool/gin>. It provides the ability to handle multi-class projects, gradle and maven build tools, built-in profiler and sampling utilities, test case generation, and a load of new edit operators.



# Deep learning based program repair tools

- learn bug fixing patterns from existing databases and treat the automated program repair problem as a neural machine translation task
- DL-based program repair has shown competitive results to standard approaches [1]
- several large-scale language models based on the Transformer architecture [2] have been introduced
  - CoCoNut [3]
  - Codex (by OpenAI) → Github Copilot
  - chatGPT (by OpenAI)

---

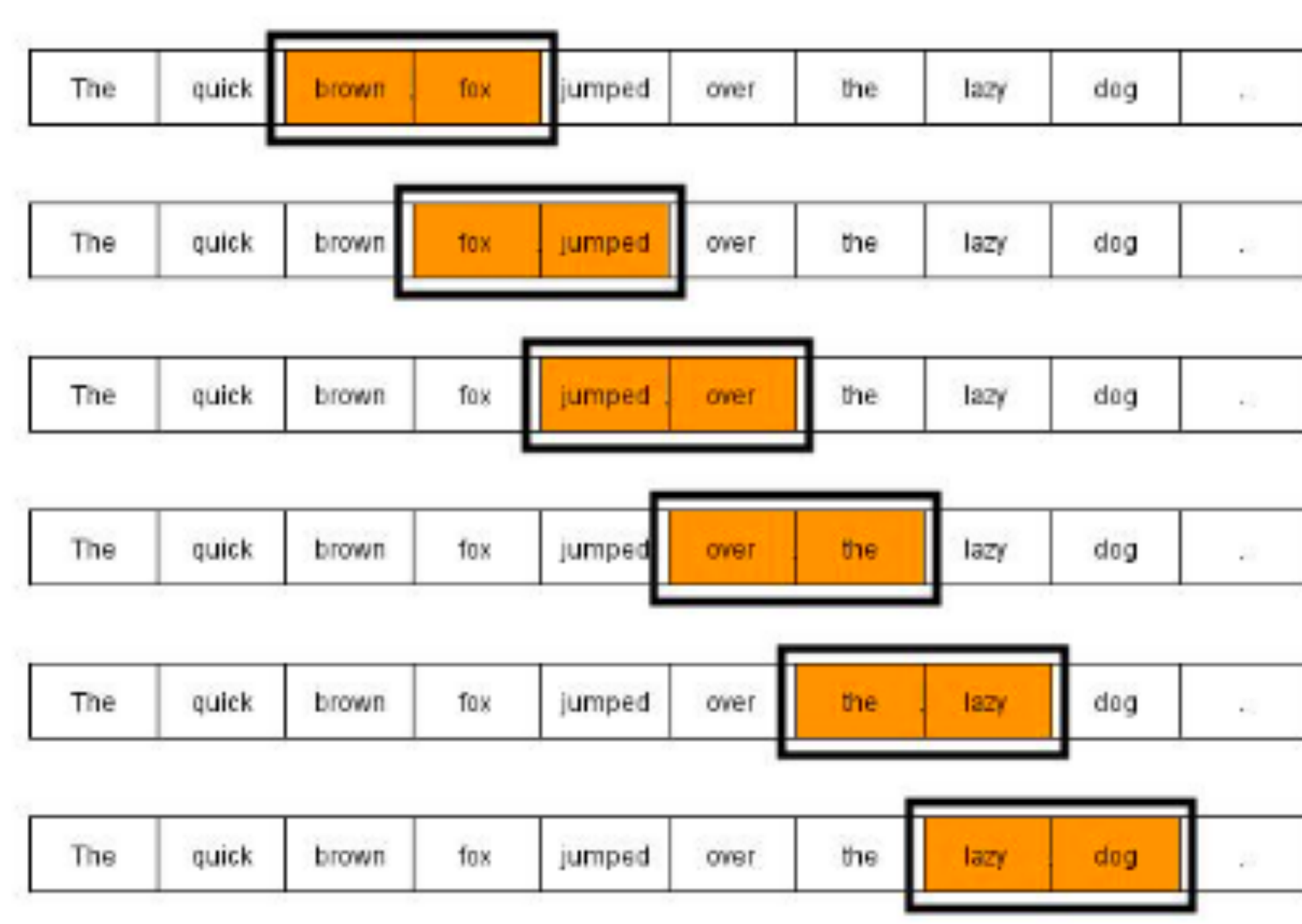
[1] Zhang, Qunjun, et al. "A Survey of Learning-based Automated Program Repair." arXiv preprint arXiv:2301.03270 (2023).

