Proceedings of the 3rd International Conference on Computing Innovation and Applied Physics
DOI: 10.54254/2753-8818/43/20240782

Towards python program repair with generative pre-trained transformer (GPT-3.5)

Zhi Cao[1,3,4], **Qiuyu Ren**[2,5]

[1]Department of Artificial Intelligence, University of Michigan, Ann Arbor, 48109, United States,
[2]Department of Software Engineering, Dalian University of Technology, Dalian, 116620, China,
[3]Corresponding author

[4]zhicao@umich.edu
[5]643064456@mail.dlut.edu.cn

Abstract. ChatGPT has a great potential using a simple prompt design, which means further studies can be done to investigate the effect of different prompt designs. Complex software often contains hidden bugs in its source code. Recent study suggests that OpenAI's ChatGPT can perform numerous operations including code-to-code operations like code completion, translation, repair, and summarization, along with language-to-code operations such as code explanation and search. ChatGPT's dialogue capability can assist in generating more accurate bug fixes. However, it sometimes offers solutions without seeking further information, which can mislead users. To address this issue and enhance user experience, we conducted three design iterations to develop "D-bugger" — a system enabling programmers to fix bugs more effectively. We conducted a survey to gauge the need for refinement, designed a low-fidelity prototype with key features, and then created a high-fidelity prototype evaluated by Python users. Our work aims to enhance the debugging process and user engagement.

Keywords: Program repair, ChatGPT, Misleading rate

1. Introduction

Large generative AI models, such as ChatGPT, GPT-4 or Stable Diffusion, are rapidly transforming the way we communicate, illustrate, and create [1]. Specifically, ChatGPT has a great potential using a simple prompt design, which means further studies can be done to investigate the effect of different prompt designs [2]. Complex software often harbors unnoticed bugs in its source code, which can have extensive consequences if not addressed promptly. Recent study suggests that OpenAI's ChatGPT can perform numerous operations including code-to-code operations like code completion, translation, repair, and summarization, along with language-to-code operations such as code explanation and search [3]. ChatGPT has shown proficiency in bug fixing, with its unique dialogue feature allowing it to enhance the accuracy of its answers. However, there are instances where ChatGPT provides solutions without requesting additional information, leading to potential user confusion. This paper introduces "D-bugger," a system aimed at improving the accuracy and user experience of debugging.

2. Iteration #1: Recognizing the Need to Refine ChatGPT-Based Debugger

To assess the popularity and accuracy of the ChatGPT-based debugger, we conducted a survey. This survey laid the foundation for subsequent design activities. Our investigation covered debugging frequency, accuracy, and user experience with ChatGPT-based debugging.

2.1. Survey Design

We formulated questions about debugging time and experience, ChatGPT-based debugger usage, and suggestions for improvements. The survey revealed that programmers spend a significant portion of their time debugging and that many find ChatGPT's suggestions valuable but sometimes misleading due to the lack of context.

2.2. Participants

We distributed our surveys via social media groups of University of Michigan, Ann Arbor and
Dalian University of Technology. We recruited 64 participants in total, for most who are undergraduates.

2.3. Findings

2.3.1. Whether Refinement Is Needed

Our survey shows that above half of programmers spend more than 50% of programming time on debugging, and they feel tired to face bugs. We can conclude that programmer need a more useful debugger to solve their problem.

2.3.2. How to decrease misleading rate

Our survey suggests that three quarters think ChatGPT often misleads them, making them more annoying, and their advice boils down to giving more information to ChatGPT, *e.g., "Give it the lines I want to modify", "Give it options like 'I need suggestion and fully modified codes'"*.

2.3.3. How to improve user-experience

Our survey tells us that more than three-quarters of programmers they need functions like *"Highlight where is modified", "Comments about how it fixes the bug",* which are included in our application and inspire us to design "Just Copy" Debugger based on ChatGPT.

