Evaluating Explanations for Software Patches Generated by Large Language Models

Dominik Sobania[1 [0000−0001−8873−7143]], Alina Geiger[1 [0009−0002−3413−283X]], James Callan[2 [0000−0002−5692−6203]], Alexander Brownlee[3 [0000−0003−2892−5059]], Carol Hanna[2 [0009−0009−7386−1622]], Rebecca Moussa[2 [0000−0001−9123−6008]], Mar Zamorano López[2 [0000−0002−8872−4876]], Justyna Petke[2 [0000−0002−7833−6044]], and Federica Sarro[2 [0000−0002−9146−442X]]

[1] Johannes Gutenberg University Mainz, Germany
{dsobania,geiger}@uni-mainz.de
[2] University College London, United Kingdom
{james.callan.19,carol.hanna.21,r.moussa,maria.lopez.20,
j.petke,f.sarro}@ucl.ac.uk
[3] University of Stirling, United Kingdom
alexander.brownlee@stir.ac.uk

Abstract. Large language models (LLMs) have recently been integrated in a variety of applications including software engineering tasks. In this work, we study the use of LLMs to enhance the explainability of software patches. In particular, we evaluate the performance of GPT 3.5 in explaining patches generated by the search-based automated program repair system ARJA-e for 30 bugs from the popular Defects4J benchmark. We also investigate the performance achieved when explaining the corresponding patches written by software developers. We find that on average 84% of the LLM explanations for machine-generated patches were correct and 54% were complete for the studied categories in at least 1 out of 3 runs. Furthermore, we find that the LLM generates more accurate explanations for machine-generated patches than for human-written ones.

Keywords: Large Language Models · Software Patches · AI Explainability · Program Repair · Genetic Improvement

1 Introduction

Generative AI tools, especially large language models (LLMs), have recently been used for a variety of general application scenarios [5]. Moreover, applications like ChatGPT are known to an even broader public. In addition to these general applications, LLMs are often used for software development tasks. Recent work studies, for example, their usage for automated code generation [2], bug fixing [11] and explaining source code [1].

Genetic improvement (GI) [10] uses search-based strategies to find patches that improve existing software. GI can improve a software's functional (i.e., fix

bugs) or non-functional properties (e.g., runtime). Despite the successful application of GI in various application domains [10], GI has not yet seen a wider uptake in industry. One of the limitations of current tools is the lack of explanations of the generated patches. Thus it is not surprising, that the use of automatic explanations has already been discussed [7] and investigated [6]. However, the quality of the generated patch explanations has not yet been studied in detail.

Therefore, this work presents an empirical study investigating the quality of LLM-generated explanations of the patches obtained when applying GI. For comparison, we examine how well LLMs can explain human-written software patches. Specifically, we use the API of the `gpt-3.5-turbo-16k` LLM model by OpenAI to generate explanations for software patches, automatically generated by one of the most well-known GI tools, namely ARJA-e [12], to fix 30 bugs extracted from the popular Defects4J benchmark [4]. We also consider corresponding human-written patches to fix the same bugs, in order to assess the quality of LLM-generated explanations for patches generated automatically by GI tooling vs. those applied manually by the developers. Each generated explanation was evaluated by three independent reviewers.

We find that, on average across the studied categories, 84% of the explanations for machine-generated patches were correct and 54% were complete in at least 1 out of 3 runs. We observe a low level of complexity in the explanations.

Given the positive results achieved herein, in the future, it would be interesting to extend the current investigation and include other GI tools, LLM models, and benchmarks. We made the source code along with the relevant scripts and all of the LLM's patch explanations available online.[4]

To the best of our knowledge this is the first work investigating the effectiveness of LLMs for generating explanations for software patches. A comprehensive survey on the use of LLMs in Software Engineering can be found in the work by Fan et al. [3].

2 Experimental Design

In this section, we explain in detail how we generated and evaluated the explanations for the considered software patches.

2.1 Generation of Software Patch Explanations

To analyse and compare explanations of an LLM for human-written and machine-generated patches, we considered bugs from three projects: Chart, Lang, and Math, included in the well-known Defects4J benchmark [4] and used the API of the `gpt-3.5-turbo-16k` LLM model by OpenAI for the explanations. In total, we used 30 randomly sampled bugs under the condition that all changes are made in one file and the length of a file has a maximum of 1500 lines after removing comments, since the used LLM has an upper limit for the length of requests.