[2] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

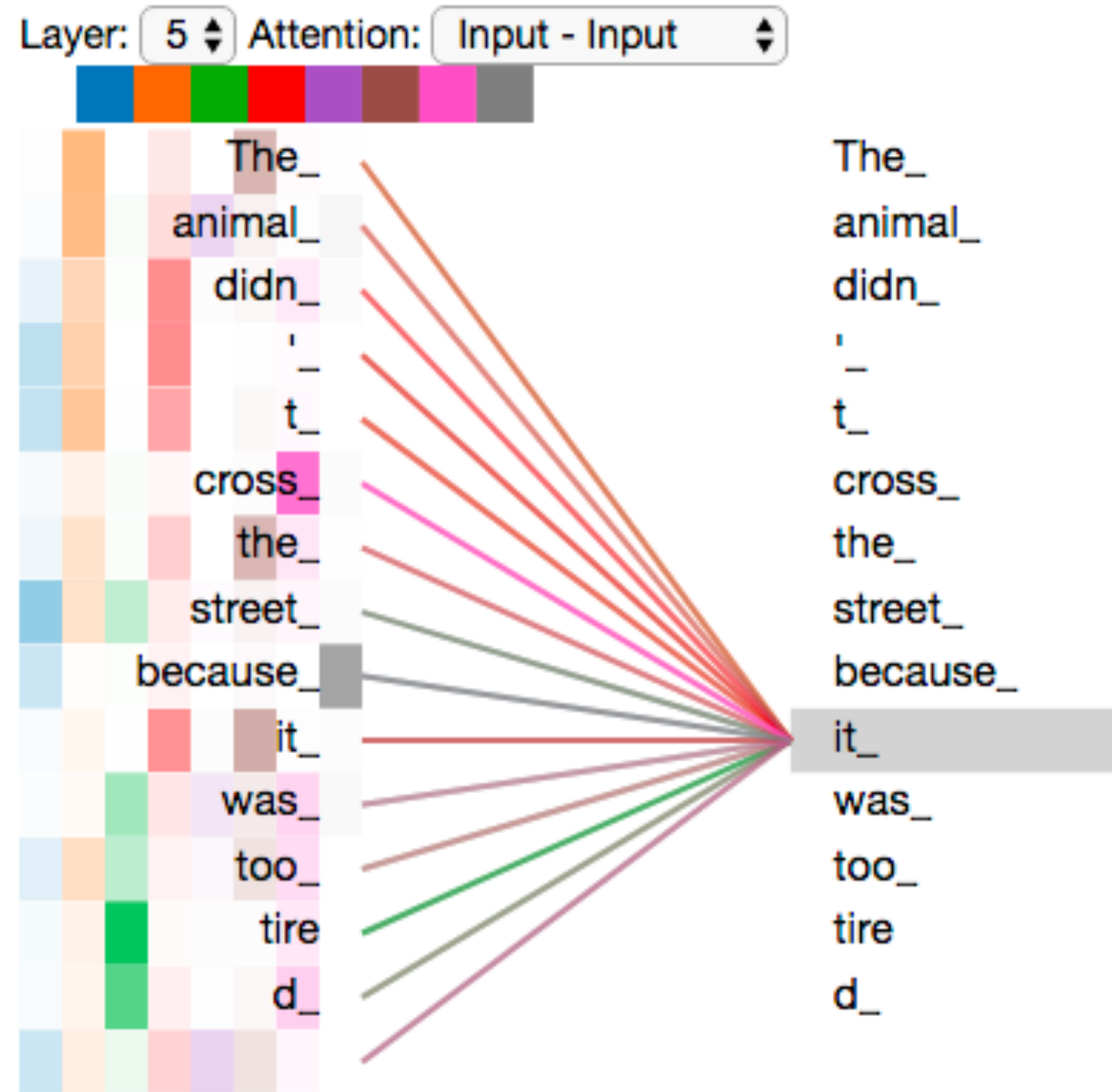
[3] Lutellier, Thibaud, et al. "Coconut: combining context-aware neural translation models using ensemble for program repair." *Proceedings of the 29th ACM SIGSOFT international symposium on software testing and analysis*. 2020.

