Topic Modelling-Based Code Review Hotspot Prediction

Tongwei Zhang
University of Waterloo
Waterloo, Canada
t98zhang@uwaterloo.ca

Zhaoyi Ge
University of Waterloo
Waterloo, Canada
zhaoyi.ge@uwaterloo.ca

Zhiao Wei
University of Waterloo
Waterloo, Canada
zhiao.wei@uwaterloo.ca

Abstract

Abstract— Code review is expensive and time-sensitive, and it is increasingly shaped by agent-authored pull requests (Agentic PRs), where autonomous coding agents produce multi-file changes. Reviewers must quickly determine where to focus their attention, yet existing work lacks a reproducible definition of review hotspots and baselines for predicting them. In this study, we construct a file-level hotspot ground truth for Agentic PRs using review comments and follow-up commits from the AIDev dataset and derive attention distributions. We then test whether the attention distributions combined with semantic signals from PR metadata can predict these hotspots through a simple, interpretable topic–file-category model. The model achieves strong Hit@k, nDCG@k, and distributional similarity scores. Our contributions include the hotspot dataset, a prediction baseline, and empirical insights into the evaluation results.

ACM Reference Format:
Tongwei Zhang, Zhaoyi Ge, and Zhiao Wei. 2025. Topic Modelling-Based Code Review Hotspot Prediction. In . ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Software development increasingly happens through pull requests, and those conversations now often involve AI coding agents such as GitHub Copilot, Cursor, and Claude Code, which propose patches, rework modules, and scaffold tests. As agent-authored pull requests (Agentic-PRs) become more common, reviewers face a practical bottleneck: in a multi-file patch with a long discussion thread, where should they look first?

Code review is expensive and time-sensitive; reviewers must decide what to read and in what order, often under pressure [1]. With agent-produced patches, this problem becomes harder because of stylistic variation, multi-module edits, and vague rationales. If a tool could reliably predict the top-k files likely to draw comments, teams could focus attention better, reduce delays, and improve review quality. Our goal is to provide a small, evidence-based "attention map" that helps reviewers prioritize their first passes.

Prior work has characterized code review processes and outcomes, but we lack a shared, reproducible way to define "review hotspots" at file level using observable evidence from public repositories [1]. We also lack transparent baseline models that are simple enough to run across ecosystems yet strong enough to beat heuristics based only on code churn when predicting those hotspots from a PR's text and structure. Agentic-PRs make this harder: agent patches often bundle changes across loosely coupled modules, and their descriptions may not map cleanly to subsystem boundaries [6].

We address this gap by defining hotspots using two observable signals in AIDev: inline or file-scoped code review comments that belong to specific files or lines, and follow-up commits during review that touch those files. From these signals, we derive per-file hotspot labels and construct an *attention distribution* over changed files based on where reviewers actually left comments. For future PRs, we then predict review hotspots from the PR's description and diff context using topic-level signals derived from topic modelling. This design connects to classic work on degree of interest and developer context, particularly Kersten and Murphy's Mylar [5], but applies it to the code review setting, where "interest" comes from the public record of comments rather than IDE traces.

1.1 Research Questions

We structure our study around the following research questions:

RQ1. How can we define review hotspots in agentic PRs?
We operationalize review hotspots using observable signals, specifically review comments and follow-up commits, and develop a systematic approach to identify and label hotspot files from the AIDev dataset, producing per-file hotspot labels and comment-based attention distributions.

Conference'17, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

RQ2. What are the characteristics of review hotspots in agentic PRs?
We characterize review hotspots by analyzing comment and follow-up commit patterns across files in AIDev, focusing on how hotspots relate to coarse file categories (e.g., source, tests, configuration) and how reviewer attention is distributed across changed files in a PR.

RQ3. How accurately can we predict review hotspots from PR descriptions and diffs?
We design a simple, transparent model that combines topic-level information with file-level attributes and evaluate its ability to rank files by expected reviewer attention using Hit@k, Precision@k, Recall@k, and nDCG@k. We also compare its predicted attention distributions to the empirical distributions derived from comments.

