# Unity Is Strength: Collaborative LLM-Based Agents for Code Reviewer Recommendation

Luqiao Wang
Xidian University
Xi'an, Shaanxi, China
wangluqiao@stu.xidian.edu.cn

Yangtao Zhou
Xidian University
Xi'an, Shaanxi, China
zhou_yt@stu.xidian.edu.cn

Huiying Zhuang
Xidian University
Xi'an, Shaanxi, China
huiying818@stu.xidian.edu.cn

Qingshan Li
Xidian University
Xi'an, Shaanxi, China
qshli@xidian.edu.cn

Di Cui
Xidian University
Xi'an, Shaanxi, China
cuidi@xidian.edu.cn

Yutong Zhao
University of Central Missouri
Warrensburg, USA
yutongzhao@ucmo.edu

Lu Wang
Xidian University
Xi'an, Shaanxi, China
wanglu@xidian.edu.cn

## Abstract

Assigning pull requests to appropriate code reviewers can accelerate the review process and help uncover potential bugs. However, the inherent complexities in pull requests and code reviewers present challenges in making suitable matches between them. Prior studies focus on mining rich semantic information from pull requests or profile information from code reviewers to improve efficiency. These approaches often overlook the intrinsic relationships between pull requests and code reviewers, which can be represented by a combination of multiple factors and strategies, resulting in suboptimal recommendation accuracy.

To address this issue, we propose CoRe, a collaborative agent-based code reviewer recommendation approach that emphasizes flexibility and adaptability. We leverage Large Language Models (LLMs) to precisely capture the rich textual semantics of both pull requests and reviewers. Additionally, we integrate various factors into the recommendation process through the robust planning, collaboration, and decision-making capabilities of multi-agent systems. This integration significantly enhances the performance of LLM-based code reviewer recommendations. We evaluate the effectiveness of our approach on four widely used projects. The results demonstrate that CoRe outperforms state-of-the-art methods in both performance and interpretability.

Qingshan Li is the corresponding author.

ACM ISBN 979-8-4007-1248-7/24/10
https://doi.org/10.1145/3691620.3695291

## CCS Concepts

• **Software and its engineering** → **Collaboration in software development**.

## Keywords

Code Reviewer Recommendation, Large Language Model.

**ACM Reference Format:**
Luqiao Wang, Yangtao Zhou, Huiying Zhuang, Qingshan Li, Di Cui, Yutong Zhao, and Lu Wang. 2024. Unity Is Strength: Collaborative LLM-Based Agents for Code Reviewer Recommendation. In *39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3691620.3695291

## 1 Introduction

Code review (CR) is a critical practice in software maintenance, aimed at reducing defects and improving software quality [9, 20]. In pull-based development, contributors submit pull requests (PRs) for review prior to integration into the main branch. Reviewers assess the changes, discussing their relevance and quality, and make approval decisions. Thongtanunam et al. [17] reported that 4% to 30% of reviews experience issues with reviewer assignment, delaying the CR process and extending the approval time to an average of 12 days. Despite advancements, the diversity in PR and reviewer information complicates the task of recommending suitable reviewers.

Code reviewer recommendation (CRR) aims to identify suitable reviewers by analyzing pull request (PR) features and reviewers' historical interactions. Existing CRR methods can be divided into PR semantic-oriented and PR-reviewer interaction-oriented approaches. PR semantic-oriented methods analyze modified code, file paths, and PR comments to find reviewers who have handled similar PRs. However, since PR data is mostly textual, methods based on character matching or vector space mapping often fail to capture deeper semantics. PR-reviewer interaction-oriented approaches focus on historical relationships between PRs and reviewers, but the complexity introduced by multiple factors, along with limited semantic understanding, often results in suboptimal performance.

Large language models (LLMs) demonstrate remarkable capabilities in understanding and reasoning with textual information [11, 15, 23]. Their extensive knowledge base and powerful semantic processing enable deep analysis of natural language, such as code changes, file paths, PR descriptions, reviewer comments, and more. AI agents excel in planning, decision-making, and tool invocation, allowing them to break down complex tasks into manageable subtasks [4, 16]. By leveraging these strengths, LLM-based AI agents can effectively extract semantic relations between code changes and reviewers.

