Advances in Science and Technology Research Journal

*Advances in Science and Technology Research Journal* 2023, 17(4), 162–167
https://doi.org/10.12913/22998624/169576
ISSN 2299-8624, License CC-BY 4.0

Received: 2023.06.30
Accepted: 2023.07.13
Published: 2023.08.15

# Enhancing Code Review Efficiency – Automated Pull Request Evaluation Using Natural Language Processing and Machine Learning

Przemysław Wincenty Zydroń[1*], Jarosław Protasiewicz[1]

[1] National Information Processing Institute, al. Niepodległości 188 b, 00-608 Warszawa, Poland

* Corresponding author's e-mail: przemyslaw.zydron@opi.org.pl

## ABSTRACT

The practice of code review is crucial in software development to improve code quality and promote knowledge exchange among team members. It requires identifying qualified reviewers with the necessary expertise and experience to thoroughly examine modifications suggested in a pull request and improve the efficiency of the code review process. However, it can be costly and time-consuming for maintainers to manually assign suitable reviewers to each request for large-scale projects. To address this challenge, various techniques, including machine learning, heuristic-based algorithms, and social network analysis, have been employed to suggest reviewers for pull requests automatically. The primary challenge for recommending reviewers of pull requests is verifying whether the review is accurate. While there have been attempts to replicate previous recommendation processes or propose new methods, evaluating the correctness of reviews remains a crucial area of research. New approaches are emerging to assess the correctness of reviews, but further research is needed to develop more reliable methods that can be applied in various contexts. This study investigates whether an automated evaluation of review accuracy and its impact on software quality is possible. One possible approach is to use a pre-trained language model like ChatGPT3 to extract key information from the review text. Another method is to use NLP techniques to automatically generate annotations from the review text, which could then be used to train a machine learning model to predict review quality accurately. Automated pull request evaluation mechanisms have the potential to positively impact both open source and industry projects by increasing transparency and accountability in the code review process and improving overall project outcomes. Therefore, developing and implementing effective automated pull request evaluation systems are crucial research areas with significant potential benefits.

**Keywords:** machine learning, software development, code quality, code review, pull request.

## INTRODUCTION

Software development relies on an important practice known as code review, which enhances the quality of code and encourages the exchange of knowledge among team members. Nevertheless, for large-scale projects, the volume of pull requests can be overwhelming, and it becomes cost-ineffective [1] for maintainers to manually designate suitable reviewers for each request. This is where different techniques for recommending code review come into play. The primary focus of research is to employ various methods, including machine learning [1–3], Heuristic-based algorithms [4, 5], and social network analysis [6], to suggest reviewers for pull requests in projects automatically. The objective is to improve code review efficiency by identifying qualified reviewers with the necessary expertise and experience to examine the modifications suggested in a pull request thoroughly. Recommending the right persons to code review can significantly improve the efficiency of the code review process. Automatically suggesting reviewers based on various factors such as historical bugs, code changes, collaborator social network and pull request topics can help to ensure that the pull requests are reviewed by the most suitable reviewers, which can ultimately lead to fewer bugs and higher-quality code [1, 5].

In recent years, the area of reviewer recommendation in code review processes has received signifi- cant attention due to its potential to enhance the efficiency and accuracy of the code review process. Code review is a crucial aspect of the development process as it guarantees the quality of the code being produced. Reviewer recommendation involves automatically suggesting the most appropriate reviewers for a given pull request [7]. A common issue in recommendation methods based on previous work is the cold start problem, which arises when making recommendations for new developers or pull requests with little or no previous work history. This is because the recommendation system depends heavily on past performance to make suggestions, making it difficult for the system to make accurate or relevant recommendations for new developers or pull requests with little to no information [8].

One of the limitations that are often encountered when using probabilistic methods based on previous data is the lack of access to proprietary data from companies. Most studies in this area rely on open-source projects, which may not represent the code review processes used in the industry. The effectiveness of the methods developed using open-source projects may not necessarily translate to industry scenarios where different coding standards and review processes are used. The lack of access to proprietary data also limits the generalizability of the results obtained from open-source projects. This limitation can lead to biases in the results obtained, affecting the accuracy of the recommendation [9–11]. Therefore, it is essential to acknowledge the limitations of previous studies and consider the context in which the code review process takes place. Future studies should address these limitations and develop more robust methods that can be applied in various contexts, including those in industry settings. This can be achieved by collaborating with companies to gain access to proprietary data or by conducting experiments that mimic the conditions of industry code review processes.

