PatchTrack: Analyzing ChatGPT's Impact on Software Patch Decision-Making in Pull Requests

Daniel Ogenrwot
University of Nevada, Las Vegas
Las Vegas, Nevada, USA
ogenrwot@unlv.nevada.edu

John Businge
University of Nevada, Las Vegas
Las Vegas, Nevada, USA
john.businge@unlv.edu

ACM Reference Format:
Daniel Ogenrwot and John Businge. 2024. PatchTrack: Analyzing ChatGPT's Impact on Software Patch Decision-Making in Pull Requests. In *Proceedings of 39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24 - To appear).* ACM, New York, NY, USA, 2 pages. https://doi.org/XXXXXXX.XXXXXXX

Paper accepted for a poster publication in the proceedings of:
39th IEEE/ACM Int. Conf. on Automated Software Engineering

The present document is the preliminary version of the work prior to peer-review. The final version can be found on the publisher website.

1 INTRODUCTION

In recent years, the integration of AI tools such as ChatGPT into software development has grown significantly, reflecting broader trends in AI-assisted workflows [8]. These tools have great potential to improve decision making related to software patches in pull requests (PR), which are vital components of collaborative software development. Specifically, developers are using features such as link sharing in ChatGPT to enhance collaborative practices, streamline code reviews, and make more informed patch integration decisions.

At the same time, research on fork-based software families has highlighted persistent challenges associated with collaboration, reuse, and code integration across different variants [1, 2, 7]. Managing contributions in distributed software ecosystems, where coordination and reuse are critical to maintaining divergent forks, presents considerable complexity. Recognizing these challenges sets the stage for our investigation into how conversational AI tools like ChatGPT can enhance these collaborative efforts.

To address this gap, our study evaluates ChatGPT's role in improving the efficiency and effectiveness of patch decisions in merged PRs. By analyzing real-world uses of ChatGPT, including how conversations are shared and referenced in collaborative coding scenarios, we aim to provide a comprehensive picture of AI's impact on software development practices. Ultimately, we demonstrate how the integration of ChatGPT into the PR process can improve productivity and decision-making in software teams.

2 STUDY DESIGN

This section presents the research method from data collection, tool design, and quantitative and qualitative analysis. We set the following research questions:

- **RQ1:** What is the distribution of merged pull requests with patches applied (`PA`), not applied (`PN`), or not suggested (`NE`) by ChatGPT?
- **RQ2:** Why are patches suggested by ChatGPT in conversations not integrated (`PN`) into the pull request?

2.1 Data Collection

Step 1: *Identification of merged pull requests containing ChatGPT conversation shared links:* We used DevGPT dataset [9] mined between July 27, 2023, and October 12, 2023. Our study focuses only on the merged pull requests. To ensure a comprehensive and up-to-date analysis, we extended the dataset to accommodate data from October 13, 2023, to February 18, 2024. After filtering out 'toy' projects and inactive links, we remained with 183 PRs from 151 unique repositories.
Step 2: *Extracting pull requests and ChatGPT patches.* Due to OpenAI's updated terms of service, manual extraction of patches from ChatGPT conversations was necessary. Shared links from Step 1 were converted into HTML files, allowing for local scraping without HTTP requests. This process yielded two sets of patches for each pull request: one from ChatGPT and one from the PR, which will be used in the next patch classification step.
Step 3: *Patch classification.* `PatchTrack`, adapted from `PaReco` [6], is used for patch classification by comparing code snippets from ChatGPT-developer conversations with PR patches. Unlike `PaReco`, which handles mainline-variant relationships, `PatchTrack` focuses solely on identifying integrated code snippets. It normalizes code for 34 supported file types and employs `n-gram` tokenization (n=1) to detect subtle code variations, ensuring high precision. Patches are classified at the hunk level into categories: `PA`, `PN`, `NE`, `CC`, or `EE`, depending on the match with ChatGPT-suggested code.
Step 4 & 5: *Pull request classification:* PRs are classified based on the results from Step 3. If a PR contains any `PA` patches, it is classified as `PA`. If no `PA` patches are found but `PN` patches exist, the PR is classified as `PN`. If neither `PA` nor `PN` patches are present, the PR is classified as `NE`. The Jaccard containment metric was used to calculate the percentage of ChatGPT's code integrated into PRs by comparing the token streams from ChatGPT and GitHub patches.
Step 6: *Qualitative analysis:* To address RQ2, we employed a rigorous qualitative analysis methodology. This involved summarizing ChatGPT-developer conversations and GitHub PRs using the Framework Method [5], followed by thematic extraction using card sorting technique. This process was carried out collaboratively by 2 Ph.D. students and 1 Professor from the same lab.

3 RESULTS & DISCUSSION - RQ1

What is the distribution of merged pull requests with patches applied, patches not applied, and no patches suggested by ChatGPT?

In RQ1, we focus on quantifying the distribution of PRs classified as `PA`, `PN` and `NE` from 183 cases in our dataset using `PatchTrack`.

Figure 1 illustrates the distribution of PRs categorized as either PA, PN, NE, CC, or EE. Notably, 70/183 of the PRs are classified under the NE category, indicating that a significant portion of the conversations between developers and ChatGPT does not involve code snippet suggestions. The classifications for PA and PN are 58/183 and 55/183, respectively. Notably, no patches were classified as CC or EE.

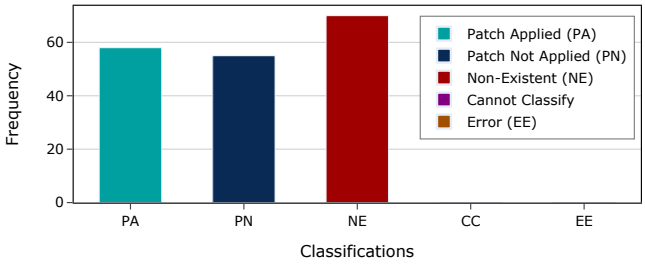

Figure 1: Distribution of patch classification results.