The procedure of our approach: **Co**llaborative LLM-based agents for **Co**de **Re**viewer **Re**commendation (**CoRe**), employs six distinct roles performed by ChatGPT-3.5: *Coordinator*, *Code Analyzer*, *Reviewer Evaluator*, *Supervisor*, *Data Retriever*, and *Reasoning Explainer*. The Coordinator manages task distribution and overall system operation. The Code Analyzer and Reviewer Evaluator analyze code changes and reviewer profiles, respectively. The Supervisor assesses the recommendations generated by the Coordinator to ensure they meet quality standards. If necessary, the Data Retriever gathers additional information to refine the recommendations. The Reasoning Explainer provides detailed explanations for the recommendations, ensuring transparency and interpretability of the decision-making process. This collaborative approach leverages the strengths of each role to enhance the accuracy and efficiency of the CRR process.

## 2 Related Work

In this section, we explore various approaches that enhance the understanding of PR semantics and techniques focusing on the interactions between PRs and reviewers. We also illustrate emerging issues and challenges within these fields.

**PR semantic-oriented.** Balachandran [2] suggests that the best reviewers are those who have previously modified or reviewed the code sections in the current review, using a method called Review Bot based on line-by-line code modification history. Similarly, Thongtanunam et al. [17] model the semantics of PRs using file path similarity. Furthermore, Xia et al. [18] introduce TIE, a text mining and file location-based approach that enhances PR semantic information through the textual description of code changes. Kim et al. [7] use Latent Dirichlet Allocation (LDA), a topic modeling algorithm, to extract topics from source code changes.

**PR-reviewer interaction-oriented.** Yu et al. [21] capture the relationships between contributors and developers by constructing a comment network for PRs. Jiang et al. [5] extend this approach by categorizing comment types into follower relations, following relations, prior evaluations, and recent evaluations, which are then used as features to train a classifier. Yang et al. [19] enhance the reviewer profile by considering the number of lines of code reviewed and the corresponding timestamps. Zanjani et al. [22] propose an approach called cHRev, which utilizes the number of comments and the duration of review days to assess a reviewer's expertise. This method incorporates recency to account for the decline in expertise over time, effectively recommending reviewers at the file level in most cases. Rahman et al. [12] extend the relationship between PRs and reviewers by exploring cross-project contributions, broadening the scope of reviewer assessment beyond single-project work. Jiang et al. [6] extract activeness, text similarity, file similarity, and social relations from comment and review history as attributes to generate a recommendation list. Their findings indicate that activeness is the most important attribute in predicting commenters. Rong et al. [14] employ hypergraph techniques to capture higher-order relationships among PRs, reviewers, and contributors, such as one PR with multiple reviewers.

Recently, several studies explored the potential of ensemble strategies in CRR. Asthana et al. [1] discovered a Pareto effect in the distribution of review tasks. They proposed the WhoDo algorithm, which demonstrates that load balancing can significantly reduce PR completion time. Mirsaeedi et al. [10] introduced the concept of Files at Risk (FaR) to facilitate knowledge transfer and mitigate the negative impacts of turnover. Rebai et al. [13] defined CRR as finding a trade-off between multiple objectives, such as expertise, availability of peer reviewers, and collaboration history. They employed the NSGA-II algorithm to achieve effective results. Similarly, Chouchen et al. [3] utilized the indicator-based evolutionary algorithm (IBEA) to balance reviewer expertise and workload. These approaches underscore the necessity of integrating multiple factors for optimal reviewer recommendations. However, they still fall short in addressing the full complexity of the task. We can further improve by employing an ensemble strategy that leverages the strengths of AI agents in solving complex tasks and the advanced semantic understanding capabilities of LLM.

