# Studying and Understanding the Effectiveness and Failures of Conversational LLM-Based Repair

Anonymous Author(s)

***Abstract*—Automated program repair is designed to automate the process of bug-fixing. In recent years, thanks to the rapid development of large language models (LLMs), automated repair has achieved remarkable progress. Advanced APR techniques powered by conversational LLMs, most notably ChatGPT, have exhibited impressive repair abilities and gained increasing popularity due to the capabilities of the underlying LLMs in providing repair feedback and performing iterative patch improvement. Despite the superiority, conversational APR techniques can still fail to repair a large number of bugs. For example, a state-of-the-art conversational technique CHATREPAIR does not correctly repair over half of the single-function bugs in the Defects4J dataset. To understand the effectiveness and failures of conversational LLM-based repair and provide possible directions for improvement, we studied the exemplary CHATREPAIR with a focus on comparing the effectiveness of its cloze-style and full-function repair strategies, assessing its key iterative component for patch improvement, and analyzing the repair failures. Our study has led to a series of findings, which we believe provide key implications for future research.**

***Index Terms*—Large language models, conversational APR**

## I. INTRODUCTION

Automated program repair (or APR) aims to alleviate a developer's burden by automatically identifying buggy code and proposing and validating patches for it. Powered by the Large Language Models (LLMs), advanced APR techniques have demonstrated remarkable repair abilities. Among them, conversational APR techniques, which use a conversational LLM (most notably ChatGPT) to understand the failure and generate patches, have gained increasing attention due to the conversational LLM's unique capabilities in providing repair accompanying feedback (which can for example help the developer understand the repair solution) and opportunities for iterative patch improvement (which can enhance patch quality).

CHATREPAIR [1] is a state-of-the-art conversational APR technique that uses ChatGPT as the underlying LLM for bug repair. It assumes a known buggy location and communicates with ChatGPT for cloze-style or full-function patch generation via a prompt that requests generating patched code to replace the buggy location either as a line or a hunk (cloze-style) or an entire method (full-function). At the heart of CHATREPAIR is its key component performing iterative communication with ChatGPT for two purposes: (1) fixing the previous failing patches generated by ChatGPT in an attempt to obtain a plausible patch that can lead to an all-test-passing result and (2) generating alternative plausible patches to improve patch diversity and increase the chance of finding a correct patch.

While conversational APR techniques have exhibited superior repair abilities, they still fail for a large number of real-world bugs, even those requiring only one location to repair. For example, according to a previous evaluation conducted by Xia and Zhang [1], CHATREPAIR fails to repair 175 (over 50%) of 337 Defects4J bugs whose developer patches (serving as the ground-truth) change a line, a hunk (contiguous lines), or a method. There is a lack of research investigating why conversational APR fails to repair so many bugs, even the relatively simple ones. The research is crucial, as it can provide critical guidance on the improvement of conversational LLM-based repair.

To bridge the gap, we took CHATREPAIR as an exemplar conversational APR technique and conducted a study to compare CHATREPAIR's cloze-style and full-function repair strategies, investigate the effectiveness of its key iterative component, and analyze the failures. We implemented the CHATREPAIR tool[1] using ChatGPT (gpt-3.5-turbo-0125) as the underlying conversational LLM and two variants, CHATREPAIR-ONEITER-SH and CHATREPAIR-ONEITER-M, that perform cloze-style and full-function repair with no iterative patch improvement. We applied these tools to a sample of 53 Defects4J bugs whose developer patches change a line, hunk, or method for repair. We slightly adapted the prompts used by the tools to request the underlying ChatGPT to provide not only a patch but also an analysis of the problem and the program expected behavior. We determined the correctness of a repair by comparing the patch against the developer patch provided in the benchmark, checking whether they are semantically equivalent. We also analyzed the repair failures based on the patch and the description of the problem and the program behavior given by ChatGPT.

The key findings of the study are as follows.