To evaluate the performance of PatchTrack, we employed the metrics of accuracy, precision, recall, and F1-score. The performance of the tool is notable for identifying the NE class, without errors, and demonstrates reasonable precision in classifying the PN (93.4%) and PA (93.4%) classes. The PA class shows a lower precision (73.3%), likely due to its sensitivity to single-line code snippets. This sensitivity, stemming from the use of n-gram tokenization with $n = 1$, may cause false positives. Despite these challenges, the tool performs well and is comparable to related tools, PaReco and ReDebug. Recall that in Section 2–Step 5, we used Jaccard's containment ratio to calculate the percentage of patches applied into the pull requests. The findings indicate that typically, a quarter (25%) of the suggested code from ChatGPT is incorporated into the combined PRs.

Implication

The findings from RQ1 reveal that developers generally trust AI-generated patches, with 58 PA cases showing ChatGPT's effectiveness in straightforward tasks. Typically, 25% of ChatGPT code is integrated into pull requests, with some developers fully adopting AI suggestions. The 55 PN cases suggest a need to explore why AI suggestions are not always integrated, while the 70 NE cases highlight AI's broader roles in development. The tool effectively identifies NE and PN cases but needs refinement in PA precision.

4 RESULTS & DISCUSSION - RQ2

Recall that to address RQ2, in Section 3.2–Step 6 we employed a rigorous qualitative analysis methodology that involves summarizing ChatGPT developer conversations and GitHub PRs, followed by thematic extraction using a card sorting technique.

RQ2: Why are the patches suggested by ChatGPT in conversations not integrated (PN) into the pull request?

In our analysis, we reviewed 53 out of the 55 PN instances classified by PatchTrack, excluding 2 instances identified as false positives. We identified six reasons why patches suggested by ChatGPT were not integrated: adaptation to project needs (33.9%), methodological guidance (18.7%), specific functionality enhancements (15.1%), technical limitations (13.2%), clarification & correction (11.3%), and documentation improvements (7.5%). These findings show that developers often modify ChatGPT's suggestions to better suit their projects, seeking insights, or enhancing internal processes.

Implication

The findings indicate that developers often customize ChatGPT's suggestions to better fit their specific project needs, seek methodological insights, or improve documentation and internal processes. For practitioners, this emphasizes the value of AI tools as supplementary tools that provide conceptual guidance, not standalone solutions. Developers could use AI to refine their practices and align with project-specific requirements, highlighting the need for AI systems that understand project contexts and support broader development activities beyond just code generation.

5 CONCLUSION

This study explored ChatGPT's role in improving patch decision-making within pull requests (PRs). By integrating AI tools like ChatGPT into development workflows, developers can streamline coding practices, documentation, and problem-solving. Additionally, the findings have implications for managing variant software families, where collaboration and code reuse are critical [3, 4]. Future research should focus on optimizing AI for context-specific tasks, such as handling divergent forks, and investigating cases where ChatGPT does not suggest patches (NE).

REFERENCES

[1] John Businge, Mehrdad Abdi, and Serge Demeyer. 2023. *Analyzing Variant Forks of Software Repositories from Social Coding Platforms.* Springer International Publishing, 131–152.

[2] John Businge, Alexandre Decan, Ahmed Zerouali, Tom Mens, Serge Demeyer, and Coen De Roover. 2022. Variant Forks – Motivations and Impediments. In *Proceedings of the 29th edition of the IEEE International Conference on Software Analysis, Evolution and Reengineering.* IEEE Computer Society, 867–877. https://doi.org/10.1109/SANER53432.2022.00105

[3] John Businge, Moses Openja, Sarah Nadi, Engineer Bainomugisha, and Thorsten Berger. 2018. Clone-Based Variability Management in the Android Ecosystem. In *International Conference on Software Maintenance and Evolution.* IEEE, 625–634.

[4] John Businge, Moses Openja, Sarah Nadi, and Thorsten Berger. 2022. Reuse and maintenance practices among divergent forks in three software ecosystems. *Empirical Softw. Engg.* 27, 2 (mar 2022), 47 pages.

[5] Nicola K. Gale, Gemma Heath, Elaine Cameron, Sabina Rashid, and Sabi Redwood. 2013. Using the framework method for the analysis of qualitative data in multi-disciplinary health research. *BMC Medical Research Methodology* 13, 1 (2013), 117. https://doi.org/10.1186/1471-2288-13-117

[6] Poedjadevie Kadjel Ramkisoen, John Businge, Brent van Bladel, Alexandre Decan, Serge Demeyer, Coen De Roover, and Foutse Khomh. 2022. PaReco: patched clones and missed patches among the divergent variants of a software family. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2022).* Association for Computing Machinery, New York, NY, USA, 646–658. https://doi.org/10.1145/3540250.3549112

[7] Henrique Rocha and John Businge. 2022. Blockchain-Oriented Software Variant Forks: A Preliminary Study. In *5th International Workshop on Blockchain Oriented Software Engineering.*

[8] Jaakko Sauvola, Sasu Tarkoma, Mika Klemettinen, Jukka Riekki, and David Doermann. 2024. Future of software development with generative AI. *Automated Software Engineering* 31, 26 (2024). https://doi.org/10.1007/s10515-024-00330-1

[9] Tao Xiao, Christoph Treude, Hideaki Hata, and Kenichi Matsumoto. 2023. *DevGPT: Studying Developer-ChatGPT Conversations.* https://doi.org/10.5281/zenodo.8304091