## 3 Methodology

In this section, we introduce our approach in detail. As shown in Figure 1, the overall framework of CoRe is comprised of six core AI agents: Coordinator, Code Analyzer, Data Retriever, Reviewer Evaluator, Supervisor, and Reasoning Explainer. Through the collaborative interactions among these AI agents, CoRe can fully leverage the potential of large language models and integrate various factors influencing the code review process to deliver precise and explainable code reviewer recommendations (CRR).

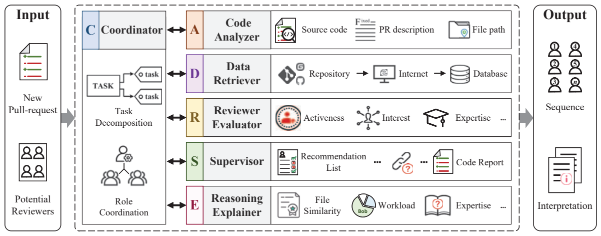


**Figure 1: Overview of the proposed approach.**

**Coordinator.** The *Coordinator* plays a central role in the CRR process by collaborating with other AI agents to achieve effective and reasonable code reviewer recommendations. This process involves three main steps: thought, action, and reflection.

- **Thought Step:** The coordinator reasons what information about the PRs and candidate need to be extracted for CRR.
- **Action Step:** The coordinator generates the recommendation results or calls other AI agents (including the code analyzer, reviewer evaluator, and data retriever) to provide refined information about the PRs and candidate reviewers.
- **Reflection Step:** The coordinator invokes the supervisor agent to check the generated recommendation results by considering multiple factors. If they pass the supervisor's verification, the coordinator calls the reasoning explainer to

generate reasons for the recommendation results and outputs the recommendation results with their reasons; otherwise, the coordinator synthesizes the feedback from the supervisor and re-enters the thought step. To avoid infinite loops, we set a threshold for the maximum number of thought steps. If the count of the coordinator's thought step exceeds this threshold, the last round of recommendation results and reasons will be output directly.

By following this structured process, the Coordinator ensures that the CRR task is performed effectively, leveraging the strengths of each agent to deliver precise and explainable recommendations.

**Code Analyzer.** The Code Analyzer is designed to understand and encapsulate the characteristics of a given PR using prompt-driven LLMs. To facilitate this, the Code Analyzer is equipped with an information retrieval tool that extracts data about the PR from databases or storage files. This tool allows the Code Analyzer to access essential details such as modified source code, textual descriptions, programming languages, comments, and associated file paths. Following data acquisition, the Code Analyzer synthesizes these details to delineate the profiles of the code modifications, thereby generating a comprehensive and insightful summary of the given PR. This summary is then relayed back to the Coordinator.

**Reviewer Evaluator.** The Reviewer Evaluator aims to refine the profiles of candidate reviewers by leveraging LLMs prompted with key attributes, such as their historically reviewed PRs and associated comments. To achieve a comprehensive understanding, the Reviewer Evaluator can invoke the Code Analyzer to provide detailed summaries of these historical PRs. By analyzing the historical review data, the Reviewer Evaluator can effectively identify the collaborative filtering signals to perceive similarity patterns among reviewers, which is critical for precise recommendations. Ultimately, the Reviewer Evaluator generates informative profiles of candidate reviewers with the collaborative filtering signals and conveys these to the Coordinator.

**Supervisor.** The Supervisor serves as a quality control agent, ensuring the recommendations meet predefined standards and criteria. This feedback is crucial for maintaining the reliability and accuracy of the CRR process. The Supervisor evaluates the list of CRRs generated by the Coordinator, ensuring they meet criteria such as expertise relevance, past review performance, and specific requirements of the current PR. The Supervisor also considers factors highlighted in previous research, including the reviewers' activeness [5], workload [1], and risk of knowledge loss [10]. If the recommendations meet all these standards, the Supervisor approves them for final submission. If not, the Supervisor identifies areas for improvement and requests further refinement. This rigorous evaluation process ensures that only the most appropriate reviewers are recommended, thereby enhancing the effectiveness of the code review process.