[4] https://github.com/SOLAR-group/ExplanationsForSoftwarePatches

We took the human-written patches from Defects4J[5] and the machine-generated patches (one patch per bug) from the ARJA-e repository.[6]

```
In the given code, lines starting with an "o" indicate
unchanged lines, lines starting with a "+" indicate added
lines, and lines starting with a "−" indicate removed lines.
Please explain only the modifications made using the
provided template:

Condition: Under what circumstances or conditions was the
change necessary?
Consequence: What are the potential impacts or effects of
this change?
Position: Where in the codebase was the change implemented?
Cause: What was the motivation for this change? Why was the
previous implementation insufficient or problematic?
Change: How was the code or behavior being altered to address
the identified condition or problem?

The code:

- import java.io.Serializable;
o import java.util.Collections;
+ import java.util.List;
```

Fig. 1. The prompt we used for requests to the LLM with 3 example code lines. Line breaks were added and relevant keywords are printed in **bold** font for better display.

The prompt we used to issue a request to the LLM is shown with some example code lines in Fig. 1 and consists of three parts. In the first part, we explain how we represent the changes introduced by a patch. In the second part, we force the LLM to describe the main characteristics of a software patch as identified by Liang et al. [8]. The third part consists of the source code of the whole file where the patch was applied and each line is marked with the symbols indicating the introduced changes.

Overall, we performed 3 runs for each manually and automatically generated patch for each of the 30 bugs resulting in a total of 180 explanations. For the requests to the LLM, we set the temperature $t = 0.8$, as used by Chen et al. [2].

2.2 Evaluation of the Software Patch Explanations

To evaluate the explanations of the considered patches, we carried out manual assessment. Specifically, we provided each of our 9 reviewers with a set of 10

[5] https://github.com/rjust/defects4j
[6] https://github.com/yyxhdy/arja/tree/arja-e

bugs with their corresponding patches. This means that each of these sets was evaluated by 3 independent reviewers. Furthermore, we anonymized whether it was a human-written or machine-generated patch during the evaluation process.

In order to ensure consistent evaluation, we used the following applicable content-based categories for evaluation of explainable AI as recommended by Nauta et al. [9]:

- **Correctness:** Is the explanation accurate with regards to the provided patch?
- **Completeness:** Does the explanation of the patch describe the changes in the patch in full?
- **Complexity:** Does the explanation for the patch have unnecessary complexity?

		At least 1 run		**3 out of 3 runs**	
Patch generated by:		**Machine**	**Human**	**Machine**	**Human**
Correctness	Condition	**24**	21	**12**	7
	Consequence	**25**	22	**11**	7
	Position	29	29	**22**	21
	Cause	**25**	21	**9**	6
	Change	**23**	22	**9**	6
Completeness	Condition	13	13	**3**	2
	Consequence	**16**	14	**5**	4
	Position	16	**19**	7	7
	Cause	**17**	14	**4**	3
	Change	**19**	18	**6**	3
Complexity	Condition	**5**	7	**0**	1
	Consequence	**5**	7	**0**	1
	Position	4	**3**	0	0
	Cause	**5**	9	**0**	1
	Change	**6**	14	**1**	3
Continuity	Condition	-	-	10	10
	Consequence	-	-	**11**	6
	Position	-	-	**17**	12
	Cause	-	-	9	9
	Change	-	-	**11**	8

Table 1. Results of the manual validation of the LLM-generated explanations for the machine-generated and human-written patches for 30 bugs. At least 1 run refers to the case that in at least 1 out of 3 runs, the reviewers evaluated the categories positively. 3 out of 3 refers to the case where the evaluation was positive for all runs. For Complexity results are reported for a negative evaluation. Best results are marked in **bold** font.

– **Continuity:** Does the model give similar explanations for the same patch over all 3 runs?

For each considered patch and each category (see Fig. 1), reviewers checked whether the explanations given by the LLM are correct. After that, we used a majority vote in order to get the final result. Furthermore, we calculated the inter-rate agreement.

3 Results

Table 1 shows the results of our evaluation for both the machine-generated and the human-written patches. Since we performed 3 runs per patch (API calls to retrieve a patch explanation), we report the sum of correct patch explanations for the case where only 1 run per patch was correct and for the case where all 3 runs provided a correct patch explanation. Best results are printed in **bold** font. Note that for the results in the Complexity category lower values are better.

We see that in most cases, best results are achieved for the machine-generated patches. On average across all categories, explanations for machine-generated patches are correct in 84% of the cases and complete in 54% of the cases in at least 1 out of 3 runs. For human patches, explanations were on average 76.67% correct and 52% complete (in 1 out of 3 runs). It is thus noticeable that the results for Correctness are quite high, but for Completeness we observe significantly lower results. Although descriptions are often formally correct, important details are sometimes missing. On a positive note, the complexity of the patch explanations is generally very low.