3. Iteration #2: Low-Fi Prototyping with Participatory Design

Based on the survey's findings that point to system requirements, we described "D-bugger" as a low-fidelity prototype based on ChatGPT that can solve bugs with low misleading rate and high user experience. Then, we conducted a participatory design with three programmers to co-design the initial low-fidelity prototype.

3.1. Participants

We contacted with three programmers who had different educational backgrounds and pain points towards debugging, thus allowing us to calibrate system designs that serve for both high-school students and postgraduates in computer programming, as shown in Table 1.

Table 1. Iteration #2 programmers' pain points in debugging via different educational backgrounds.

Participant	Educational Background	Pain Point
P1	Postgraduate	Annoyed about unexpected whole codes changing
P2	Undergraduate	Unsatisfied with the current visualization function
P3	High-school Student	Cannot determine what information need to be given

3.2. Procedure

To help participants understand the status of "D-bugger", we first used paper/pencil to present a simple interface as shown in Figure 2. Then, we asked participants to brainstorm more interactive features that could help them better exploring and developing "D-bugger".

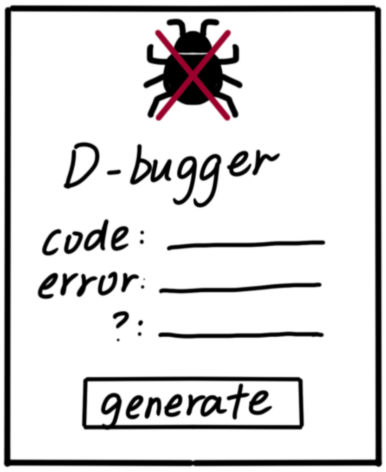


Figure 1. A simple interface of "D-bugger" to help participants understand the current status of "D-bugger".

3.3. Results: Key System Designs

Through iterative collaboration, we identified six key features of "D-bugger" that address programmers' needs:

1.Options to choose debugger behavior.
2.Ability to copy potential error lines for analysis.
3.Inclusion of brief code descriptions to guide the debugger.
4.Visual highlighting of modified lines for easy comparison
5.Detailed explanations of changes made in the code
6.Display of time complexity comparison before and after fixes

#1 Add options to decide what "D-bugger" do (input)

ChatGPT is often considered a textual processor as the uncontrollable output with the only input. However, programmer's requirements are variable, sometimes with important contextual information. According to P1, it is annoying to see his codes changed a lot by ChatGPT. P2 and P3 mentioned that sometimes they tend to learn from specified suggestions, but sometimes they just need the whole modified codes.

Design As shown in Figure 3, we designed a ComboBox to give users the right to choose what function they need and thus what output they want. Previous options include "suggestion and fixed code", "suggestion and just modified error lines", "just suggestion", "just fixed code" and "just modified error lines".

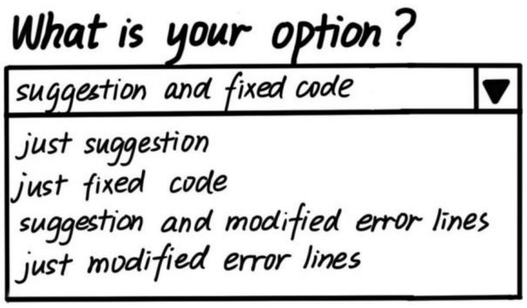

What is your option?
suggestion and fixed code
just suggestion
just fixed code
suggestion and modified error lines
just modified error lines

Figure 2. Demonstration of feature # 1.

3.3.1. #2 Copy possible error lines to guide "D-bugger" (input)

Sometimes, ChatGPT produces a misleading result because of lacking enough information, and it is important to determine what information need to be given. P3 suggested that the most effective way to solve this issue is to show ChatGPT the possible error lines.

Design As shown in Figure 3b, we added a text box to enable the user to copy the error lines. From our test cases in Iteration #3, it is reliable that giving ChatGPT possible error lines can decrease misleading rate of ChatGPT.

