Interrogative Comments Posed by Review Comment Generators: An Empirical Study of Gerrit

Farshad Kazemi*, Maxime Lamothe†, Shane McIntosh*
* University of Waterloo, Canada, † Polytechnique Montréal, Canada
E-mail: {given_name}.{family_name}@{*uwaterloo,†polymtl}.ca

***Abstract*—*Background*: Review Comment Generators (RCGs) are models trained to automate code review tasks. Prior work shows that RCGs can generate review comments to initiate discussion threads; however, their ability to interact with author responses is unclear. This can be especially problematic if RCGs pose interrogative comments, i.e., comments that ask questions of other review participants.**

***Aims*: We set out to study the prevalence of RCG-generated interrogative code review comments, their similarity with the interrogative comments of humans, and the predictability of the generation of interrogative comments.**

***Method*: We study three task-specific RCGs and three RCGs based on Large Language Models (LLMs) on data from the Gerrit project using quantitative and qualitative methods.**

***Results*: We find that RCGs: (1) generate interrogative comments at a rate of 15.6%–65.26%; (2) differ from humans in generating such comments, which can stifle conversations if RCGs dissuade human reviewers from commenting deeply; and (3) produce interrogative comments with low predictability. Finally, we find that (4) the interrogative comments posed by LLM-based RCGs can differ even more substantially from human behaviour than those of task-specific RCGs. For example, the studied LLM-based RCGs pose rhetorical questions 3.16% of the time, whereas human-submitted interrogative comments pose rhetorical questions 8.74% of the time.**

***Conclusions*: Our results suggest that neither task-specific nor LLM-based RCGs can replace human reviewers yet; however, we note opportunities for synergies. For example, RCGs tend to raise pertinent questions about exception handling of common APIs more frequently than human reviewers. Putting greater emphasis on technical comments generated by RCGs (rather than conversational ones, such as interrogative ones) will likely improve their perceived usefulness.**

I. Introduction

Code review is a key quality assurance step in software development [1]. By focusing on the systematic inspection of source code [2], it plays a vital role in enhancing code quality [1], mitigating defects [3], ensuring maintainability [4], and facilitating knowledge sharing among developers [5].

Despite its benefits, code review is time consuming [6], [7] and error prone [8]. These limitations inspired the creation of *Review Comment Generator*s (RCGs), which can automatically generate code review comments [9], [10]. The goal of RCGs is to provide feedback that is more timely, consistent, and objective than what human reviewers can provide [11].

Although RCGs aim to emulate human reviewers in comment generation, they are not without limitations. For example, task-specific RCGs [12], [13], [14] may pose questions, but do not comprehend author responses, and hence, cannot follow up like human reviewers. While *Large Language Model*s (LLMs) can follow up on such author responses, their correctness is unclear. Incorrect responses can hinder useful discussions [15], leading to less productive code reviews. Indeed, we are not the first to question how RCGs respond in the wild [16], [17].

Prior work has shown that *interrogative* comments, i.e., comments that ask questions of other review participants (e.g., "why the regex?"[1]) are common outputs generated by RCGs [18]; however, a systematic study of these comments has not yet been conducted [19]. Our study aims to fill this gap by conducting quantitative and qualitative analyses of task-specific and LLM-based RCG-generated interrogative comments. We aim to *quantify* the prevalence and predictability of RCG-generated interrogative comments, and *characterize* them with respect to interrogative comments posed by humans. We conduct our study using 172,919 code review comments from the Gerrit project that were posted from 2018 to 2023.[2]

Quantitative Analyses (Section V). We find that a median of 15.61% of comments generated by task-specific RCGs are interrogative, whereas 65.26% and 47.67% of DBRX and LLaMA2 comments (i.e., LLM-based RCGs) are interrogative, respectively. Comparatively, 39.91% of human-submitted comments in our corpus are interrogative, suggesting that task-specific RCGs do not pose questions frequently enough, and that LLM-based RCGs may be overcorrecting. We also study whether RCGs and humans ask questions about the same code changes. Fisher's exact test suggests that, indeed, there is a statistically significant association among RCG-generated and human-submitted comments in their mood, i.e., whether the comment is declarative or interrogative. Finally, we find that RCG behaviour in code changes with discussion threads that start with interrogative comments is unpredictable and erratic, i.e., the rate of RCG-generated interrogative comments varies more across the studied *Merge Request*s (MRs).