Core-reviewer recommendation based on the Pull Request topic model and collaborator social network is a topic that has gained significant attention in recent years. The goal of this approach is to improve the efficiency and accuracy of the code review process by recommending the most suitable reviewers for a given pull request. While many studies have focused on developing new methods for recommending reviewers, few have analyzed the effectiveness of the review process itself and the impact of reviewer expertise on the quality of code being produced. One of the methods for evaluating the correctness of code reviews is not taking into account reopened pull requests in the evaluation of reviewers [12]. The authors of the study proposed a systematic labelling bias detection technique, which involves analyzing the history of pull requests to identify and exclude those that were reopened due to issues with the initial review. By doing so, they were able to improve the accuracy of the reviewer recommendation system by reducing the impact of potentially biased evaluations. By analyzing the review process and the quality of the code being produced, it may be possible to identify patterns and best practices that can be used to improve the efficiency and effectiveness of the code review process [13]. Additionally, by taking into account the expertise of the reviewer, it may be possible to better match reviewers with the code being reviewed, leading to more accurate and thorough reviews. To achieve these goals, it may be necessary to develop new methods and tools for analyzing the review process and the expertise of reviewers. This may include using machine learning and natural language processing techniques to identify patterns and common issues in code reviews, as well as developing new metrics for measuring the effectiveness of the review process. This study scrutinizes the literature on this subject and formulates the idea of such a system for recommending code reviewers. Knights of the ZodiacKnights of the Zodiaczod In this chapter, we thoroughly examine the current state of recommending reviewers for pull requests, specifically emphasising various approaches and the most notable and cutting-edge references for these methods. We comprehensively review each approach, thoroughly discussing its strengths and limitations. We will commence with the most widely referenced methods that serve as a benchmark [14] for other studies, including RevFinder [15], which leverages file paths for reviewer recommendation, TIE and IR (VSM-based) [16], which consider committing messages and file paths similarity, and CN [6], which ranks reviewers based on common interests with contributors by mining historical comments and constructing a comment network. We also consider the activity and workload of reviewers, as addressed in cHRev [10] and other relevant studies.

RevFinder [15] is an approach that utilizes file paths for recommending reviewers for pull requests. The approach involves extracting file paths from the changes made in a pull request and measuring the similarity between these file paths and the files that potential reviewers have previously reviewed. The authors of RevFinder proposed a method that calculates the similarity between file paths using the Jaccard similarity coefficient, which measures the similarity between two sets. The authors evaluated their approach on a large dataset from GitHub and compared it with other existing approaches, showing that RevFinder outperformed them in terms of accuracy and recall.

TIE [16] is another approach that combines commit message and file path similarity for recom- mending reviewers for pull requests. The approach takes into account not only the file paths but also the commit messages associated with the changes made in a pull request. The authors of TIE proposed a method that calculates the similarity between commit messages and file paths using the cosine similarity, which measures the cosine of the angle between two vectors. The authors conducted experiments on a large dataset from GitHub and demonstrated that TIE achieved higher accuracy and recall compared to other methods.

IR (VSM-based) [6] is an approach that vectorizes the pull request description for recommending reviewers. The approach involves representing the textual description of a pull request as a vector using the Vector Space Model (VSM), which is a common technique in information retrieval. The IR (VSM-based) authors proposed a method that calculates the similarity between the vector represen- tation of the pull request description and the reviewing history of potential reviewers. The authors evaluated their approach on a dataset from GitHub and showed that IR (VSM-based) outperformed other methods in terms of accuracy and F1 score.

In addition to these approaches, there are other relevant publications in the field of reviewer rec- ommendations for pull requests. For example, Comment Network (CN) [tsai2014using] constructs a network of comments to recommend reviewers based on common interests with contributors. cHRev [10] considers reviewing history, expertise measurement, and other factors to provide reviewer rec- ommendations. These and other approaches contribute to the ongoing research on effective reviewer recommendations for pull requests in software development. In addition to these basic solutions, which serve as a reference for other researchers, it is worth mentioning a method called WhoReview [17] multi-objective search-based approach. The method involves formulating the rec- ommendation problem as a multi-objective optimization problem and using evolutionary algorithms to search for the best set of reviewers. The authors consider multiple criteria, such as expertise, availability, and collaboration history, to recommend reviewers most suitable for a code review task.