RQ4. What topics drive hotspot predictions?
We study how latent topics influence hotspot formation and prediction by analyzing topic-level patterns in the density of commented files and how topic assignments interact with file categories, highlighting which topics are most strongly associated with higher reviewer attention.

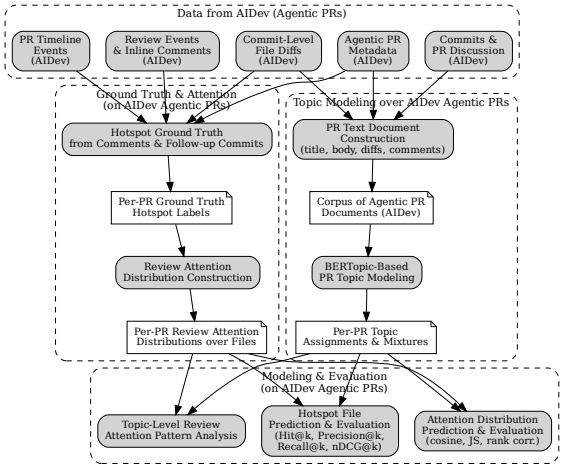

Figure 1. Overview of our topic-modelling-based code review hotspot prediction pipeline. We derive hotspot ground truth and review attention distributions from repository and review artifacts, and combine them with BERTopic-based PR topics for attention and hotspot prediction.

2 Approach

2.1 Dataset

We base our work on the AIDev dataset [6], a large-scale dataset comprised of 456,535 Agentic PRs created by Autonomous Coding Agents on GitHub. Specifically, the dataset features 7,122 Agentic PRs and 6,628 human PRs from popular repositories with more than 500 stars. For each PR, the dataset contains all associated review comments and commits; each commit includes its message and the set of modified files.

We primarily use the AIDev-Pop subset, which only contains PRs from popular repositories. This helps improve the quality of review comments and increases the amount of data available per repository. In our pipeline, we ultimately operate on the intersection of PRs for which we can (i) derive file-level hotspot and attention information from review artifacts and commit metadata, and (ii) construct a corresponding PR-level text document for topic modelling.

2.2 Defining Attention and Hotspots

For each PR in a repository, our first step is to define an *attention signal* at the file level and derive hotspot labels.

We construct per-PR file sets using commit-level metadata. From the commit details associated with each PR, we extract the set of *changed files*. We then link review artifacts back to these files using two observable signals from the AIDev dataset [6]:

- **Review comments.** We use inline or file-scoped review comments that are attached to specific files or lines. Aggregating these comments per PR yields a set of *commented files*.
- **Follow-up commits.** We analyze the PR review timeline to locate follow-up commits that occur after the first review event, and then map those commits back to the files they touch. Aggregating these per PR yields a set of *follow-up commit files*.

Using these signals, we define ground-truth hotspot labels as follows. For each PR, we take the union of the commented files and follow-up commit files. Any changed file in this union is labeled as a *hotspot file*; all other changed files are labeled as non-hotspot. We also maintain, for each PR, a per-file record indicating whether the file was changed, commented on, modified in a follow-up commit, and whether it is considered a hotspot.

For modelling reviewer attention as a distribution, we derive a separate, comment-based signal. For each PR, we start from the set of changed files and assign a positive attention score to files that received at least one review comment, with zero scores for other changed files. We then normalize these scores to obtain a probability distribution over the changed files. If a PR has no commented files, we fall back to a uniform distribution over all changed files. This distribution serves as the empirical "attention map" that our models aim to approximate.

2.3 Topic Modelling and Feature Extraction

For each PR, we construct a unified text "document" that combines multiple sources of information about the change. Using the AIDev metadata [6], we concatenate:

- the PR title and description,
- commit messages associated with the PR,
- discussion comments attached to the PR, and
- a textual representation of file-level diff hunks, grouped by filename and annotated with markers that identify the corresponding files.