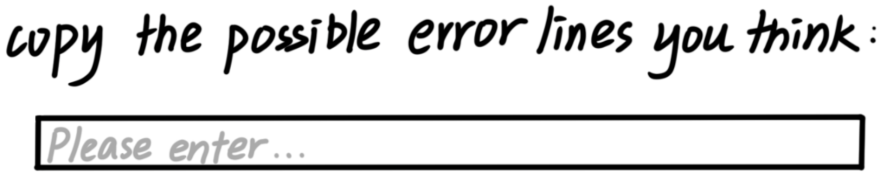


Figure 3. Demonstration of feature # 2.

3.3.2. #3 Give brief description of the code to guide "D-bugger" (input)

To give programmers a clearer idea of why their program is being changed, we include a description of the reason for the change, such as to change the number of loops, or to correct the output. This allows programmers to quickly determine if the modification of the "D-bugger" meets their requirements and is reasonable and helps them decide if they need to give the "D-bugger" more information.

Design As shown in Figure 3c, we added "description part" at the end of the code. To distinguish it from the source code, we deepened the background of "description part" and change the font to red.

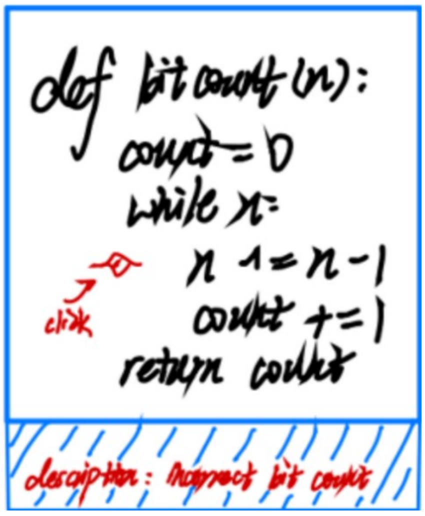
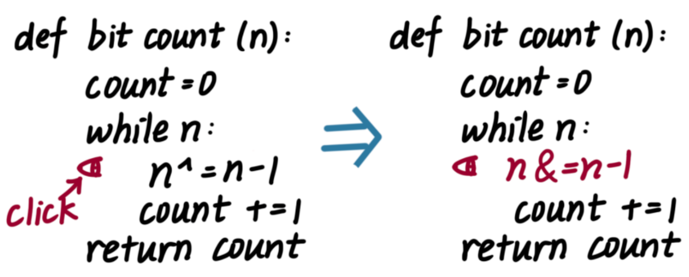


Figure 4. Demonstration of feature # 3.

3.3.3. #4 Click eye icon to show the modified lines (output)

One flaw with the output of ChatGPT is that it's pure text content cannot tell the difference between the output and the original input. Participants tent to learn from the modified lines to prevent the same mistake.

Design As shown in Figure 3d, we made an eye icon which is available to click, and the original input will be shown in the next lines marked as red lines. Thus, it is convenient to recognize the modified lines.



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

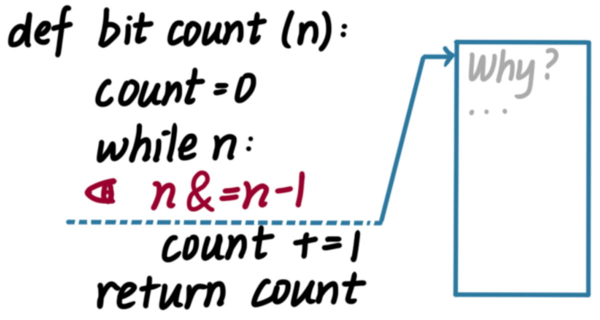
```
def bit count (n):
    count = 0
    while n:
        n&=n-1
        count +=1
    return count
```

Figure 5. Demonstration of feature # 4.

#5 Detailed explanation of where and why the changes are made (output)

ChatGPT do have some explanations about the code generated. However, P2 told us that they are too general to understand the details, especially where and why the changes are made. Thus, a more specified explanation with the place and the reason of the modified code is needed.