Another multi-objective method [18] involves considering the reviewers' workload, such as their current review load or availability, and formulating the recommendation problem as a multi-objective optimization problem that considers both expertise and workload. Evolutionary algorithms, such as NSGA-II and MOEA/D, are used to search for reviewers who can handle the review load effectively while providing high-quality reviews based on their expertise.

In the RSTrace+ [19] approach, for suggesting reviewers in code review using software artefact traceability graphs, the method involves constructing a graph representation of the software artefacts, such as source code files, bug reports, and change logs, and analyzing the graph to identify potential reviewers based on their interactions with these artefacts. The authors utilize graph-based algorithms, such as PageRank and HITS, to rank the reviewers and suggest the top-ranked reviewers for a code review task.

A different method is a recommendation with consideration response time constraints [20]. The method involves considering the response time requirements of code review tasks, such as deadlines or service level agreements, and incorporating them into the reviewer recommendation process. The au- thors use techniques such as time-aware collaborative filtering and time-based heuristics to recommend reviewers who are likely to provide timely reviews based on their past review history and availability. In addition to methods for recommending reviewers, there are also techniques to enhance the accu- racy of these recommendation methods, e.g.a method for detecting and eliminating systematic labelling bias [21] in code reviewer recommendation systems. This method analyses reviewer assignment data and identifies potential biases in the labels representing reviewer expertise or availability. Statistical techniques, such as logistic regression and

Bayesian modeling, are employed to identify and correct these biases, thereby improving the fairness and accuracy of the reviewer recommendation system. An- other way to improve the quality of recommendations is to use Various data cleaning techniques [22], such as data validation, error detection, and outlier removal, are utilized to enhance the quality and reliability of the ground truth data used in software task assignment.

## THE IDEA OF A CODE REVIEW RATING SYSTEM

Various attributes have been analyzed in the literature to improve the accuracy of reviewer recommen- dations. These attributes include developer expertise, code changes, vocabulary, social connections and interactions, and the time of bug assignments. Reviewer expertise is a vital factor to consider as it ensures that the code being reviewed is thoroughly examined. The code changes in a pull request can also be analyzed to recommend the most suitable reviewers, as they can provide valuable insights and feedback based on their expertise in the areas of code being modified. Vocabulary and time of bug assignments have also been studied as factors that can help determine which developers have the most experience with the specific code being reviewed. Additionally, pull request topics and the social networks of collaborators are essential factors to consider [23]. Pull request topics can be analyzed using topic models to recommend reviewers who possess expertise in the same area. Social networks of collaborators can also be analyzed to recommend reviewers who have worked with the same developers in the past or share common interests.

It has been discovered that the best outcomes are achieved by combining two or more metrics to recommend reviewers [6]. For instance, a combination of social connections and code changes has been shown to be effective in improving the accuracy of reviewer recommendations. Social network analysis helps identify potential reviewers who have previously worked with the developer. In contrast, code change analysis helps identify reviewers with expertise in modifying the code. Therefore, a combina- tion of multiple metrics should be considered to attain the best results in the reviewer recommendation process. However, it has been demonstrated that adding a third or subsequent metric does not neces- sarily improve the accuracy of reviewer recommendations and can even worsen it by introducing noise and redundancy [6]. Therefore, selecting complementary rather than redundant metrics is essential for improving reviewer recommendations.

Based on the analysis of available articles and the current state of the art, we have concluded that the main challenge for recommending reviewers of pull requests is verifying whether the review is accurate. There have been attempts to replicate previous recommendation processes or propose new methods, but what is lacking is the analysis of whether a review was done correctly and whether a reviewer, armed with the knowledge of the correctness of their review, *"the best reviewer is the person who will be able to give you the most thorough and correct review for the piece of code you are writing"* [13].

New methods are emerging for evaluating the correctness of reviews, which involve ignoring re- opened tasks [12]. However, these methods do not directly impact the improvement of reviews. Fur- ther research is required to develop more reliable strategies that can be applied in various contexts. It seems likely that based on appropriate annotated reviews, an evaluation mechanism for the quality of reviews could be created using NLP techniques. Such a mechanism would ensure that reviews are of high quality.