# Bigram language model

- predicts the next token only by the current token

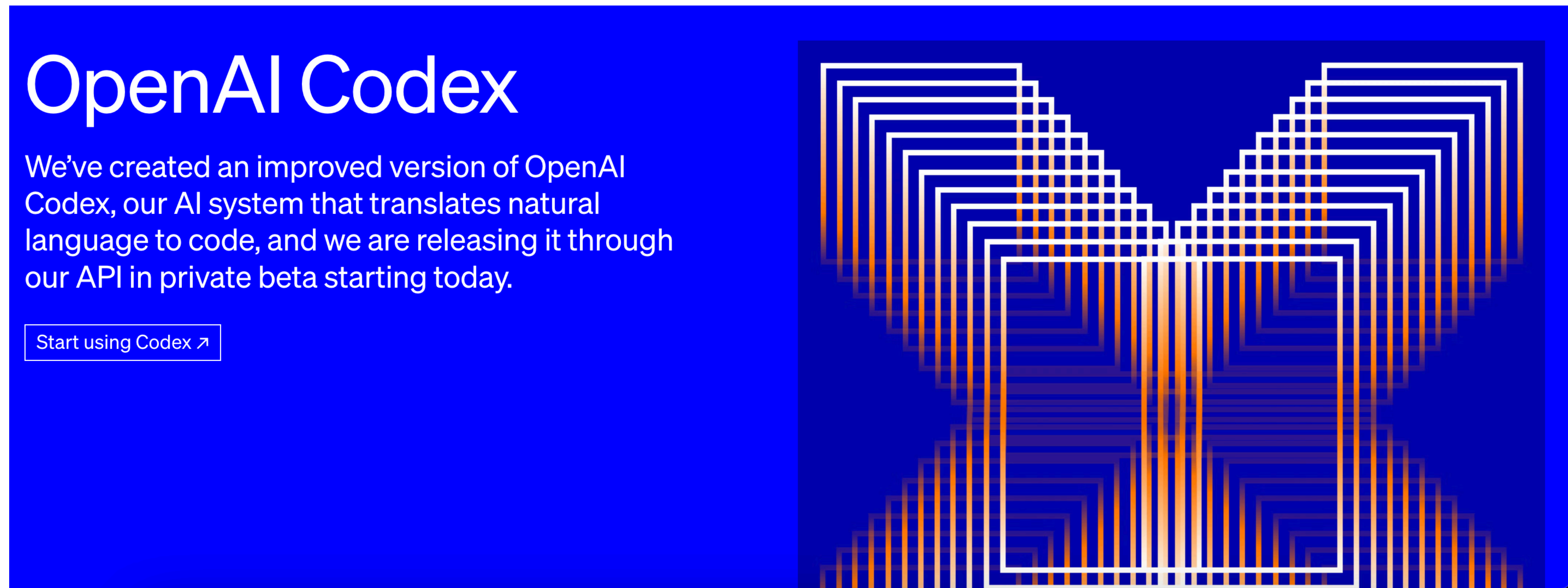


# Transformer



# Codex

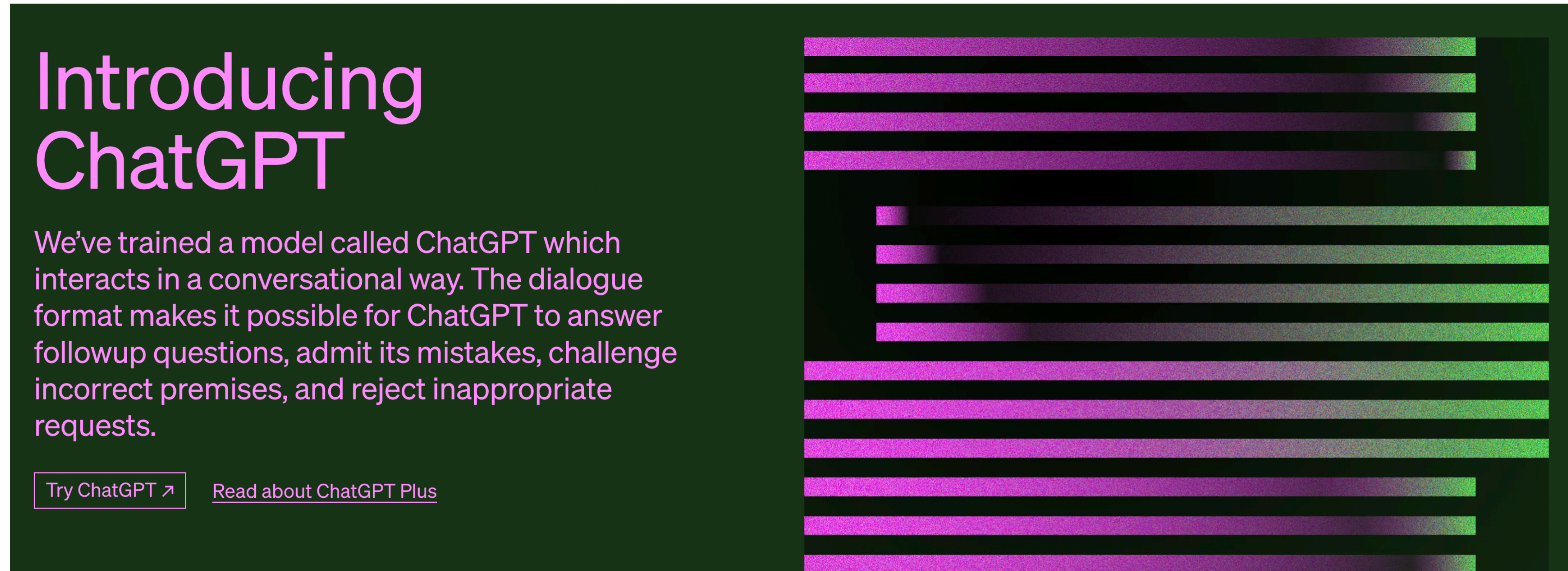
- The Codex model series is a descendant of GPT-3 series that's been trained on both natural language and billions of lines of code. [1]



[1] OpenAI API. (n.d.). Retrieved March 7, 2023, from <https://platform.openai.com/docs/guides/code>

# chatGPT

- ChatGPT is powered by *gpt-3.5-turbo*, OpenAI's most advanced language model.
- it is possible to have a conversation with language model and the previous chat history is taken into account



# Comparison

# QuixBugs benchmark

- A multi-lingual program repair benchmark set based on the Quixey Challenge
- small, yet challenging programs (40 in total)
- <https://github.com/jkoppel/QuixBugs>

jkoppel / QuixBugs Public

Watch 6 Fork 28 Star 46

<> Code Issues 3 Pull requests Actions Projects Security Insights

master 1 branch 0 tags

Go to file Add file <> Code

drckln Merge pull request #52 from h4iku/add-manual-run	4257f44 on Aug 29, 2022	145 commits
.github/workflows	Update ci.yml	7 months ago
correct_java_programs	Fix FLATTEN indentation	7 months ago
correct_python_programs	Added test cases for shortest path length for python.	6 years ago
java_programs	Fix tab/space consistency in indentations	7 months ago
java_testcases	Merge pull request #48 from h4iku/fix-indent	7 months ago
json_testcases	Add pytest files for the Python version	7 months ago
python_programs	Fixed TabError	last year
python_testcases	Add pytest files for the Python version	7 months ago

**About**  
A multi-lingual program repair benchmark set based on the Quixey Challenge

Readme  
MIT license  
46 stars  
6 watching  
28 forks

**Releases**  
No releases published

[1] D. Lin, J. Koppel, A. Chen, and A. Solar-Lezama, “QuixBugs: A multi-lingual program repair benchmark set based on the Quixey Challenge,” in Proceedings Companion of the 2017 ACM SIGPLAN international conference on systems, programming, languages, and applications: software for humanity, 2017, pp. 55–56.