Qualitative Analysis (Section VI). Through a manual inspection of samples of RCG-generated and human-submitted interogative comments, we observe that a considerably larger share of questions focus on the rationale for code changes when RCGs generate questions rather than humans (13.86%–22.11% vs. 1.94%). Humans discuss logical code flow more often (54.37% vs. RCGs

[1]https://gerrit-review.googlesource.com/c/gerrit/+/40230?tab=comments
[2]https://gerrit-review.googlesource.com/

38.64% to 42.17%) and predominantly use questions for suggestions (63.11%), whereas RCGs tend to request additional context (56.84%–84.09%). Furthermore, humans often employ rhetorical questions (8.74%) and hypotheticals (4.85%), whereas task-specific RCGs do not. Our LLM-based RCG experiments show that LLMs outperform task-specific RCGs in hypothetical inquiries (12.65% for GPT-4), but lag in rhetorical questioning (2.27% for LLaMA2).

We conclude that RCGs could aid the review process by focusing on technical comments, like those related to exception handling. Meanwhile, their interrogative review comments should be deprioritized because they exhibit a disparity with respect to human behaviour (shown by differences in entropy and density) and their limited linguistic ability impacts comment quality. Although LLM-based RCGs show promise in addressing the linguistic limitations of task-specific RCGs, they do not consistently improve performance across different types of interrogative comments. We thus recommend using RCGs for complementary technical comments where their benefits can be harnessed while mitigating their shortcomings.

II. Background and Related Work

Below, we present related research on automatic code review and code review comment generation to position our study and reason about our experimental design choices.

Automatic Code Review. Previous studies [11], [13] defined three primary tasks for code review automation: (1) code quality estimation [20], [21], [22]; (2) revising code after [23], [24] or before [11], [12], [25], [26], [27] the review; and (3) code review comment generation. Machine learning approaches have been tailored to each of these, including code review comment generation. Indeed, Zhou *et al.* [18] found that the CodeT5 [28] general model surpassed the best RCG in code revision by 13.4%–38.9% and the T5-review [12] RCG model was the best for code review comment generation.

Although task-specific RCGs can emulate human behaviour to some extent [29], it remains unclear whether their interrogative comments promote discussion or inadvertently hinder it. Zhou *et al.* [18] noted that interrogative comments are common in human code review (over 33.8% of comments) [18], and advocated for having models that emulate this behaviour; however, the ability of RCGs to generate such comments and their synergies with human comments has yet to be studied. We therefore aim to study the generation of interrogative code review comments for task-specific and LLM-based RCGs to better understand their roles in the code review process.

LLMs have become a resource for code review [30], [31]. Indeed, ChatGPT has been used for various software engineering tasks, including code review, finding that ChatGPT's responses aligned with human reviewers in only 4 out of 10 tested instances [32]. Further, Guo *et al.* [33] examined ChatGPT's revision effectiveness and its strengths and weaknesses in post-review code refinement. Their findings indicate that despite higher operational costs, ChatGPT underperformed with respect to CodeReviewer [13] in one of two studied datasets. Sun *et al.* [34] introduced a feedback loop to enhance LLM-based code reviews; however, users complained about slow response times. Rasheed *et al.* [35] studied the automation of code review using multi-agent LLM-based RCGs. These findings suggest a complex tradeoff between task-specific and LLM-based RCGs. Since neither type of RCG always outperforms the other, we compare both types in our study.

We investigate how RCGs generate interrogative comments, focusing on three state-of-the-art RCGs (AUGER [14], CodeBert [36], and CodeReviewer [13]), and three LLM-based RCGs (DBRX,[3] GPT-4,[4] and LLaMA2 [37]).