This yields one document per PR that captures both natural language discussion and code-level context. We then apply a topic modelling pipeline based on BERTopic to this corpus. The model takes the PR documents as input and produces, for each PR, a dominant topic assignment and an associated topic mixture over latent topics.

We store these results in a per-PR topic summary that includes the dominant topic, a sparse probability distribution over topics, and the list of changed files. In the subsequent modelling stages, we primarily use the dominant topic as a compact representation of the PR's high-level intent and combine it with coarse file-level information derived from filenames.

2.4 Hotspot Prediction Model

We design a simple, transparent hotspot prediction model that operates at the level of *topic* × *file category*. For each changed file, we first assign a coarse file category using a filename-based classifier (e.g., "test", "docs", "config", "build", "source", or "other"). This yields, for each PR, a set of (file, file-category) pairs along with a dominant topic for the PR.

On the training set, we aggregate evidence across PRs to learn, for each topic, which file categories tend to attract reviewer attention. Concretely, for every combination of dominant topic and file category, we count how many changed files of that category in PRs with that topic received non-zero attention in the empirical attention distributions, and how many such files were present in total. From these counts, we identify, per topic, the set of file categories that have been observed as "attention-worthy" (i.e., categories for which at least one file has received comments).

At prediction time, given a new PR, we use its dominant topic and the categories of its changed files to assign scores. Files whose categories have been observed as attention-worthy for the PR's topic receive a positive score; other changed files receive a lower (or zero) score. We use these scores in two ways:

- by normalizing them over the changed files to obtain a predicted attention distribution, and
- by sorting them in descending order to obtain a ranking of changed files from most to least likely hotspot.

This topic-category-based and file-category-based model deliberately avoids complex features and black-box predictors, providing a clear baseline that is easy to interpret and reproduce.

2.5 Evaluation Protocol and Metrics

We evaluate our approach at two levels: distributional similarity and ranking quality. To assess how well the model reproduces reviewers' empirical attention patterns, we compare the predicted attention distributions to the comment-based ground-truth distributions on a held-out test set of PRs. For this comparison, we use standard measures of distribution and ranking similarity, including cosine similarity, Jensen–Shannon divergence, and rank correlation between per-file attention scores.

To assess hotspot ranking quality, we use the hotspot labels derived from comments and follow-up commits. For each PR in the test set, we take the model's ranking of changed files and compute:

- Hit@k
- Precision@k
- Recall@k
- nDCG@k

for several values of k (e.g., $k = 1, 3, 5, 10$). We perform training and evaluation over disjoint sets of PRs, using a random split at the PR level so that no PR appears in both training and test data. This protocol allows us to quantify how well a simple topic- and file-category-based model can anticipate where reviewers will focus their attention in agentic pull requests.

3 Results

3.1 RQ1: Defining Review Hotspots

Our analysis of review attention data reveals that reviewer attention is highly sparse. A file is defined as a "hotspot" if it receives any reviewer attention (attention score > 0). Across all learned topics, the mean attention probability is only 2.53%, with a median of 0.56% (Std: 4.86%). This sparsity confirms that in most Pull Requests, reviewers focus on a very small subset of files, validating the need for precision in hotspot prediction.

3.2 RQ2: Characteristics of Review Hotspots

To characterize these hotspots, we employed a "Topic × File Type" classifier. This approach implicitly characterizes hotspots by learning which file types (e.g., `test`, `docs`, `config`, `build`, `source`) are most relevant for a given topic. For instance, certain topics may drive attention primarily to `source` and `test` files, while others focus on `config` or `docs`. The model's success (see RQ3) suggests that the combination of semantic topic context and file type is a strong predictor of reviewer attention.

3.3 RQ3: Prediction Accuracy

We evaluated the model's ability to predict hotspot files using both ranking and distribution metrics. **Ranking Metrics:** The model achieves high accuracy in identifying the most important files.