**Data Retriever.** The Data Retriever supports the system by gathering additional information to ensure accurate CRRs. This role is vital for enriching the data available to other agents, particularly when initial analyses are insufficient. The Data Retriever collects detailed information from external sources such as GitHub, including additional code context, history of code changes, and relevant discussions. This agent supplements the findings of the Code Analyzer and Reviewer Evaluator by providing deeper insights and more extensive data. By accessing a broader range of information, the Data Retriever helps refine and enhance the CRRs, ensuring they are well-informed and thoroughly scrutinized.

**Reasoning Explainer.** The Reasoning Explainer is responsible for generating clear explanations for the CRRs. This role enhances the transparency and interpretability of the recommendation process. The Reasoning Explainer articulates the reasoning behind the Coordinator's recommendations by compiling and presenting the information analyzed by the Code Analyzer, Reviewer Evaluator, and Data Retriever. This includes explaining how the reviewers' expertise aligns with the specific code changes, the relevance of their past review performance, and any other pertinent factors considered during the evaluation. By providing detailed and well-founded justifications, the Reasoning Explainer enhances the credibility and acceptability of the CRRs, ensuring that all stakeholders understand the criteria for the selections.

## 4 Empirical Evaluation

To evaluate the effectiveness of our approach, we conduct an empirical study on four open-source software systems frequently used in prior research. And we aim to answer the following questions:

- **RQ1:** How effective is our approach CoRe in CRR? How effective is LLM-based agents collaboration in CoRe?
- **RQ2:** How is the Rationality and interpretability of CoRe?

## 4.1 Experimental Settings

**Dataset.** In our preliminary study, we randomly sampled PRs and their review records from four real-world projects: android, openstack, qt, and libreoffice. The details of sampled data are shown in Table 1. These projects were selected due to their diversity in size, domain, and development practices, which helps in evaluating the generalizability of our approach.

**Table 1: Statistics of sampled dataset.**

| System | Language | Files | #PR | #Reviewer | Year |
|---|---|---|---|---|---|
| android | C/Java | 51 | 19 | 12 | 2008-2012 |
| openstack | C++ | 36 | 16 | 6 | 2011-2012 |
| qt | Python | 42 | 30 | 11 | 2011-2012 |
| libreoffice | C++ | 111 | 9 | 7 | 2012-2014 |
| Total | / | 240 | 80 | 38 | / |

**Evaluation Metrics.** To evaluate the performance of the top-k recommendations, we adopt two widely used metrics accuracy (*Acc@k*) and Mean Reciprocal Rank (*MRR@k*) [14], where $k$ is set to 1, 3, and 5. Higher *Acc@k* and *MRR@k* values reflect superior CRR performance. Besides, we introduce a case study to evaluate the rationality of the recommendation results.

**Implementation Details.** We utilized GPT-3.5-turbo-16k to implement the various agents during the recommendation process. This model was chosen for its advanced language understanding, planning, and decision-making capabilities, which are crucial for tasks such as code analysis, reviewer evaluation, and rationale explanation. Additionally, we employed LangChain to manage the interaction between different agents and streamline the workflow. LangChain's framework allows us to efficiently integrate and coordinate the activities of our multi-agent system, ensuring seamless communication and task execution.