Code Review Comment Generation aims to emulate human reviewers and automatically generate review comments for a given code change. The goal is to minimize the workload for reviewers and the delay for authors to receive feedback. Many such approaches exist. For example, Review Bot [38], which produces code review comments derived from the outputs of various static analyzers, received approval on 93% of its generated comments. DeepCodeReviewer [39] and CORE [40] leverage deep learning to recommend and automate reviews. CommentFinder [9] addresses the latency in deep learning methods using an information retrieval approach.

Recent advances shifted focus to task-specific RCG models [27]. This trend typically involves pre-trained models [12], [13], [14], [41], [42] that combine natural and programming language processes to enhance the code review process. With the emergence of LLMs, studies have also examined their utility in automating code review tasks. They were found to have a variety of limitations [17], e.g., a significant gap between perceived and actual effectiveness [16], issues in performing like humans [29], and drawing out code reviews without providing a commensurate increase in code quality [43]. Moreover, despite LLaMA-Reviewer's [44] higher resource demand for inference, it did not consistently outperform state-of-the-art task-specific RCGs like CodeReviewer [13]. Similarly, Pornprasit and Tantithamthavorn [45] assessed GPT-3.5's capabilities for code review, observing that state-of-the-art task-specific models still tended to outperform GPT-3.5. Given these findings and the high operational costs of LLMs, we choose to consider three high-performing [18] task-specific RCG and two types of LLMs for each of our studies.

III. Model Selection

We select six models of varying types for our study. AUGER, CodeBert, and CodeReviewer are task-specific RCGs, selected because of their strong performance in prior work [18]. GPT-4 Turbo[4] is an exemplar of enterprise models, known for their performance across various tasks, albeit at higher costs. Due to this cost, we select DBRX-instruct[3] and LLaMA2-7B [37] as freely available alternative LLMs for our quantitative analysis. We describe each model below.

AUGER [14] is an RCG that uses the pre-trained CodeTrans T5 model [28], [46], which was further fine-tuned on ∼10K

[3]https://github.com/databricks/dbrx

[4]https://bit.ly/open-ai-gpt-4

code review instances from 11 Java projects. AUGER thus can leverage its training data to provide review context. AUGER was assessed with a survey that revealed that 29% of developers found its generated comments useful [14].

CodeBert [36] has a transformer-based architecture [27] and is trained with Masked Language Modelling (MLM) and replaced token detection using NL-PL pairs and unimodal code data. This approach allows CodeBert to excel in tasks like code search and documentation generation. Like Zhou *et al.* [18], we use the pre-trained CodeBert model, fine-tuning it with ∼50K review records. Fine-tuning this model helps with the generation of the comments instead of code, leading to more understandable generated comments.

CodeReviewer [13] employs CodeT5 [47] and further fine-tunes it on data in the form of `<comment, code hunk>` to process code diffs as input. Compared to AUGER, CodeReviewer focuses on understanding code changes and their relationship to review comments because its output explicitly highlights line additions and deletions. Indeed, CodeReviewer has been shown to perform comparatively well in terms of comment generation among RCGs [44].

When it was introduced in March 2024, *DBRX*[3] tended to outperform competing open models and established models like GPT-3.5, particularly in code-related tasks. The instruct version is optimized for scenarios like programming and coding assistance. Models like GPT-4 Turbo outperform DBRX, but they are more expensive to run. We use the instruct version of the model to achieve the best possible performance at a manageable price when GPT-4 Turbo would be prohibitive.

GPT-4 Turbo[4] upgraded GPT-4 capabilities, including support for more diverse inputs. As a proprietary model, its architecture details are not publicly disclosed, and to interact with the model, developers should use OpenAI's API.

LLaMA2 [37] offers 7, 13, and 70 billion parameter configurations. These models, trained on a dataset spanning 2 trillion tokens from January to July 2023, follow the standard transformer architecture with an auto-regressive model design.

IV. Dataset Preparation

In this section, we describe how we create the dataset for our analysis. Figure 1 shows the steps which consists of: (1) Data Collection, (2) Data Cleaning, and (3) Review Generation components. Below, we describe each component.