Design As shown in Figure 3e, at the right side of the modified code, we put annotations to automatically show the detailed explanation of where and why the changes are made. Usually, the annotation window is always opened, but the user can close it manually.



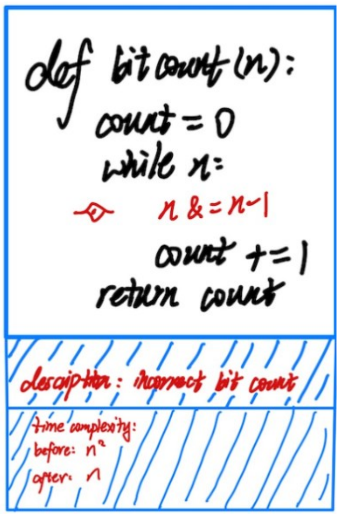
```
def bit count (n):
    count = 0
    while n:
        n&=n-1
        count +=1
    return count
```

Figure 6. Demonstration of feature # 5.

#6 Show comparison of time complexity before and after the change (output)

The time complexity of an algorithm is critical. For software engineers looking for speed of a program, an algorithm becomes meaningless when time complexity exceeds a certain range. Therefore, adding time complexity comparison before and after the fix to the output of "D-bugger" can help programmers quickly determine the rationality of the modified algorithm, which also helps programmers determine whether they need to give "D-bugger" more information.

Design As shown in Figure 3f, at the bottom of the modified code right after the description part, we put the direct comparison of time complexity before and after the fix. Same as feature #3, we deepened the background of "time complexity part" and change the font to red.



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

Figure 7. Demonstration of feature # 6.

4. Iteration #3: Evaluating A High-Fi Prototype

We evaluated the input and output features of "D-bugger" using benchmark evaluation and user interaction. The results demonstrated improved bug fixing performance with the input features, and user feedback highlighted the benefits of the output features.

4.1. Input Features

In this section we present our methodology for assessing program repair performance for different models.

Benchmark Evaluation To evaluate the bugs fixing performance of each method, we use the QuickBugs benchmark set. The QuixBugs benchmark contains 40 programs translated into both Java and Python. Unlike other benchmark sets for automated program repair, QuickBugs contains relatively small problems-each program with a bug on a single line. We take the first 20 programs and for each of them, we take the erroneous Python code and remove all contained comments. At the same time, we randomly select 10 pieces of code from python beginners' projects (we ensure that each piece of code belongs to a function but also ensures randomness, as a control group for the previous example). Figure 4 shows an example random code.

```
def distance(x1,x2,y1,y2):
if x1 == 0 and x2 == 0:
    return float( 'inf ')
else:
    temp1 = x1 - x2 + 1;
    temp2 = y1 - y2;
    return (temp1 * temp1 + temp2 * temp2)** 0.5
```

Code 1. An example for random code.

Comparison Study We run three independent requests to ChatGPT for each problem from the QuickBugs dataset and manually check whether the given answer is correct or not. We standardize our procedure by using the same format for each request. We ask: "Does this program contain a bug? How can we fix it?" followed by an empty line and the buggy code without comments. Figure 5 shows an example request to ChatGPT for the breadth-first search problem.

```
#Does this program contain a bug? How can we fix it?
from collections import deque as Queue

```

```
def breadth_first_search(startnode,goalnode):
    queue = Queue()
    queue.append (startnode)
    nodesseen = set()
    nodesseen.add(startnode)
    while queue:
        node = queue. popleft()
        if node is goalnode:
            return True
        else:
            queve.extend(node for node in node.successors if node not in nodesseen)
            nodesseen.update(node.successors)
    return False
```

Code 2. Request to ChatGPT for the breadth-first search problem.