**Table 2: Performance of various approach. The greatest Acc@k and MRR@k are highlighted with gray background.**

| | CoRe | | | | | | ChatGPT | | | | | | RevFinder | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Acc@1* | *Acc@3* | *Acc@5* | *MRR@1* | *MRR@3* | *MRR@5* | *Acc@1* | *Acc@3* | *Acc@5* | *MRR@1* | *MRR@3* | *MRR@5* | *Acc@1* | *Acc@3* | *Acc@5* | *MRR@1* | *MRR@3* | *MRR@5* |
| **Android** | **0.905** | **0.952** | **0.952** | **0.905** | **0.929** | **0.929** | 0.810 | 0.857 | 0.905 | 0.810 | 0.833 | 0.845 | 0.667 | **0.952** | **0.952** | 0.667 | 0.802 | 0.802 |
| **Openstack** | **0.941** | **1** | **1** | **0.941** | **0.971** | **0.971** | 0.588 | 0.647 | 0.824 | 0.588 | 0.608 | 0.649 | 0.529 | 0.824 | 0.941 | 0.529 | 0.657 | 0.683 |
| **Qt** | **0.714** | **0.971** | **1** | **0.714** | **0.829** | **0.836** | 0.343 | 0.829 | 0.914 | 0.343 | 0.552 | 0.572 | 0.514 | 0.771 | 0.886 | 0.514 | 0.638 | 0.667 |
| **LibreOffice** | **0.714** | **1** | **1** | **0.714** | **0.857** | **0.857** | 0.571 | 0.857 | 0.857 | 0.571 | 0.691 | 0.691 | 0.429 | 0.714 | **1** | 0.429 | 0.548 | 0.612 |
| **Average** | **0.819** | **0.981** | **0.988** | **0.819** | **0.896** | **0.898** | 0.578 | 0.797 | 0.875 | 0.578 | 0.671 | 0.689 | 0.535 | 0.815 | 0.945 | 0.535 | 0.661 | 0.691 |

## 4.2 Performance Comparison (RQ1)

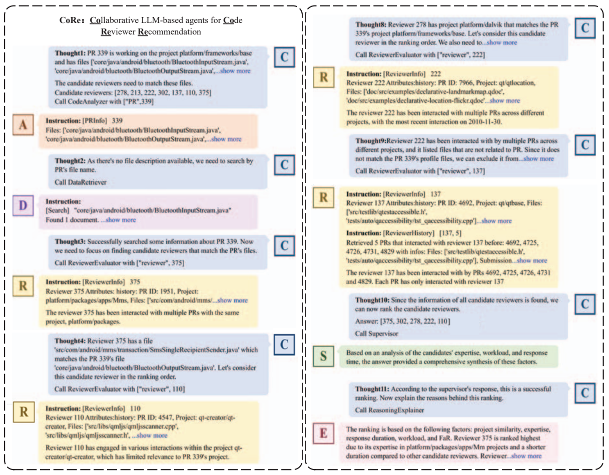
To verify the superiority and effectiveness of our proposed CoRe approach, we compare its recommendation performance with two baselines: ChatGPT and RevFinder. ChatGPT is an LLM-based approach that prompts GPT-3.5-turbo-16k as a code reviewer recommender to achieve recommendations, and RevFinder is a classical recommendation approach based on PR path similarity. The experimental results are detailed in Table 2, highlighting the best accuracy (Acc) and Mean Reciprocal Rank (MRR) for top-1, top-3, and top-5 recommendations. By analyzing the results, we can draw the following conclusion.

First, our proposed CoRe approach outperforms all baselines, which highlights the effectiveness and superiority of our approach. The key advantages of our CoRe include three aspects: 1) We innovatively introduce an LLM to capture rich semantic characteristics for both PRs and reviewers from existing textual data, such as file paths, descriptions of PRs, and comment records. 2) To address the complex CRR tasks, we carefully devise a collaborative agent system to maximize the semantic understanding and inference potential of LLMs. To the best of our knowledge, our proposed approach is the first work to incorporate LLM-based AI agents into CRR tasks, significantly improving the accuracy and interpretability of recommendations. Second, ChatGPT exhibits superiority over classical approach RevFinder, indicating that LLMs can effectively perceive the semantic characteristics of PRs and reviewers to boost recommendation performance. Finally, our proposed CoRe achieves significant improvements over the suboptimal approach ChatGPT. This result demonstrates that multiple AI Agents can effectively improve the recommendation performance, benefiting from the significant impact of collaboration.