1. Data Collection. Our study aims to use RCGs trained on human code reviews to generate code review comments, allowing us to analyze the prevalence and patterns of questions that are generated. To that end, we must first select a development community that produces a large number of high-quality human-submitted code reviews. Human-submitted code reviews are necessary both to fine-tune RCGs and to compare their results against a baseline. We choose the Gerrit community for our study because it provides us with the opportunity to analyze how RCGs perform in a high-quality case. The Gerrit community uses Git, contains a large amount of review data (1,852 code reviews and 15,000 code review comments in 2022), has contributors representing organizations of influence

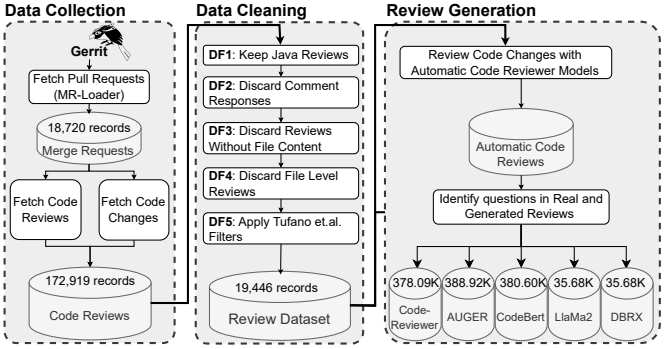

Fig. 1. The overview of the data preparation procedure.

(e.g., Google, Cisco, and Spotify), and tends to provide review records that are well linked to the commits on which they were performed. Focusing on the Gerrit community does pose clear risks to external validity but allows control over internal validity threats, such as understanding of data linkage properties, highlighting a subjective trade-off between these validity types among researchers as studied previously [48]. Furthermore, the Gerrit community has been the subject of previous studies [49] as the its maintainers are committed to best practices for code review, and as such, its review comments are of high quality and quantity [49], [50].

To start our data collection, we use MR-Loader[5] to obtain the raw MR data from the Gerrit community. As the primary quality gate through which all code changes of the community flow, MRs are a prime source of data for our study. Indeed, all the details related to a change, including the code review comments, can be accessed through MRs. We also collect metadata, such as the author and committer of each code change, to use in our data analysis.

We collect this data from the Gerrit community from the first available MR (2018-10-21) for a span of five years (until 2023-08-22). While this filters more recent data from our dataset, it improves the odds that all collected MRs are finalized. For this work, we focus on the core community project—the Gerrit code collaboration tool. We filter out MRs that are not related to that core project. This produces a dataset of 172,919 code review comments, which we complement[6] with the content of the files under discussion before a change and the code diff on which the reviewers commented.

2. Data Cleaning. After data collection, we apply five *Data Filter*s (DFs) to mitigate noise (e.g., comments like "lgtm"). We describe each DF below.

DF1: We filter out comments on files other than Java files by identifying files with the `.java` extension. We concentrate on Java files because two of the three studied *Review Comment Generator*s (RCGs) are trained on Java code review data and provide feedback only on Java code changes. 77,529 code review comments survive this filter.

[5]https://github.com/JetBrains-Research/MR-loader

[6]Using its API: https://gerrit-review.googlesource.com/Documentation/

DF2: We filter out response comments, i.e., comments posted in reply to another comment, using a Gerrit API flag that indicates whether a comment is in reply to another comment. We remove response comments because non-LLM-based RCGs only review code changes and do not engage in follow-up discussions. Similar filters are common in previous studies [12], [14] to prepare the data for training and inference. After **DF2**, our dataset contains 37,656 code review comments.

DF3: We filter out records that have irretrievable file content before the code change. For each code review comment, we use a Gerrit API request to retrieve related file content. We remove the few cases where the API request is unsuccessful because the data cannot be processed by the studied RCGs. After **DF3**, our dataset contains 37,625 code review comments.