Moreover, we calculated the inter-rate agreement between the reviewers by analysing how often all three reviewers agreed in their ratings of the explanations. We found that the inter-rate agreement between the reviewers was 84.27%.

4 Threats to Validity

It is worth mentioning, that we collected the LLM explanations via the API for the `gpt-3.5-turbo-16k` model. Since this is a proprietary model offered by OpenAI, future responses may differ from those in our experiments.

Additionally, the evaluation of the given explanations was done manually and therefore reflects the subjective opinion of the reviewers. To counteract this, each patch was evaluated by three reviewers and the final result was determined by a majority vote. Further, we reported the inter-rate agreement.

Moreover, the patches examined were all correct with respect to the existing software tests, but this does not necessarily mean that they are correct in general. However, this does not pose a threat to our evaluation, as we have investigated how well LLMs can explain the changes made.

5 Conclusions and Future Work

In this paper, we evaluated the quality of explanations generated by an LLM for software patches. Furthermore, we studied whether the quality of LLM explanations differs for machine-generated and human-written patches. We found that, on average across the studied categories, 84% of the LLM explanations for machine-generated patches were correct and 54% were complete in at least 1 out of 3 runs. Our analysis shows that the explanations for machine-generated patches were overall better than the explanations for human-written patches.

In future work, we intend to expand our study to include more benchmark problems and patches generated by other GI tools, in addition to analysing LLM-generated explanations for patches improving non-functional software properties.

6 Acknowledgements

This work was supported by the UKRI EPSRC grant no. EP/P023991/1 and the ERC advanced fellowship grant no. 741278.

References

1. Chen, E., Huang, R., Chen, H.S., Tseng, Y.H., Li, L.Y.: GPTutor: a ChatGPT-powered programming tool for code explanation. arXiv preprint arXiv:2305.01863 (2023)
2. Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H.P.d.O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al.: Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374 (2021)
3. Fan, A., Gokkaya, B., Harman, M., Lyubarskiy, M., Sengupta, S., Yoo, S., Zhang, J.M.: Large language models for software engineering: Survey and open problems (2023)
4. Just, R., Jalali, D., Ernst, M.D.: Defects4J: A database of existing faults to enable controlled testing studies for Java programs. In: Proceedings of the 2014 International Symposium on Software Testing and Analysis. p. 437–440. ISSTA 2014, Association for Computing Machinery, New York, NY, USA (2014)
5. Kaddour, J., Harris, J., Mozes, M., Bradley, H., Raileanu, R., McHardy, R.: Challenges and applications of large language models. arXiv preprint arXiv:2307.10169 (2023)
6. Kang, S., Chen, B., Yoo, S., Lou, J.G.: Explainable automated debugging via large language model-driven scientific debugging. arXiv preprint arXiv:2304.02195 (2023)
7. Krauss, O.: Exploring the use of natural language processing techniques for enhancing genetic improvement. In: 2023 IEEE/ACM International Workshop on Genetic Improvement (GI). pp. 21–22. IEEE (2023)
8. Liang, J., Hou, Y., Zhou, S., Chen, J., Xiong, Y., Huang, G.: How to explain a patch: An empirical study of patch explanations in open source projects. In: 2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE). pp. 58–69. IEEE (2019)

9. Nauta, M., Trienes, J., Pathak, S., Nguyen, E., Peters, M., Schmitt, Y., Schlötterer, J., van Keulen, M., Seifert, C.: From anecdotal evidence to quantitative evaluation methods: A systematic review on evaluating explainable AI. ACM Comput. Surv. **55**(13s) (jul 2023)
10. Petke, J., Haraldsson, S.O., Harman, M., Langdon, W.B., White, D.R., Woodward, J.R.: Genetic improvement of software: a comprehensive survey. IEEE Transactions on Evolutionary Computation **22**(3), 415–432 (2017)
11. Sobania, D., Briesch, M., Hanna, C., Petke, J.: An analysis of the automatic bug fixing performance of ChatGPT. In: 2023 IEEE/ACM International Workshop on Automated Program Repair (APR). pp. 23–30. IEEE Computer Society (2023)
12. Yuan, Y., Banzhaf, W.: Toward better evolutionary program repair: An integrated approach. ACM Transactions on Software Engineering and Methodology (TOSEM) **29**(1), 1–53 (2020)