To compare the results of the original ChatGPT and Chatgpt with additional input features, we build up two new models: ChatGPT with feature #1 (M1) and ChatGPT with feature #1 and feature 2 (M2). For the new models, we add more comments at the end of the buggy code. Figure 6 shows an example request to M1 for the breadth-first search problem and Figure 7 shows an example request to M2 for the same problem.

```
#Does this program contain a bug? How can we fix it?
from collections import deque as Queue

def breadth_first_search(startnode,goalnode):
    queue = Queue()
    queue.append (startnode)
    nodesseen = set()
    nodesseen.add(startnode)
    while queue:
        node = queue. popleft()
        if node is goalnode:
            return True
        else:
            queve.extend(node for node in node.successors if node not in nodesseen)
            nodesseen.update(node.successors)
    return False
#There may be bugs in while loop
```

Code 3. Request to M1 for the breadth-first search problem.

```
#Does this program contain a bug? How can we fix it?
from collections import deque as Queue

def breadth_first_search(startnode,goalnode):
    queue = Queue()
    queue.append (startnode)
    nodesseen = set()
    nodesseen.add(startnode)
    while queue:
        node = queue. popleft()
        if node is goalnode:
            return True
        else:
            queve.extend(node for node in node.successors if node not in nodesseen)
            nodesseen.update(node.successors)
    return False
#There may be bugs in while loop.Also,this code implements breadth-finst search
```

Code 4. Request to M2 for the breadth-first search problem.

RESULTS AND DISCUSSION In the table 8, we present the results of ChatGPT, M1 and M2. For those results, a green 'T' indicates that a correct answer was given, a blue 'M' indicates that the model needs more information, and a red 'W' indicates that a wrong answer was given. We decide that only when at least one of the three requests is 'T' can we prove that the model can solve this program. At the end of the table, we calculate the number of successful solutions and accuracy of each model.

We see that for the first 20 problems, the numbers of successful solutions of M1 and M2 are much higher than the result achieved by ChatGPT. We find bug fixes for 16 benchmark problems for both M1 and M2 but only 8 are reported for ChatGPT. The large gap in performance between original ChatGPT and two revised model hightlight the importance of feature 1, which is shared by M1 and M2 but not ChatGPT.

For the random code, we find a large gap in performance between M2 and the other two models. Also, for original ChatGPT and M1, there are more cases where ChatGPT need more information. This can be explained by the fact that the algorithm we are implementing in the random code we extract from python projects is not very clear, and ChatGPT needs to get more information [4].

Table 2. Results of ChatGPT, M1 and M2. For those results, a green 'T' indicates that a correct answer was given, a blue 'M' indicates that the model needs more information, and a red 'W' indicates that a wrong answer was given.

Problem	ChatGPT	M1	M2	Wrong Answer 1 by Chatgpt	Wrong Answer 2 by Chatgpt	Wrong Answer 3 by Chatgpt
depth-first search	TTW	TTT	TTT	Correct Fix	Correct Fix	Fixed Something Else
bitcount	TWT	TTT	TTT	Correct Fix	Alternative Implementat ion	Correct Fix
breadth-first search	WWW	TTT	TTT	Fixed Something Else	Fixed Something Else	Fixed Something Else
bucketsort	TTT	TTT	TTT	Correct Fix	Correct Fix	Correct Fix
detect-cycle	WWM	MWT	MWT	Fixed Something Else	Fixed Something Else	Correct Fix
find_first_in_sorted	WWW	WWW	WWW	Alternative Implementat ion	Alternative Implementat ion	Alternative Implementat ion
find_in_sorted	WWW	TTT	TTT	Fixed Something Else	Fixed Something Else	Fixed Something Else