## 4.3 Case Study (RQ2)

In this section, we present an case study to provide an intuitive impression of CoRe's effectiveness and interpretability, which the top-1 recommendation result does not hit the ground truth. We randomly select a PR 339 with file path `platform/frameworks/base` from `android` project, and generate a top-5 recommendation list [375, 302, 278, 222, 110] from the given candidate reviewer pool. The ground truth reviewer ID is 278, and our approach CoRe ranks this reviewer in the third position. The explanation of the reasoning process is shown in Fig 2 (Due to space concern, we only present part of the reasoning process of CoRe). Although CoRe did not rank the ground truth reviewer as the top candidate, the recommendations were based on reasonable inferences and valid explanations:

**Reviewer 375** reviewed PRs in `platform/packages`, which shares the same top-level path (`platform`) with the target PR. The average review time is 10 days, and they have reviewed only four PRs in the past 6 months, indicating a relatively low workload.



**CoRe:** Collaborative LLM-based agents for Code Reviewer Recommendation

**Thought1:** PR 339 is working on the project platform/frameworks/base and has files ['core/java/android/bluetooth/BluetoothInputStream.java', 'core/java/android/bluetooth/BluetoothOutputStream.java',...show more
The candidate reviewers need to match these files.
Candidate reviewers: [278, 213, 222, 302, 137, 110, 375]
Call CodeAnalyzer with ["PR",339]

**Instruction:** [PRInfo] 339
Files: ['core/java/android/bluetooth/BluetoothInputStream.java', 'core/java/android/bluetooth/BluetoothOutputStream.java',...show more

**Thought2:** As there's no file description available, we need to search by PR's file name.
Call DataRetriever

**Instruction:**
[Search] "core/java/android/bluetooth/BluetoothInputStream.java"
Found 1 document. ...show more

**Thought3:** Successfully searched some information about PR 339. Now we need to focus on finding candidate reviewers that match the PR's files.
Call ReviewerEvaluator with ["reviewer", 375]

**Instruction:** [ReviewerInfo] 375
Reviewer 375 Attributes: history: PR ID: 1951, Project: platform/packages/apps/Mms, Files: ['src/com/android/mms/...show more
The reviewer 375 has been interacted with multiple PRs with the same project, platform/packages.

**Thought4:** Reviewer 375 has a file 'src/com/android/mms/transaction/SmsSingleRecipientSender.java' which matches the PR 339's file 'core/java/android/bluetooth/BluetoothOutputStream.java'. Let's consider this candidate reviewer in the ranking order.
Call ReviewerEvaluator with ["reviewer", 110]

**Instruction:** [ReviewerInfo] 110
Reviewer 110 Attributes:history: PR ID: 4547, Project: qt-creator/qt-creator, Files: ['src/libs/qmljs/qmljsscanner.cpp', 'src/libs/qmljs/qmljsscanner.h', ...show more
Reviewer 110 has engaged in various interactions within the project qt-creator/qt-creator, which has limited relevance to PR 339's project.

**Thought8:** Reviewer 278 has project platform/dalvik that matches the PR 339's project platform/frameworks/base. Let's consider this candidate reviewer in the ranking order. We also need to...show more
Call ReviewerEvaluator with ["reviewer", 222]

**Instruction:** [ReviewerInfo] 222
Reviewer 222 Attributes:history: PR ID: 7966, Project: qt/qtlocation, Files: ['doc/src/examples/declarative-landmarkmap.qdoc', 'doc/src/examples/declarative-location-flickr.qdoc'...show more
The reviewer 222 has been interacted with multiple PRs across different projects, with the most recent interaction on 2010-11-30.

**Thought9:**Reviewer 222 has been interacted with by multiple PRs across different projects, and it listed files that are not related to PR. Since it does not match the PR 339's profile files, we can exclude it from...show more
Call ReviewerEvaluator with ["reviewer", 137]

**Instruction:** [ReviewerInfo] 137
Reviewer 137 Attributes:history: PR ID: 4692, Project: qt/qtbase, Files: ['src/testlib/qtestaccessible.h', 'tests/auto/qaccessibility/tst_qaccessibility.cpp']...show more
**Instruction:** [ReviewerHistory] [137, 5]
Retrieved 5 PRs that interacted with reviewer 137 before: 4692, 4725, 4726, 4731, 4829 with infos: Files: ['src/testlib/qtestaccessible.h', 'tests/auto/qaccessibility/tst_qaccessibility.cpp'], Submission...show more
The reviewer 137 has been interacted with by PRs 4692, 4725, 4726, 4731 and 4829. Each PR has only interacted with reviewer 137

**Thought10:** Since the information of all candidate reviewers is found, we can now rank the candidate reviewers.
Answer: [375, 302, 278, 222, 110]
Call Supervisor

Based on an analysis of the candidates' expertise, workload, and response time, the answer provided a comprehensive synthesis of these factors.