DF4: We filter out file-level comments, i.e. those that are not connected to any line of code, but are instead about a higher-level concept in that file.[7] We do this by identifying comments that mention a file name, but no comment lines. We remove these records because task-specific RCGs review functions, sub-function blocks, or lines of code. Therefore, these file-level comments cannot be produced by the studied RCGs. 37,133 code review comments survive this filter.

DF5: Our final filter removes empty comments after pre-processing. We preprocess comments using the replication packages from Tufano et al. [12]. Specifically, we use their `code/Analyzer.py` and `code/Cleaner.py` files. This filters out non-English comments and comments on code portions that are not part of a function, such as comments on imports. This filter also removes empty comments after the removal of emojis and links. In the end, 19,446 code review comment records survive this filter.

Our dataset, scripts, and the resulting 19,446 code review records are available in our replication package.[8]

3. Review Generation. We use our collected dataset and our selected RCGs to generate code review comments based on the given code changes. Task-specific RCGs are trained to excel in comment generation with a specific input format, but have not undergone Reinforcement Learning from Human Feedback (RLHF), a technique used to align intelligent agents with human preferences. Conversely, LLMs are more generalized models that interact through natural language interfaces, but typically require more resources. Below, we elaborate on each type of RCG in more detail.

Task-specific RCGs. The selected task-specific RCGs differ in their input format. Both AUGER and CodeReviewer adopted the Text-To-Text-Transfer Transformer (T5) model [28], but each chooses one of its many existing variations [51], which require different inputs. AUGER's input requires a function that contains commented code, allowing it to use this context to comment on issues at the function level [14]. Conversely, CodeReviewer uses inputs in the form of code diffs. Thus, while we follow prior work [44] and use off-the-shelf versions of AUGER [14] and CodeReviewer [13], we tailor the inputs to each model. Unlike CodeReviewer and AUGER, CodeBert, as a general-purpose model, can ingest a code change as-is; however, it must be further fine-tuned [12]. Thus, we use the pre-trained CodeBert model and train it for another 50,000 epochs on the Tufano *et al.* dataset [12] to perform code reviews. To prevent data leakage from fine-tuning to testing phase, we remove the duplicate entries when evaluating this model. Furthermore, to minimize implementation errors, we use the code provided by Zhou *et al.* [18] and only modify it to add top-k sampling [52].

Our diverse selection of RCGs allows us to generate a wide range of code review comments. Because our chosen models are non-deterministic, we allow each model to provide multiple outputs (denoted by k) for a given input. Similar to prior work [13], [14], we run experiments for all of our studied RCGs for the top-k results, where $k = \{1, 3, 6, 10\}$.

We apply a rule-based heuristic to detect interrogative moods in comments, building on Zhou *et al.* [18]. To improve accuracy, we first use NLTK[9] to split review comments into sentences, then identify interrogative ones by checking for question-indicative keywords (e.g., what) and verifying their role as interrogative adverbs. To verify the performance of this heuristic, we draw a random sample of 377 comments and inspect the output of the heuristic for each one manually. The heuristic yields an AUC of 0.96 for human review comments (95% confidence level, 5% margin of error). Using this heuristic, we individually classify both the actual comments and each of the top-k generated code review comments of each of the studied RCGs. After collecting and classifying the model outputs as interrogative or declarative, we analyze them to address our RQs.

Large Language Models. For code review comment, each LLM is run with the $k = 1$ setting, is prompted to adopt the persona of a code reviewer [53], and leverages two-shot prompting [45], [54] to enhance model efficacy. We provide two examples—one interrogative comment and one declarative comment—to prevent model bias and pose questions only when necessary. These examples are selected from a randomly chosen set of code reviews and inspected to ensure that the issues raised in the comments are confined to the code change and do not reference materials outside the scope of the change. We choose not to fine-tune the studied LLMs because they have already been trained on extensive code review data. The input comprises a code hunk, similar to CodeReviewer [13], which is known for its superior performance in comment generation among RCGs [44]. After prompting the LLM, we record its response for further analysis.

V. Quantitative Analyses

In this section, we quantitatively study interrogative comments generated by RCGs during code reviews. Below, we describe our approach to data analysis and then present the results with respect to a set of observations.