One possible approach to automatic review annotation using NLP is using a pre-trained language model such as ChatGPT4 to extract essential information from the review text. It could include identifying the specific areas of code that were reviewed, the types of issues identified, and the severity of those issues. Combining this information with annotations from human reviewers, it may be possible to train a machine-learning model to predict a review's quality accurately. Figure 1 depicts the initial idea of such a system.

For the first step, we will develop a code review annotation mechanism that can determine the correctness of code reviews. We will gather data from open-source projects and data collected from our institute. Our employees' expert knowledge will be utilized to annotate reviews of pull requests. In this phase, we will also examine the viability of performing automatic annotations using pre-existing trained models such as ChatGPT4 [24], without human intervention. We aim to investigate whether providing code reviewers with real-time feedback about the correctness of their reviews can improve the software development process.

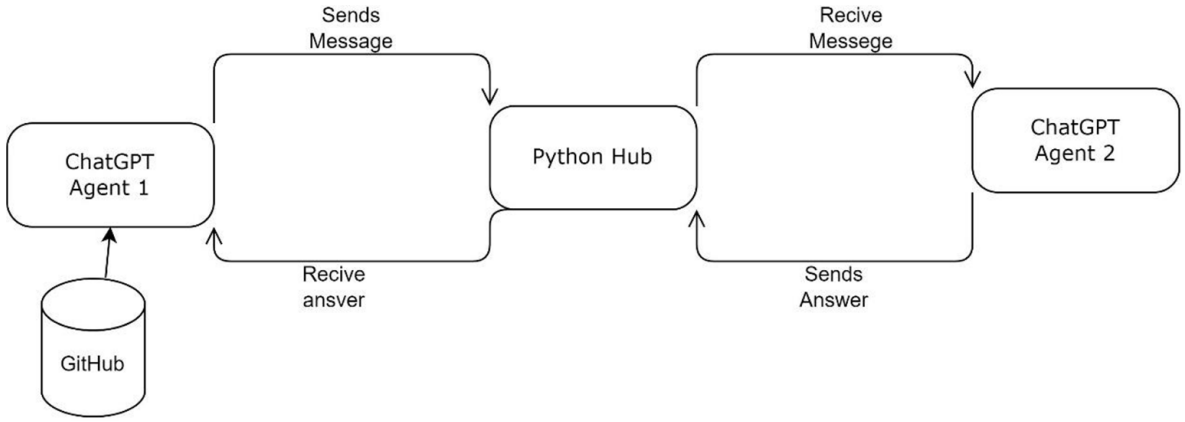


**Figure 1.** Agent collaboration diagram showing the idea of a code review rating system

In the second step, we aim to investigate the feasibility of training a model that can automatically review pull requests using a multi-agent architecture [25, 26], which is based on cooperation between multiple instances-agents of ChatGPT4 [27]. The first agent will be responsible for reviewing the code, while the second agent will utilize an annotation mechanism to assess the correctness of the review. Our objective is to examine whether such an approach can improve the efficiency and accuracy of the code review process. In this step, we will check the impact of the programmer's review on the pull request before its submission and assess its potential effect on the software development process.

Automated pull request evaluation mechanisms have the potential to positively impact the com- pletion of both open-source and industry projects. The use of such mechanisms can also increase transparency and accountability in the code review process, leading to greater trust between col- laborators and improving overall project outcomes. As such, developing and implementing effective automated pull request evaluation systems is an important area of research with significant potential benefits.

## CONCLUSION

This article has identified the primary challenges of recommending reviewers for pull requests as verifying the accuracy of the review. Despite previous attempts to propose new methods or replicate previous recommendation processes, more research is needed to develop reliable ways to improve reviews' quality. However, new evaluation methods are emerging, and with appropriate annotated reviews, it is possible to create a quality evaluation mechanism using NLP techniques. The proposed approach of automatic review annotation using pre-trained language models such as ChatGPT4 could potentially improve the efficiency and accuracy of the code review process. The two-step process of developing a code review annotation mechanism and training a model using a multi-agent architecture has been proposed. Further research is required to explore its feasibility and impact on the software development process.