# QuixBugs

## Bitcount

```
def bitcount(n):  
    count = 0  
    while n:  
        n ^= n - 1  
        count += 1  
    return count
```



```
def bitcount(n):  
    count = 0  
    while n:  
        n &= n - 1  
        count += 1  
    return count
```



# QuixBugs

## Bitcount

```
1
2 def bitcount(n):
3     count = 0
4     while n:
5         n ^= n - 1
6         count += 1
7     return count
8
9
10 """
11 Bitcount
12 bitcount
13
14
15 Input:
16     n: a nonnegative int
17
18 Output:
19     The number of 1-bits in the binary encoding of n
20
21 Examples:
22     >>> bitcount(127)
23     7
24     >>> bitcount(128)
25     1
26 """
```

# Evaluation

- remove contained comments (comments sometimes contains the solution)
- ask chatGPT if the code contains a bug and how it can be fixed
  - manually check whether the given answer is correct or not
    - same code with fixed bug
    - description of how to fixe the bug
- repeat 4 times to account for the heuristic nature of chatGPT
  - success if at least one request fixes the bug

# Evaluation

---

1 Does this program have a bug? How to  
2 fix it?

3

```
4 def bitcount(n):  
5     count = 0  
6     while n:  
7         n ^= n - 1  
8         count += 1  
9     return count
```

---

# Results

- standard APR approaches - 7 / 40 bugs fixed
- CoCoNut - 19 / 40 fixed
- Codex - 21 / 40
- chatGPT - 19 / 40 fixed

# Results

Benchmark problem	ChatGPT	Codex [15]	CoCoNut [14]	Standard APR [13]
bitcount	✗ (0 / 4)	✓	✓	✗
breadth-first-search	✓ (2 / 4)	✗	✓	✗
bucketsort	✓ (4 / 4)	✓	✓	✗
depth-first-search	✗ (0 / 4)	✓	✗	✗
detect-cycle	✗ (0 / 4)	✗	✗	✓
find-first-in-sorted	✓ (2 / 4)	✓	✓	✗
find-in-sorted	✓ (3 / 4)	✗	✗	✗
flatten	✓ (4 / 4)	✓	✓	✗
gcd	✗ (0 / 4)	✓	✗	✗
get-factors	✓ (1 / 4)	✓	✓	✗
hanoi	✗ (0 / 4)	✓	✓	✗
is-valid-parenthesization	✓ (2 / 4)	✓	✗	✗
kheapsort	✗ (0 / 4)	✓	✗	✗
knapsack	✓ (1 / 4)	✓	✓	✓
kth	✗ (0 / 4)	✗	✗	✗
lcs-length	✗ (0 / 4)	✗	✓	✗
levenshtein	✗ (0 / 4)	✗	✗	✓
lis	✗ (0 / 4)	✗	✗	✓
longest-common-subsequence	✗ (0 / 4)	✓	✗	✗
max-sublist-sum	✗ (0 / 4)	✓	✗	✗
mergesort	✓ (1 / 4)	✗	✗	✓