- **Hit@k:** The probability that at least one relevant file is in the top-k predictions is extremely high: Hit@1 is 0.996, Hit@3 is 0.999, and Hit@10 is 1.000.
- **NDCG@k:** The Normalized Discounted Cumulative Gain, which accounts for ranking quality, is also strong: NDCG@1 is 0.919, improving to 0.944 at NDCG@10.
- **Precision/Recall:** Precision@1 is 0.920, indicating that the top predicted file is a true hotspot 92% of the time. Recall@10 reaches 0.997, showing that the top 10 predictions capture almost all relevant files.

Distribution Metrics: When comparing the predicted attention distribution against the ground truth:

- **Cosine Similarity:** 0.924 (Std: 0.220), indicating a very high similarity between the predicted and actual attention vectors.
- **Jensen-Shannon Divergence:** 0.046 (Std: 0.144), further confirming the closeness of the distributions (lower is better).

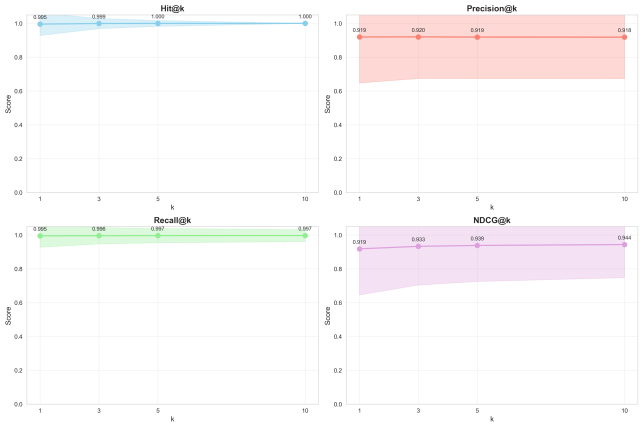

Figure 2. Ranking Metrics Evaluation (Hit@k, Precision@k, Recall@k, NDCG@k) across different k values.

3.4 RQ4: Topics Driving Hotspots

We identified 484 topics and analyzed their tendency to drive review hotspots. The attention probability varies significantly by topic (Min: 0.00%, Max: 39.13%).

- **High-Attention Topics:** The top topics have a much higher likelihood of containing hotspots. For example, Topic 446 (39.13%) and Topic 326 (38.24%) are strong drivers of attention. These topics likely correspond to critical or complex code areas requiring careful scrutiny.

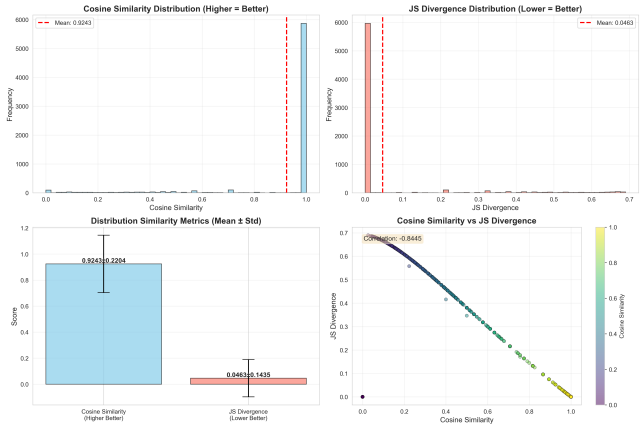

Figure 3. Distribution Metrics Evaluation showing Cosine Similarity and Jensen-Shannon Divergence.

- **Low-Attention Topics:** Conversely, the bottom topics have a 0.00% attention probability, suggesting routine or auto-generated changes that reviewers consistently skip.