- Cloze-style repair is prone to producing programs that do not compile. It is not as effective as the full-function repair, which simply asks ChatGPT to repair the whole method without showing any buggy lines of the method.
- CHATREPAIR's iterative approach for fixing previous failing patches and finding alternative plausible patches does not appear to be helpful. Compared to CHATREPAIR-ONEITER-M, which performs no iteration but independent patch generation using ChatGPT, CHATREPAIR was not better and repaired even four fewer bugs.

[1]By the time we ran our experiments, the CHATREPAIR tool was not available.

- ChatGPT is not very good at repairing bugs whose fix ingredients used for patch construction are not native (e.g., not as operators and language specific data types) and are located outside the buggy method. The success rate of repairing these bugs is 45% and is lower than those for bugs whose fix ingredients are all native (100%) and are only within the scope of the buggy method (60%).
- The main reasons for ChatGPT's failure in proposing a correct patch are that (1) it misunderstands the failure and root cause; (2) it does not know the expected behavior of the repaired program; and (3) it fails to find the key fix ingredient for patch generation.

These findings suggest that method-level fault localization is better suited for conversational LLM-based repair than those targeting smaller code granularities such as statements; that iterative communication with ChatGPT does not fulfill its potential in improving the patch quality; and that future research on ChatGPT-based repair should focus on helping ChatGPT understand the problem, infer the expected behavior, and identify relevant fix ingredients.

Our preliminary observation shows that code context of the buggy method including for example the definitions of the method calls and global variables, code summary in natural language (e.g., the Javadoc comments), and dynamic program states including variables and values recorded while testing can help ChatGPT achieve accurate and deep understanding of the root cause of the problem. We are currently doing more extensive experiments to confirm this observation and also exploring ways to enhance ChatGPT's inference on expected behavior and identifying fix ingredients.

## II. Research Questions

We seek to answer the following five research questions:

1) **RQ1:** How effective are ChatRepair-OneIter-SH and ChatRepair-OneIter-M?
2) **RQ2:** How effective is ChatRepair's iterative patch improvement?
3) **RQ3:** Where are the fix ingredients of the failed repairs from?
4) **RQ4:** What is the connection between ChatGPT's analysis and the patch code?
5) **RQ5:** What are the key factors leading to ChatGPT's repair failures?

For RQ1, we compare the effectiveness of cloze-style and full-function repair strategies. For RQ2, we evaluate the performance of ChatRepair's core iterative patch improvement component. For RQ3, we analyze the locations of the fix ingredients used for patch construction across different bugs and calculate the success rates for repairing them. For RQ4, we classify ChatGPT's analyses into fully correct, partially correct, and incorrect, and then statistically evaluate the repair outcomes for each category. This helps us understand ChatGPT's responses and uncover the relationship between its analysis and the generated patch code. Finally, for RQ5, we identify and summarize the key factors responsible for ChatGPT's repair failures based on the issues explored in the previous research questions. These findings can offer valuable insights for addressing these factors to enhance ChatGPT's repair performance.

## III. Experiment Setup and Method

We chose as benchmark the widely used Defects4J dataset. Because our study involves manual analysis of the repair failures, we used a sample of the bugs. ChatRepair currently only supports repairing single-function bugs. We randomly selected from each of the six projects (Lang, Chart, Closure, Mockito, Math, and Time) 5 single-hunk (including single-line) bugs and at most 5 multi-hunk bugs that have two or three hunks to repair (depending on the number of such bugs in the project). In this way, we obtained 30 single-hunk and 23 multi-hunk bugs. Following previous evaluation of APR techniques [1], [2], we determined the correctness of each patch manually by comparing the patch against the developer patch and checking whether they are semantically equivalent.

### A. RQ1: How effective are ChatRepair-OneIter-SH and ChatRepair-OneIter-M?

We selected 30 single-hunk bugs and ran both tools three times to repair each bug. This process yielded a total of 180 repair results, with 90 results generated by each tool.