# Results

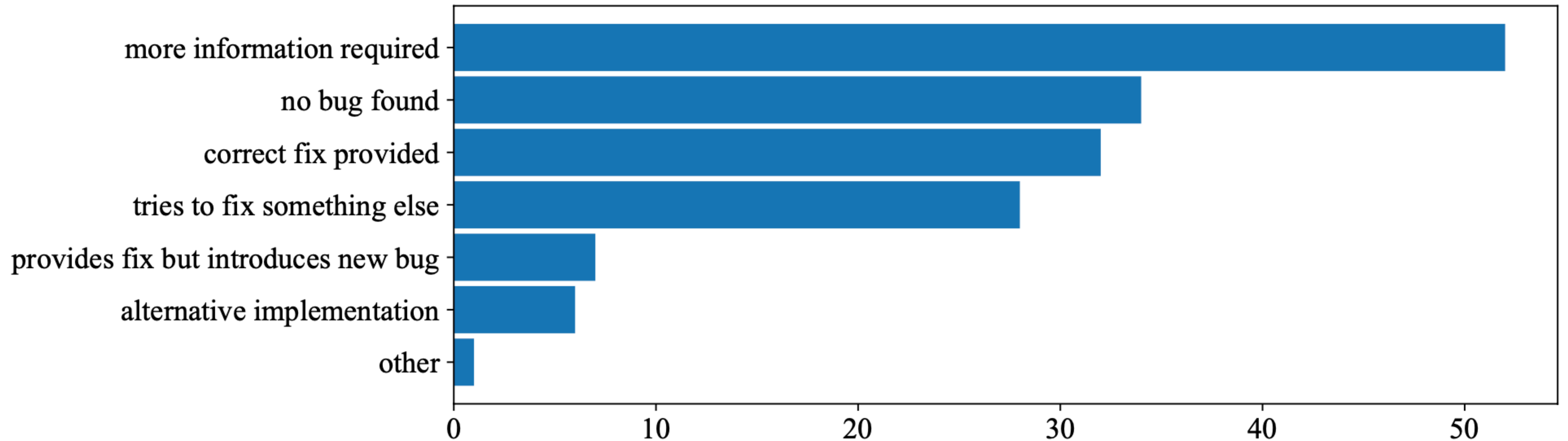
Benchmark problem	ChatGPT	Codex [15]	CoCoNut [14]	Standard APR [13]
bitcount	✗ (0 / 4)	✓	✓	✗
breadth-first-search	✓ (2 / 4)	✗	✓	✗
bucketsort	✓ (4 / 4)	✓	✓	✗
depth-first-search	✗ (0 / 4)	✓	✗	✗
detect-cycle	✗ (0 / 4)	✗	✗	✓
find-first-in-sorted	✓ (2 / 4)	✓	✓	✗
find-in-sorted	✓ (3 / 4)	✗	✗	✗
flatten	✓ (4 / 4)	✓	✓	✗
gcd				✗
get-factors				✗
hanoi				✗
is-valid-palindrome				✗
kheapsort				✗
knapsack	✓ (1 / 4)	✓	✓	✓
kth	✗ (0 / 4)	✗	✗	✗
lcs-length	✗ (0 / 4)	✗	✓	✗
levenshtein	✗ (0 / 4)	✗	✗	✓
lis	✗ (0 / 4)	✗	✗	✓
longest-common-subsequence	✗ (0 / 4)	✓	✗	✗
max-sublist-sum	✗ (0 / 4)	✓	✗	✗
mergesort	✓ (1 / 4)	✗	✗	✓

Execute requests multiple times!

# Answers classification

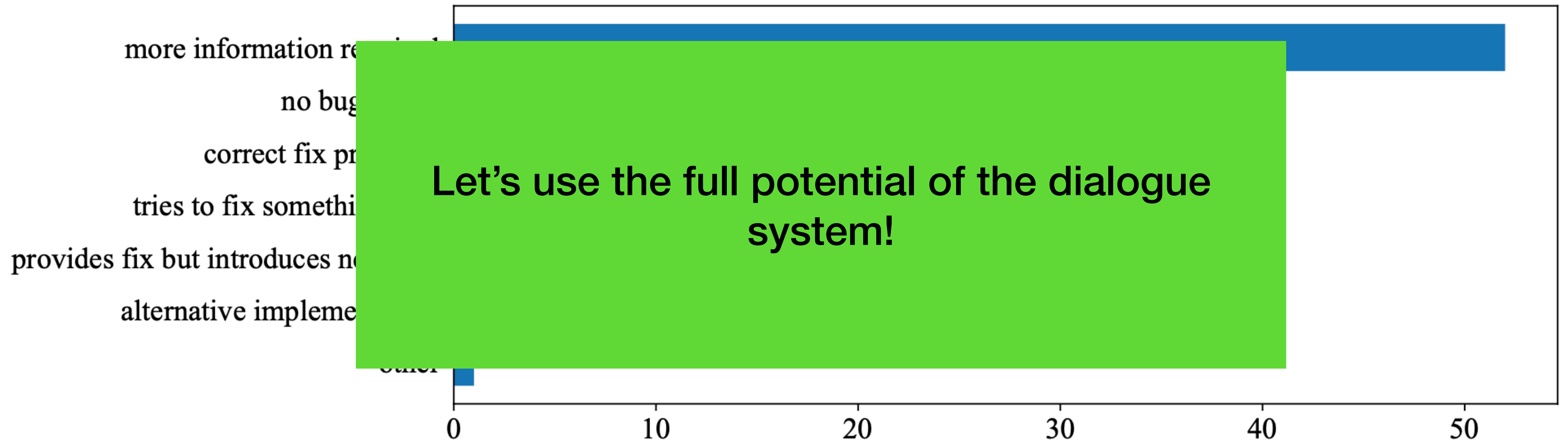
- different classes of chatGPT answers have been identified:
  - **More information required**
  - **No bug found**
  - **Correct fix provided**
  - **Tries to fix something else**
  - **Provides fix but introduces new bug**
  - **Alternative implementation**