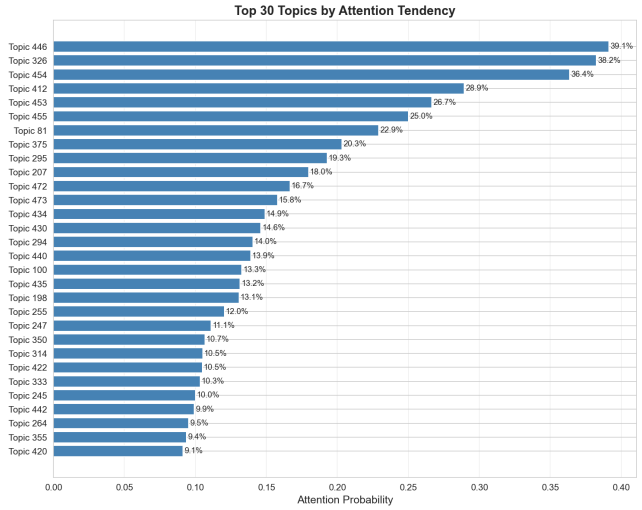

Figure 4. Distribution of Topic Attention Tendencies showing the sparsity of reviewer attention across topics.

4 Discussion

4.1 Implications for Practitioners

For code review tools, these findings suggest that file-category-based attention prediction can be reliably automated. A tool could highlight the top 1-3 predicted files to guide reviewer focus. However, the high variance in topic attention implies that a "human-in-the-loop" approach remains essential; reviewers should use these predictions as cues rather than absolute directives to avoid over-reliance, especially for novel or ambiguous topics.

4.2 Implications for Research

Our approach achieves near-perfect Hit@k rates at the file-category level. However, to further address the 'long tail' of less frequent topics, future research could explore incorporating code complexity metrics or author history.

4.3 Threats to Validity

Construct Validity. Our primary measure of "reviewer attention" is the presence of review comments on a file. This is a proxy metric and may not capture all forms of attention. For instance, a reviewer might carefully read a file but find no issues, resulting in a "silent read" that our model treats as zero attention. Additionally, we treat all comments equally, whereas some comments represent trivial nitpicks while others address critical logic errors. Future work could incorporate dwell time (if available) or sentiment analysis to refine this metric.

Internal Validity. Our prediction model relies on a heuristic-based file type classifier (e.g., mapping `.py` to `source`, `.md` to `docs`). While this covers standard naming conventions, it may misclassify files in projects with non-standard structures or ambiguous extensions. Furthermore, we assume that the dominant topic of a PR is the primary driver of attention distribution. In reality, confounding factors such as PR size, author experience, and code complexity also influence reviewer behavior. We mitigate this by using a large dataset to average out individual variations, but these factors remain unmodeled.

External Validity. Our findings are based on a dataset of Pull Requests from open-source repositories. Review practices in these communities may differ from those in proprietary, industrial environments where strict compliance or security reviews are mandated.

5 Related Work

Our work relates to modern code review, prediction of review activity, defect and hotspot prediction, topic models in software engineering, and recent work on AI coding agents and agentic pull requests.

Modern code review has been widely studied. Bacchelli and Bird showed that review at Microsoft is a lightweight, tool based practice that serves multiple purposes but is constrained by time pressure and uneven participation [1]. Rigby and Bird found that large organizations such as Google, Microsoft, AMD, and major open source projects have converged on similar contemporary review workflows [9]. Thongtanunam et al. observed that participation and discussion are not evenly distributed across patches and that patch characteristics influence whether a change receives sufficient attention [11]. These studies motivate our focus on how attention is allocated within a pull request and why some changed files attract more review activity.

Our notion of an attention distribution over files connects to work on degree of interest. Kersten and Murphy introduced Mylar, which models a developer's task context as a degree of interest over program elements based on interaction histories in the IDE [5]. In their work, degree of interest captures where a programmer spends effort while implementing a change. In our work, attention is derived from review comments and follow up commits: we treat comment and edit counts on each changed file as a proxy for reviewer interest and normalize this into a per pull request distribution over files.