TABLE I
The comparison of repair results between ChatRepair-OneIter-SH and ChatRepair-OneIter-M. RT: Repetition Times; CE: Percentage of Compilation Errors; CP: Percentage of Correct Patches.

| Method | RT | CE | CP |
|---|---|---|---|
| ChatRepair-OneIter-SH | 3 | 58.9% | 6.7% |
| ChatRepair-OneIter-M | 3 | 11.1% | 23.3% |

Our result presented in Table III-A shows that 58.9% of the repair results given by ChatRepair-OneIter-SH have compilation errors. Compared to ChatRepair-OneIter-SH, ChatRepair-OneIter-M produced repair results containing much fewer (only 11.1%) compilation errors. ChatRepair-OneIter-M's correct repair rate is 23.3% whereas ChatRepair-OneIter-SH's rate is only 6.7%. This result shows that ChatRepair-OneIter-M is significantly more effective than ChatRepair-OneIter-SH for bug repair.

We identified three types of compilation errors for ChatRepair-OneIter-SH's repair: (1) the patch contains redundant context for the target location; (2) the patch is not made at the target location; and (3) the patch introduces undefined items such as variables. The first two types of compilation errors are dominant, and they accounted for 92.5% of the compilation errors. For ChatRepair-OneIter-M, compilation errors arose for two reasons: the introduction of undefined items and incomplete code generation.

***Finding 1:* ChatRepair-OneIter-SH *generates a substantial number (58.9%) of invalid repairs with compilation errors.***

CHATREPAIR-ONEITER-SH generated correct patches in 6 cases for 5 bugs, and CHATREPAIR-ONEITER-M found correct patches in 21 cases for 9 bugs. Lang-24 and Lang-51 are two bugs that were only repaired by CHATREPAIR-ONEITER-SH but not by CHATREPAIR-ONEITER-M. The remaining 3 bugs correctly repaired by CHATREPAIR-ONEITER-SH were also repaired by the CHATREPAIR-ONEITER-M. 16 of the 21 correct patches made by CHATREPAIR-ONEITER-M are significantly different from the developer patches in terms of the syntax. In contrast, only 1 of the 6 correct patches made by CHATREPAIR-ONEITER-SH are syntactically different from the developer patch.

***Finding 2: The full-function repair strategy can generate a diverse range of patches, increasing the likelihood of finding a correct patch. It however is not very effective at repairing long methods.***

### *B. RQ2: How effective is the iterative patch improvement component of* CHATREPAIR*?*

To evaluate the effectiveness of the iterative patch improvement component of CHATREPAIR, we set the maximum number of attempts in CHATREPAIR to 24 (Xia et al. reported an average of 21.86 attempts to generate plausible patches with CHATREPAIR). We also set the number of repetitions for CHATREPAIR-ONEITER-M to 24. This setup ensures that both CHATREPAIR and CHATREPAIR-ONEITER-M call ChatGPT API 24 times per bug, enabling a fair comparison of the iterative patch improvement component versus independent repeated prompting in terms of the number of bugs repaired and correct patches generated.

The results indicate that the CHATREPAIR-ONEITER-M method repaired 23 bugs, whereas the CHATREPAIR method repaired only 18. Additionally, during the plausible patch generation, CHATREPAIR produced duplicate patches that constituted 65% of the total patches.

***Finding 3: The iterative patch improvement component of* CHATREPAIR *demonstrates no significant advantage over independent repeated prompting. Moreover, the current plausible patch generation process in* CHATREPAIR *produces a substantial proportion of duplicate patches (65%), which limits its overall effectiveness.***

### *C. RQ3: Where are the fix ingredients of the failed repairs from?*

To classify the fix ingredients, we adopted the concept of donor code: the code fragment (e.g., operator, identifier or expression) that, combined with a code change action, forms the replacemen code at the buggy code location [3]. Yang et al. categorized donor code into seven distinct categories based on its source. The *intrinsic* category refers to donor code defined as native tokens specified in the program language including operators and keywords such as basic data types and control structures. The *local method* category encompasses donor code retrieved from the method where the bug was identified. Since CHATREPAIR operates as a repair method at the function level, we collectively group the remaining donor code categories under the label "others". This is because donor code from these categories does not appear in the provided prompts. We used scripts from Yang et al.'s code repository[2] to analyze and determine the donor code category for each bug.