# Answers classification





# Answers classification



# Discussion with chatGPT

---

```
1 Request (1):  
2 Does this program have a bug? How to fix it?  
3  
4 def gcd(a, b):  
5     if b == 0:  
6         return a  
7     else:  
8         return gcd(a % b, b)  
9
```

```
10 Response (1):  
11 This is the Euclidean algorithm for finding the greatest common divisor (GCD) of two  
12 integers, a and b. It appears to be implemented correctly. If you are experiencing  
13 issues with this program, it is likely that the problem lies elsewhere in your code.  
14 Without more information about the bug you are experiencing, it is difficult to say  
15 how to fix it.
```

```
16  
17 Request (2):  
18 No, the function is not correct. E.g., for the input a = 4 and b = 5 I get the following  
19 error message: "RecursionError: maximum recursion depth exceeded in comparison".  
20
```

```
21 Response (2):  
22 The issue is that the function calls itself recursively, and in some cases, it never  
23 reaches the base case (b == 0), causing it to recurse indefinitely. The base case of  
24 the gcd algorithm is when the remainder is zero, but in this implementation, it's when  
25 b is zero. For example, when a=4 and b=5, gcd(4, 5) is called, which then calls  
26 gcd(4 % 5, 5) which is gcd(4, 5). This causes an infinite loop.
```

```
27  
28 You can fix this by swapping the parameters in the recursive call.
```

```
29  
30 def gcd(a, b):  
31     if a == 0:  
32         return b  
33     else:  
34         return gcd(b % a, a)  
35
```

```
36 This way, if a is zero, the function returns b (which is the gcd of a and b), otherwise  
37 it calls itself with the new values of b % a and a.
```

---

# Discussion with chatGPT

---

```
1 Request (1):
2 Does this program have a bug? How to fix it?
3
4 def bitcount(n):
5     count = 0
6     while n:
7         n ^= n - 1
8         count += 1
9     return count
10
11 Response (1):
12 I'm unable to tell if the program has a bug without more information on the expected
13 behavior and the input that is causing the issue. [...]
14
15 Request (2):
16 The program should count the number of 1-bits in the binary encoding of n.
17
18 Response (2):
19 The function bitcount(n) appears to be counting the number of 1-bits [...]
20
21 To confirm this, you can test the function with some test cases to check if
22 it produces the expected output:
23
24 test_cases = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
25 expected_output = [0, 1, 1, 2, 1, 2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4]
26
27 Request (3):
28 The function does not work. For input 1 it seems to run into an infinite loop.
29
30 Response (3):
31 You are correct that the function will run into an infinite loop when the input is 1.
32 [...]
33
34 def bitcount(n):
35     if n == 0:
36         return 0
37     count = 0
38     while n:
39         n ^= n - 1
40         count += 1
41     return count
```

# Do hints help?

- a checkmark with an asterisk (\*) defines that a solution was found without a follow-up request being necessary in this run
- after adding hints, chatGPT solved **31 out of 40** problems (77.5%)
  - Codex - 21 / 40
  - CoCoNut - 19 / 40
  - standard APR - 7 / 40

Benchmark problem	ChatGPT
bitcount	✓
depth-first-search	✓*
detect-cycle	✓*
gcd	✓
hanoi	✓
kheapsort	✗
kth	✓
lcs-length	✗
levenshtein	✓
lis	✗
longest-common-subsequence	✗
max-sublist-sum	✓
minimum-spanning-tree	✓
next-permutation	✓
powerset	✓
rpn-eval	✗
shortest-path-lengths	✗
sieve	✓*
to-base	✗
topological-ordering	✗
wrap	✗
<b>Σ (Solved)</b>	<b>9 (12)</b>

# Is it worth it?

- Despite its great performance, the question arises whether the mental cost required to verify ChatGPT answers outweighs the advantages that ChatGPT brings.
- Overall, we find that participants who had access to an AI assistant based on OpenAI's codex-davinci-002 model **wrote significantly less secure code** than those without access. Additionally, participants with access to an AI assistant **were more likely to believe they wrote secure code** than those without access to the AI assistant. Furthermore, we find that participants who trusted the AI less and engaged more with the language and format of their prompts (e.g. re-phrasing, adjusting temperature) provided code with fewer security vulnerabilities. [1]