There is also work on predicting which artifacts will attract review activity and how to prioritize review effort. Olewicki et al. studied code review activity prediction at file level and defined three tasks: predicting whether a file will be commented, revised, or be a hot spot that is either commented or revised [8]. They combined text embeddings with review process features and showed that ordering files according to predicted hot spots can increase the number of comments and improve precision and recall on hot spot files compared to alphabetic ordering [8]. Fregnan et al. showed that file position in the review interface affects review outcomes: files shown earlier tend to receive more comments, and for some defects reviewers are much less likely to detect the defect if the file appears last instead of first [3]. Yang et al. looked at prioritizing code review requests at pull request level and evaluated learning to rank models for ordering incoming review requests using process simulation, later extending this line with a simulation study of how different prioritization strategies affect review efficiency [12, 13].

Our work is closest to Olewicki et al. but differs in three ways. First, we work on agent authored pull requests from the AIDev dataset [6], where patch structure and review dynamics reflect the behavior of autonomous coding agents as well as human reviewers. Second, instead of a complex model with many process and history features, we focus on a simple baseline that combines the dominant topic of a pull request with coarse file categories to estimate which files are likely to receive attention. Third, besides evaluating file ranking in terms of Hit@k, Precision@k, Recall@k, and nDCG@k, we also compare predicted and empirical attention distributions over changed files.

Our definition of hotspot files is related to work on defect prediction and effort aware quality assurance. Nam et al. proposed transfer defect learning, which adapts defect prediction models across projects by transforming the feature space [7]. Tan et al. studied online defect prediction for imbalanced data and evaluated techniques to handle severe class imbalance in change level prediction [10]. Zhang et al. showed that simple summation of class level metrics into file level aggregates can harm defect prediction performance and argued for more careful aggregation strategies [14]. These works share our goal of highlighting high risk or high value parts of a system so that testing and review resources can

be used more effectively. However, defect prediction usually operates across releases and uses future defects as labels, whereas our hotspot labels are defined within a single pull request using review comments and follow up commits. We study how attention is distributed across the files of a single pull request and which files become review hotspots, regardless of whether those files eventually contain defects.

Topic models are a standard tool when mining software repositories. Chen et al. surveyed the use of topic models in software engineering and documented applications such as analyzing mailing lists, categorizing bug reports, and supporting traceability and concept location [2]. Their survey shows that models such as latent Dirichlet allocation can uncover latent structure in heterogeneous software artifacts and that topic distributions can serve as features in downstream tasks. We follow the same general pattern of representing artifacts as documents and using topic distributions as features, but adapt it to pull requests and modern neural topic models. For each pull request, we build a document that combines the title and description, commit messages, review discussion, and diff hunks grouped by filename. We then apply BERTopic, which uses transformer based embeddings, clustering, and a class based TF–IDF procedure to derive topic representations [4]. From BERTopic we obtain a dominant topic and a topic mixture for each pull request. Our baseline model uses the dominant topic as a compact representation of the pull request intent and combines it with simple file categories derived from filenames.

Finally, our work builds on recent research on AI teammates in software engineering and the AIDev dataset in particular. Li et al. introduced AIDev as a large scale dataset of pull requests authored by autonomous coding agents on GitHub and used it to study how AI teammates reshape software engineering workflows [6]. The dataset contains rich metadata about pull requests, comments, and commits, and distinguishes between agent authored and human authored changes. We reuse AIDev as the basis for our study, but focus on a different question: rather than characterizing agent behavior at a high level, we examine how reviewers allocate attention across the files of an agent authored pull request, define file level hotspots using comments and follow up commits, and study how far a simple topic and file category based model can go in predicting those hotspots and attention patterns.

6 Conclusion and Future Work

In this work, we aim to identify file-level review hotspots in agentic pull requests by combining reproducible ground-truth construction with topic modelling–based prediction. We defined hotspots using observable review artifacts—inline comments and follow-up commits—and transformed these signals into per-PR hotspot and attention distributions. Building on this dataset and its derived ground truth, we developed a simple topic–file-category baseline that captures how different latent topics shape reviewer focus. Our empirical results show that this lightweight model achieves strong performance across Hit@k, nDCG@k, and distributional similarity metrics. The study thus contributes (i) a clear methodology and dataset for file-level review hotspot labelling, (ii) an interpretable baseline for hotspot prediction, and (iii) empirical insights into topic-driven patterns of reviewer behaviour in agentic PRs.