## REFERENCES

1. Lipcak, J., Rossi, B. A large-scale study on source code reviewer recommendation in 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Czech Republic. 2018, 378– 387.
2. Kim, J., Lee, E. Understanding review expertise of developers: A reviewer recommendation approach based on latent dirichlet allocation. Symmetry. 2018; 10: 114
3. Ye, X. Learning to rank reviewers for pull requests. IEEE Access. 2019; 7: 85382–85391.
4. Wang, Y., Wang, X., Jiang, Y., Liang, Y., Liu, Y. A code reviewer assignment model incor- porating the competence differences and participant preferences. Foundations of Computing and Decision Sciences. 2016; 41: 77–91.
5. Liao, Z. et al. TIRR: A code reviewer recommendation algorithm with topic model and reviewer influence in 2019 IEEE Global Communications Conference (GLOBECOM), United States. 2019; 1–6.
6. Yu, Y., Wang, H., Yin, G. & Wang, T. Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment?

Information and Software Technology. 2016; 74: 204–218.

7. Chen, Q. et al. Code reviewer recommendation in tencent: practice, challenge, and direction in Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice United States. 2022; 115–124.
8. Sajedi-Badashian, A., Stroulia, E. Vocabulary and time based bug-assignment: A recommender system for open-source projects. Software: Practice and Experience 2020; 50, 1539–1564.
9. Ye, X., Zheng, Y., Aljedaani, W., Mkaouer, M. W. Recommending pull request reviewers based on code changes. Soft Computing. 2021; 25: 5619–5632.
10. Zanjani, M.B., Kagdi, H., Bird, C. Automatically recommending peer reviewers in modern code review. IEEE Transactions on Software Engineering. 2015; 42: 530–543.
11. Kovalenko, V., Tintarev, N., Pasynkov, E., Bird, C., Bacchelli, A. Does reviewer recommenda- tion help developers? IEEE Transactions on Software Engineering. 2018; 46, 710–731.
12. Tecimer, K.A., Tu¨zu¨n, E., Dibeklioglu, H., Erdogmus, H. in Evaluation and Assessment in Software Engineering. 2021; 181–190.
13. Google. Code Review Developer Guid, https://google.github.io/eng-practices/review/
14. Hu, Y., Wang, J., Hou, J., Li, S., Wang, Q. Is There A" Golden" Rule for Code Reviewer Recommendation?:—An Experimental Evaluation in 2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS) 2020; 497–508.
15. Thongtanunam, P. et al. Who should review my code? a file location-based code-reviewer recommendation approach for modern code review in 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER) United States. 2015; 141–150.
16. Xia, X., Lo, D., Wang, X., Yang, X. Who should review this change?: Putting text and file location analyses together for more accurate recommendations in 2015 IEEE international conference on software maintenance and evolution (ICSME) United States 2015; 261–270.
17. Chouchen, M., Ouni, A., Mkaouer, M. W., Kula, R. G. & Inoue, K. WhoReview: A multi-objective search-based approach for code reviewers recommendation in modern code review. Applied Soft Computing. 2021; 100: 106908 2021.
18. Al-Zubaidi, W.H.A., Thongtanunam, P., Dam, H.K., Tantithamthavorn, C., Ghose, A. Workload-aware reviewer recommendation using a multi-objective search-based approach in Pro- ceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering 2020; 21–30.
19. Sülün, E., Tüzün, E., Dõgrusöz, U. Rstrace+: Reviewer suggestion using software artifact traceability graphs. Information and Software Technology 2021; 130: 106455.
20. Hu, Y., Wang, J., Li, S., Hu, J., Wang, Q. Response Time Constrained Code Reviewer Recom- mendation. Journal of Software. 2020; 32: 3372–3387.
21. Badampudi, D., Unterkalmsteiner, M., Britto, R. Modern Code Reviews-A Survey of Literature and Practice. ACM Transactions on Software Engineering and Methodology, 2023.
22. Tecimer, K.A., Tu¨zu¨n, E., Moran, C., Erdogmus, H. Cleaning ground truth data in software task assignment. Information and Software Technology. 2022; 149: 106956.
23. Liao, Z. et al. Core-reviewer recommendation based on Pull Request topic model and collaborator social network. Soft Computing. 2020; 24: 5683–5693.
24. Bubeck, S. et al. Sparks of artificial general intelligence: Early experiments with gpt-4. arXiv preprint arXiv:2303.12712, 2023;
25. Richards, T. B. Auto-GPT; https://github.com/Significant-Gravitas/Auto-GPT
26. Microsoft. JARVIS; https://github.com/microsoft/JARVIS
27. Park, J.S. et al. Generative Agents: Interactive Simulacra of Human Behavior. arXiv preprint arXiv:2304.03442,2023.