**Thought11:** According to the supervisor's response, this is a successful ranking. Now explain the reasons behind this ranking.
Call ReasoningExplainer

The ranking is based on the following factors: project similarity, expertise, response duration, workload, and FaR. Reviewer 375 is ranked highest due to its expertise in platform/packages/apps/Mm projects and a shorter duration compared to other candidate reviewers. Reviewer...show more

**Figure 2: Case study on CRR. C, A, D, R, S, and E denote Coordinator, Code Analyzer, Data Retriever, Reviewer Evaluator, Supervisor, and Reasoning Explainer, respectively.**

**Reviewer 302** reviewed some PRs with similar paths to the target PR. Their average review time is about 16 days, and they have reviewed 10 PRs in the past six months, indicating a higher workload.
**Reviewer 278** has reviewed PRs that share the same top-level path (`platform`) as the target PR. Although their review time is short, averaging just 3 days, they have reviewed 18 PRs in the past six months, indicating a very high workload.
**Reviewers 222 and 110** have not reviewed any PRs in the `platform` project and have gradually increasing workloads.

This analysis demonstrates that while CoRe does not perfectly align with the historical ground truth in this case, its recommendations are based on a comprehensive analysis considering various pertinent factors, such as review history, path similarity, and workload. This not only highlights the interpretability and effectiveness of CoRe's recommendations but also underscores the challenges posed by poor quality historical data in existing approaches [8].

## 5 Conclusion and Future Work

In this paper, we propose CoRe, a novel collaborative agent-based approach for code reviewer recommendation. Our approach leverages ChatGPT to accurately capture the rich textual semantics of both pull requests and reviewers. By integrating various factors into the recommendation process through AI agents, we enhance reviewer diversity and improve recommendation quality. Our findings show that CoRe significantly increases accuracy and mean reciprocal rank, especially in top-1 performance, compared to existing approaches on four widely used datasets. Overall, CoRe advances code reviewer recommendation by integrating LLM-based agents, marking a significant step forward in code review.

## References

[1] Sumit Asthana, Rahul Kumar, Ranjita Bhagwan, Christian Bird, Chetan Bansal, Chandra Maddila, Sonu Mehta, and Balasubramanyan Ashok. 2019. Whodo: Automating reviewer suggestions at scale. In *Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*. 937–945.

[2] Vipin Balachandran. 2013. Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 931–940.

[3] Moataz Chouchen, Ali Ouni, Mohamed Wiem Mkaouer, Raula Gaikovina Kula, and Katsuro Inoue. 2021. WhoReview: A multi-objective search-based approach for code reviewers recommendation in modern code review. *Applied Soft Computing* 100 (2021), 106908.

[4] Xu Huang, Jianxun Lian, Yuxuan Lei, Jing Yao, Defu Lian, and Xing Xie. 2023. Recommender ai agent: Integrating large language models for interactive recommendations. *arXiv preprint arXiv:2308.16505* (2023).

[5] Jing Jiang, Jia-Huan He, and Xue-Yuan Chen. 2015. Coredevrec: Automatic core member recommendation for contribution evaluation. *Journal of Computer Science and Technology* 30 (2015), 998–1016.

[6] Jing Jiang, Yun Yang, Jiahuan He, Xavier Blanc, and Li Zhang. 2017. Who should comment on this pull request? analyzing attributes for more accurate commenter recommendation in pull-based development. *Information and Software Technology* 84 (2017), 48–62.