Table 2. (continued)

flatten	TTT	TTT	TTT	Correct Fix	Correct Fix	Correct Fix
gcd	WWW	WTT	WTT	Alternative Implementation	No Bugs	Alternative Implementation
get_factors	WWW	WTW	WTW	No Bugs	Alternative Implementation	Fixed Something Else
hanoi	WWW	WTW	WTW	Alternative Implementation	No Bugs	Fixed Something Else
is_valid_parenthesization	TTW	TTT	TTT	Correct Fix	Correct Fix	No Bugs
kheapsort	TTT	TTT	TTT	Correct Fix	Correct Fix	Correct Fix
knapsack	WTT	TTT	TTT	Alternative Implementation	Correct Fix	Correct Fix
kth	WWW	TTT	TTT	Fixed Something Else	Fixed Something Else	Fixed Something Else
lcs_length	WWW	WWW	WWW	Fixed Something Else	Alternative Implementation	Alternative Implementation
levenshtein	WWW	TTT	TTT	Alternative Implementation	Alternative Implementation	Fixed Something Else
lis	WWW	WWW	WWW	Alternative Implementation	Alternative Implementation	Alternative Implementation
longest_common_subsequence	TWW	TWT	TWT	Correct Fix	Alternative Implementation	Fixed Something Else
max_sublist_sum	WWW	WWW	WWW	Fixed Something Else	Alternative Implementation	Alternative Implementation
random code1	WMW	WMW	WTW			
random code2	MMW	MWW	TTT			
random code3	TWM	TWM	TWT			
random code4	MTT	MTT	WTT			
random code5	WWW	WWM	TTT			
random code6	TTM	TTM	TTT			
random code7	WMM	WWM	WWT			
random code8	TMM	TMM	TTW			
random code9	WWM	WWM	TWT			
random code10	WWM	WWM	WWM			

Classification of ChatGPT's Answer While working with original ChatGPT, we noticed different types of responses that ChatGPT gave to our requests. Therefore, we identified the different types of answers from ChatGPT for the benchmark problems from QuixBugs and analyzed their frequency.

- **More information required**: Asks for more information on the program behavior to identify the

bug.
●No bug: Did not find any bugs.
●Fix something else: fix the bugs but also fix something else.
●Alternative implementation: did not fix the bugs and provide different implementation.

Figure 9 shows the number of each type of errors of original ChatGPT answers for the 20 benchmark problems.

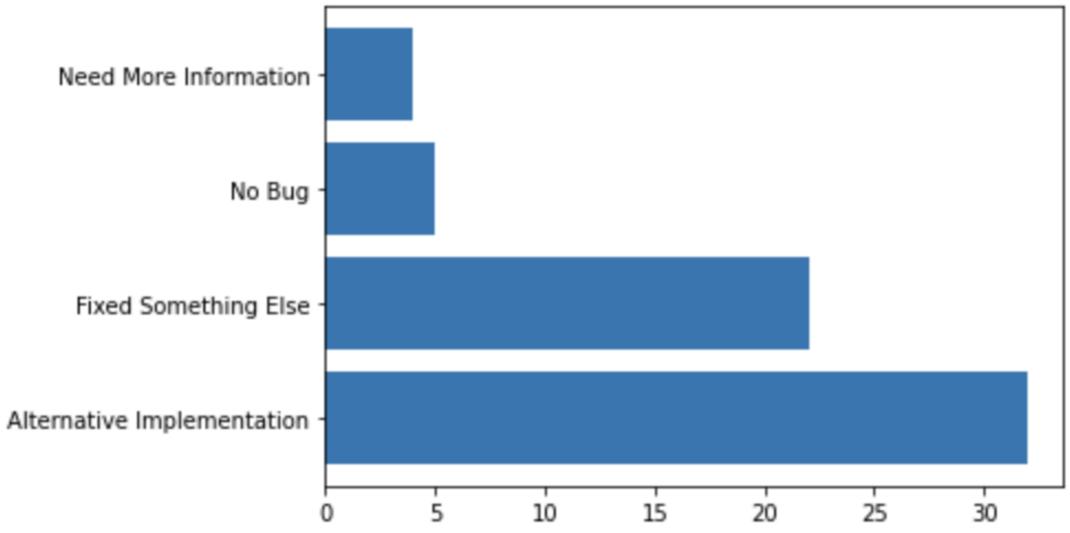


Figure 8. Distribution of each error type for 20 benchmark problem.

4.2. Output Features

We integrated the five key features into a high-fidelity prototype of "D-bugger", which allowed us to further iterate the design by having python programmers interact with the system to explore and understand the output analysis.