When extracting topics from PR metadata, one key BERTopic parameter controls the size of the topics produced. Using a larger size yields fewer, broader topics, which can become overly general and make it difficult to pinpoint the specific files likely to draw attention. Conversely, using a smaller size creates many fine-grained topics, which may overfit and cause the model to miss relevant hotspot files. A future work is to study this trade-off and determine the level of topic granularity that best supports accurate hotspot prediction.

Our prediction relies on the textual PR metadata only. Future work may explore richer modelling approaches. One possibility is using more extensive models that incorporate additional signals such as code semantics, historical reviewer behaviour, or PR size.

References

[1] Alberto Bacchelli and Christian Bird. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, page 712–721. IEEE Press, 2013.

[2] Tse-Hsun Peter Chen, Stephen Thomas, and Ahmed E. Hassan. A survey on the use of topic models when mining software repositories. *Empirical Software Engineering*, 21, 09 2015.

[3] Enrico Fregnan, Larissa Braz, Marco D'Ambros, Gül Çalıklı, and Alberto Bacchelli. First come first served: the impact of file position on code review. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE '22, page 483–494. ACM, November 2022.

[4] Maarten Grootendorst. Bertopic: Neural topic modeling with a class-based tf-idf procedure, 2022.

[5] Mik Kersten and Gail C. Murphy. Using task context to improve programmer productivity. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT '06/FSE-14, page 1–11, New York, NY, USA, 2006. Association for Computing Machinery.

[6] Hao Li, Haoxiang Zhang, and Ahmed E. Hassan. The rise of ai teammates in software engineering (se) 3.0: How autonomous coding agents are reshaping software engineering, 2025.

[7] Jaechang Nam, Sinno Pan, and Sunghun Kim. Transfer defect learning. pages 382–391, 05 2013.

[8] Doriane Olewicki, Sarra Habchi, and Bram Adams. An empirical study on code review activity prediction and its impact in practice. *Proceedings of the ACM on Software Engineering*, 1(FSE):2238–2260, July 2024.

[9] Peter C. Rigby and Christian Bird. Convergent contemporary software peer review practices. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, page 202–212, New York, NY, USA, 2013. Association for Computing Machinery.

[10] Ming Tan, Lin Tan, Sashank Dara, and Caleb Mayeux. Online defect prediction for imbalanced data. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 99–108, 2015.

[11] Patanamon Thongtanunam, Shane McIntosh, Ahmed E. Hassan, and Hajimu Iida. Review participation in modern code review: An empirical study of the android, qt, and openstack projects (journal-first abstract). In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 475–475, 2018.

[12] Lanxin Yang, Bohan Liu, Junyu Jia, Jinwei Xu, Junming Xue, He Zhang, and Alberto Bacchelli. Prioritizing code review requests to improve review efficiency: a simulation study. *Empirical Software Engineering*, 30, 11 2024.

[13] Lanxin Yang, Bohan Liu, Junyu Jia, Junming Xue, Jinwei Xu, Alberto Bacchelli, and He Zhang. Evaluating learning-to-rank models for prioritizing code review requests using process simulation. In *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 461–472, 2023.

[14] Feng Zhang, Ahmed E. Hassan, Shane McIntosh, and Ying Zou. The use of summation to aggregate software metrics hinders the performance of defect prediction models. *IEEE Transactions on Software Engineering*, 43(5):476–491, 2017.

A Use of AI-based Tools

AI-based tools are used in this study. BERTopic is a topic modeling tool that leverages transformers. In our study, BERTopic transforms PR metadata to embeddings, and cluster the embedding to extract topics among PRs.

LLMs are used to help writing boilerplate code for ground truth construction, topic modeling and evaluation. The correctness and behaviour of all LLM generated code are manually verified.