[7] Jungil Kim and Eunjoo Lee. 2018. Understanding Review Expertise of Developers: A Reviewer Recommendation Approach Based on Latent Dirichlet Allocation. *Symmetry* 10, 4 (2018). https://doi.org/10.3390/sym10040114

[8] Oleksii Kononenko, Olga Baysal, Latifa Guerrouj, Yaxin Cao, and Michael W Godfrey. 2015. Investigating code review quality: Do people and participation matter?. In *2015 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 111–120.

[9] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. 2016. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering* 21 (2016), 2146–2189.

[10] Ehsan Mirsaeedi and Peter C Rigby. 2020. Mitigating turnover with code review recommendation: balancing expertise, workload, and knowledge distribution. In *Proceedings of the ACM/IEEE 42nd international conference on software engineering*. 1183–1195.

[11] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems* 35 (2022), 27730–27744.

[12] Mohammad Masudur Rahman, Chanchal K Roy, and Jason A Collins. 2016. Correct: code reviewer recommendation in github based on cross-project and technology experience. In *Proceedings of the 38th international conference on software engineering companion*. 222–231.

[13] Soumaya Rebai, Abderrahmen Amich, Somayeh Molaei, Marouane Kessentini, and Rick Kazman. 2020. Multi-objective code reviewer recommendations: balancing expertise, availability and collaborations. *Automated Software Engineering* 27 (2020), 301–328.

[14] Guoping Rong, Yifan Zhang, Lanxin Yang, Fuli Zhang, Hongyu Kuang, and He Zhang. 2022. Modeling review history for reviewer recommendation: A hypergraph approach. In *Proceedings of the 44th international conference on software engineering*. 1381–1392.

[15] Wentao Shi, Xiangnan He, Yang Zhang, Chongming Gao, Xinyue Li, Jizhi Zhang, Qifan Wang, and Fuli Feng. 2024. Enhancing Long-Term Recommendation with Bi-level Learnable Large Language Model Planning. *arXiv preprint arXiv:2403.00843* (2024).

[16] Yubo Shu, Hansu Gu, Peng Zhang, Haonan Zhang, Tun Lu, Dongsheng Li, and Ning Gu. 2023. RAH! RecSys-Assistant-Human: A Human-Central Recommendation Framework with Large Language Models. *arXiv preprint arXiv:2308.09904* (2023).

[17] Patanamon Thongtanunam, Chakkrit Tantithamthavorn, Raula Gaikovina Kula, Norihiro Yoshida, Hajimu Iida, and Ken-ichi Matsumoto. 2015. 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)*. IEEE, 141–150.

[18] Xin Xia, David Lo, Xinyu Wang, and Xiaohu Yang. 2015. 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)*. IEEE, 261–270.

[19] Cheng Yang, Xun-hui Zhang, Ling-bin Zeng, Qiang Fan, Tao Wang, Yue Yu, Gang Yin, and Huai-min Wang. 2018. RevRec: A two-layer reviewer recommendation algorithm in pull-based development model. *Journal of Central South University* 25, 5 (2018), 1129–1143.

[20] Zezhou Yang, Cuiyun Gao, Zhaoqiang Guo, Zhenhao Li, Kui Liu, Xin Xia, and Yuming Zhou. 2024. A Survey on Modern Code Review: Progresses, Challenges and Opportunities. *arXiv preprint arXiv:2405.18216* (2024).

[21] Yue Yu, Huaimin Wang, Gang Yin, and Charles X Ling. 2014. Who should review this pull-request: Reviewer recommendation to expedite crowd collaboration. In *2014 21st Asia-Pacific Software Engineering Conference*, Vol. 1. IEEE, 335–342.

[22] Motahareh Bahrami Zanjani, Huzefa Kagdi, and Christian Bird. 2015. Automatically recommending peer reviewers in modern code review. *IEEE Transactions on Software Engineering* 42, 6 (2015), 530–543.

[23] Yaochen Zhu, Liang Wu, Qi Guo, Liangjie Hong, and Jundong Li. 2023. Collaborative large language model for recommender systems. *arXiv preprint arXiv:2311.01343* (2023).