4.3. Participants

We invited 10 undergraduate students who are python users. Through more in-depth communication, we finally selected five students to help us test our model.

4.4. Findings

We analyzed each students' response and compare their performance in debugging a given code with original ChatGPT and "D-bugger". Results show that with the help of "D-bugger", students can complete the debugging process faster and more accurately. Meanwhile, five students said that they can filter the data through output analysis, and they can judge whether the output result of "D-bugger" is what they need and whether they need to provide more information [5].

Click eye icon to show the modified lines (#1, #2, #5)

The eye icon helps students locate where "D-bugger" has been modified more quickly, and students don't need to continuously compare the source code with the modified code. They just need to click the eye icon to compare the changes before and after and make a better judgment. "It takes a long time to find a bug fix when the code is very long, and even if you find it, if ChatGPT change more than one place, it's easy to find one place and forget the other" (P1).

Detailed explanation of where and why the changes are made (#1, #3, #4, #5)

"D-bugger" provides detailed explanation of where and why the changes are made, which helps students understand the reasons for the modification of "D-bugger", such as the incorrect parameters in the loop leading to the base case not being considered, the index of the array being misplaced, etc. "D-bugger" enables students to better interpret the fixes. "By understanding the reason for the change, I can see the benefits this fix brings to me, such as helping me increase the time complexity to my desired

range, or correcting the number of my cycles, etc" (P2). Students also save a lot of time filtering useful information: "Sometimes some of the information provided by chatgpt is not useful, such as expressing the modification actions in text, which wastes a lot of time sifting through the useless information" (P4).
Show comparison of time complexity before and after the change (#4, #5)

This feature provides more detailed information, but at the same time, some students do not need the analysis of time complexity. This also depends on the level of the student and what are they using Dbugger for. "As a beginner, I don't care about time complexity, as long as I make sure the output is correct" (P5), while others who use often use ChatGPT to fix complex software bugs mentioned that "D-bugger provides an issue that everyone will be concerned about -- time complexity. The value of time complexity will directly determine whether I want to accept the change" (P4).

5. Conclution

This study introduces "D-bugger," a system designed to enhance the accuracy and user experience of debugging using ChatGPT. Through surveying, prototyping, and user testing, we demonstrated the potential for more accurate and informative debugging solutions. This work contributes to advancing the field of program repair and improving the interaction between programmers and AI systems.

Acknowledgement
Zhi Cao and Qiuyu Ren contributed equally to this work and should be considered co-first authors.

References

[1] Philipp Hacker, Andreas Engel, and Marco Mauer. 2023. Regulating ChatGPT and other Large Generative AI Models. In Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency (FAccT '23). Association for Computing Machinery, New York, NY, USA, 1112–1123. https://doi.org/10.1145/3593013.3594067
[2] Fan Huang, Haewoon Kwak, and Jisun An. 2023. Is ChatGPT better than Human Annotators? Potential and Limitations of ChatGPT in Explaining Implicit Hate Speech. In Companion Proceedings of the ACM Web Conference 2023 (WWW '23 Companion). Association for Computing Machinery, New York, NY, USA, 294–297. https://doi.org/10.1145/3543873.3587368
[3] Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, Barbara J. Ericson, David Weintrop, and Tovi Grossman. 2023. Studying the effect of AI Code Generators on Supporting Novice Learners in Introductory Programming. In Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23). Association for Computing Machinery, New York, NY, USA, Article 455, 1–23. https://doi.org/10.1145/3544548.3580919
[4] Dominik Sobania, Martin Briesch, Carol Hanna, Justyna Petke. 2023. An Analysis of the Automatic Bug Fixing Performance of ChatGPT
[5] Yao Xie, Melody Chen, David Kao, Ge Gao, Xiang Anthony Chen. 2020. CheXplain: Enabling Physicians to Explore and Understand Data-driven, AI-Enabled Medical Imaging Analysis (CHI 2020)