Across all repair experiments, a total of 28 bugs were successfully repaired. The results indicate that all five bugs requiring donor code at the *intrinsic* level were successfully fixed, while bugs with donor code at the *local method* level achieved a 60% repair rate. For the remaining 38 bugs, which required donor code beyond the *local method* level, the repair rate dropped to 45%. Although a correct patch does not need to exactly match the developer patch, the donor code level provides valuable insights into the complexity of the code elements required for bug repair. For example, when addressing bugs with donor code at the *intrinsic* or *local method* levels, ChatGPT only needs to generate built-in keywords, standard library functions, or components defined within the buggy function, which avoids the need to incorporate external project dependencies, making these bugs comparatively easier to repair.

***Finding 4: The underlying ChatGPT struggles with repairing bugs whose fix ingredients include code located outside the buggy method. The success rate for such bugs is 45%, significantly lower than the success rates for bugs with language-level fix ingredients (100%) or fix ingredients within the buggy method (60%).***

### *D. RQ4: What is the connection between ChatGPT's analysis and the patch code?*

The patch generation instructions provided at the end of the prompt explicitly requested ChatGPT to provide an analysis of the issue and the expected correct repair behavior, following a standardized format illustrated in an example. We collected and manually analyzed 249 repair results of CHATREPAIR-ONEITER-M and CHATREPAIR-ONEITER-SH. To classify the results based on the analysis provided by ChatGPT, we used the following questions to establish three evaluation criteria.

1) Does the response identify the erroneous code or code lines?
2) Does the response explain the reason for the error in the code line?
3) Does the response clarify the logic behind the test case failure?

A fully correct analysis must satisfy all three criteria. Partially correct analyses are categorized into three types: (1) partial Explanation of the Reason: The response addresses only one or two of the outlined criteria; (2) Superficial Explanation: the response fails to meet any specific evaluation criterion and provides only a general description of the test case failure; and (3) explanation with Extra Errors: The response includes

[2]https://github.com/DehengYang/repair-ingredients

correct explanations that satisfy the criteria but also contains additional incorrect explanations. If none of the criteria is met, the analysis is classified as incorrect.

TABLE II
THE PROPORTION OF CORRECT REPAIRS UNDER VARYING ANALYSIS CATEGORIES. SH: SINGLE-HUNK AND SINGLE-LINE; MH: MULTI-HUNK; CP: NUMBER OF CORRECT PATCHES; CA: NUMBER OF CORRECT ANALYSES; PCA: NUMBER OF PARTIALLY CORRECT ANALYSES; IA: NUMBER OF INCORRECT ANALYSES.

| Bug Type | Method | CP/CA | CP/PCA | CP/IA |
|---|---|---|---|---|
| SH | CHATREPAIR-ONEITER-SH | 10/21 | 1/8 | 0/61 |
| SH | CHATREPAIR-ONEITER-M | 17/29 | 1/15 | 1/46 |
| MH | CHATREPAIR-ONEITER-M | 8/16 | 1/30 | 0/23 |

Our statistical analysis reveals a strong correlation between ChatGPT's problem analysis and the quality of its generated patches. When ChatGPT correctly understands the problem, the success rate for generating correct patches is 37.8%. In contrast, this rate drops significantly when the analysis is incomplete—falling to less than 5.7% for partially correct analyses and only 0.8% for incorrect analyses.

***Finding 5: ChatGPT can generate correct patches primarily when it has a proper understanding of the problem.***

### *E. RQ5: What are the key factors leading to ChatGPT's repair failures?*

To investigate the key factors contributing to ChatGPT's repair failures, we examined 210 instances of failed repairs from CHATREPAIR-ONEITER-M and CHATREPAIR-ONEITER-SH and identified common issues leading to these failures. The results show that the primary reasons for failure can be categorized into three main types: (1) it fails to understand the root cause of the failure; (2) it does not know the expected behavior of the repaired program; and (3) it fails to find the key fix ingredient for patch generation. In addition to these primary factors, other issues contributing to failed repairs include failure to understand function logic, misinterpretation of project-defined functions, misunderstanding of prompt instructions, generation of incomplete code, and bypassing failed test cases. These secondary factors were less prevalent among the failures. Overall, 66% of the failed repairs were attributed to an inability to understand the problem, 27% to uncertainty about the correct repair behavior, and 15% to the inability to find the key patch components. The remaining six factors collectively accounted for less than 10% of the failures.

***Finding 6: The main factors contributing to ChatGPT's erroneous repairs are (1) it fails to understand the root cause of the failure; (2) it does not know the expected behavior of the repaired program; and (3) it fails to find the key fix ingredient for patch generation.***

## IV. RELATED WORK

Sobania et al. [4] explored ChatGPT's repair performance by directly using a simple prompt without iteration, conducting experiments on QuixBugs, a dataset derived from the Quixey Challenge, which consists of only 40 small bugs. Their prompt included minimal information about the bugs, limiting the study's ability to reflect ChatGPT's repair capabilities on more complex, real-world datasets. Zhang et al. [5] developed an iterative repair approach based on ChatGPT and evaluated its performance on the EVALGPTFIX dataset, which is composed of competition problems. However, their method is specifically tailored to competition tasks, making them unsuitable for evaluating real-world datasets like Defects4J. Xia et al. [1] introduced CHATREPAIR, the state-of-the-art ChatGPT-based conversational repair method. Their work lacks an analysis of the responses generated by ChatGPT, which include both textual explanations and code patches. We applied CHATREPAIR to investigate its repair effectiveness on Defects4J, focusing particularly on the bugs it failed to repair. Through a detailed analysis, we identified and summarized the reasons behind these repair failures from various perspectives.

## V. CONCLUSION AND FUTURE WORK

We conducted a study to understand the effectiveness and failures of conversational LLM-based APR. The study is based on the state-of-the-art technique CHATREPAIR. We investigated CHATREPAIR's cloze-style and full-function repair strategies, assessed its core iterative component for patch improvement, and analyzed its repair failures. Our results suggest that full-function repair is more effective than cloze-style repair for conversational APR; that iterative patch improvement is not helpful; and that problem understanding, expected behavior inference, and fix ingredient identification are three key factors affecting ChatGPT's repair result. We are currently exploring approaches to improving ChatGPT's understanding of problems for better patch generation. Future work includes conducting more extensive experiments that incorporate a broader range of conversational APR techniques and evaluate real-world bugs from diverse benchmarks, particularly those not subject to data leakage.

## REFERENCES


[1] C. S. Xia and L. Zhang, "Keep the conversation going: Fixing 162 out of 337 bugs for $0.42 each using chatgpt," *arXiv preprint arXiv:2304.00385*, 2023.

[2] Xia, Chunqiu Steven and Zhang, Lingming, "Less training, more repairing please: revisiting automated program repair via zero-shot learning," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2022. New York, NY, USA: Association for Computing Machinery, Nov. 2022, pp. 959–971.

[3] D. Yang, K. Liu, D. Kim, A. Koyuncu, K. Kim, H. Tian, Y. Lei, X. Mao, J. Klein, and T. F. Bissyandé, "Where were the repair ingredients for Defects4j bugs?" *Empir Software Eng*, vol. 26, no. 6, p. 122, Sep. 2021.

[4] D. Sobania, M. Briesch, C. Hanna, and J. Petke, "An analysis of the automatic bug fixing performance of chatgpt," *arXiv preprint arXiv:2301.08653*, 2023.

[5] Q. Zhang, T. Zhang, J. Zhai, C. Fang, B. Yu, W. Sun, and Z. Chen, "A Critical Review of Large Language Model on Software Engineering: An Example from ChatGPT and Automated Program Repair," Oct. 2023, arXiv:2310.08879 [cs]. [Online]. Available: http://arxiv.org/abs/2310.08879