[7]Sample comment: https://gerrit-review.googlesource.com/c/gerrit/+/351075/comment/2408abef_e5ea576c/

[8]https://doi.org/10.5281/zenodo.13301078

[9]https://www.nltk.org/

Approach. We first evaluate the prevalence of RCG-generated interrogative comments by measuring their frequency in our studied dataset. To measure the similarity of interrogative comments generated by RCGs and humans, we calculate the density of interrogative comments per MR for RCGs and human reviewers, i.e., the rate at which interrogative comments are posed in each MR. We report the results of our analysis using top-k values ($k \in 1, 3, 6, 10$ for task-specific and $k = 1$ for LLM-based models) for each of our studied task-specific RCGs to explore the impact of k on their behaviour. We test various k settings because higher k settings improve the quality of comments by providing a more lax guess budget and increase the diversity of suggestions [52]. Experimenting with k settings is also common in prior work [12], [14].

We also explore whether RCGs imitate human behaviour when posing interrogative comments. To evaluate the association between comments generated by RCGs and those posed by humans, we apply Fisher's exact test [55]. Additionally, to compare the density of interrogative comments generated by RCGs and humans across MRs, we apply Mann-Whitney U test [56]. We also apply Shannon's entropy [57] to measure the predictability of RCG behaviour.

To evaluate the prevalence of LLM-generated interrogative comments, we focus on the $k = 1$ setting because larger k settings generate larger computational costs. Similar to the evaluation of task-specific RCGs, we study the association between the interrogative mood of comments produced by humans and LLMs. We use Fisher's exact test with $\alpha = 0.0036$—a setting of $\alpha = 0.05$ that has been corrected for multiple comparisons using the Bonferroni method [58].

Discussion-Inducing Changes: Discussion-inducing code changes in the context of RCG-generated interrogative comments are of interest because RCGs have the potential to disrupt valuable code review discussions.[10] We analyze code review comments with discussion threads that have at least one response because these discussions are (1) closely linked to questions during the code review process [19], and (2) play an essential role in knowledge sharing and design conversations within the code review process [2], [15], [59].

Two scenarios may arise in discussion threads. First, RCGs may ask questions in a similar fashion to humans, sparking productive discussions that require active RCG engagement. This scenario would require RCGs with a deep understanding of the code and strong reasoning capabilities, which are currently lacking. Second, RCGs may ask different (perhaps even incorrect) questions, potentially stifling informative discussions. This scenario is more daunting, as it would hinder valuable project documentation [2]. We therefore study how task-specific RCGs compare to humans in such cases and whether LLMs can be applied to address their shortcomings.

Identifying Discussion-Inducing Changes: To identify these code changes, we first extract comment threads, specifically those with more than one reply from the code review records. During this process, we filter out single-word responses to focus on informative discussions related to code changes, excluding minimal responses like "Acknowledged."[11] We then flag the corresponding code and the initial comment that initiated the changes as discussion-inducing.

Results. Tables I and II show the prevalence of interrogative comments and the predictability of our studied RCG behaviour in generating them. Below, we present our observations. In this section, we use DBRX instead of GPT-4 to reduce costs.

How Often Do Review Comment Generators Generate Interrogative Comments? *15.61% of comments generated by the task-specific RCGs are interrogative.* Interrogative review comments account for 0.91% to 27.54% of all generated comments, with medians of 15.78%, 19.50%, and 10.15% for AUGER, CodeReviewer, and CodeBert, respectively. The overall median across all values of k is 15.61%. Regarding the LLM-based RCGs, 65.26% and 47.67% of the DBRX and LLaMA2-generated review comments are interrogative, respectively. To provide a benchmark, our dataset of human review comments from the Gerrit project contains interrogative comments at a median rate of 39.91%.

Among the studied task-specific RCGs, AUGER has the highest interrogative comment frequency at 15.30% for $k = 1$, with CodeReviewer and CodeBert at 4.11% and 0.91%. These rates tend to increase as k increases, indicating that these models put more weight on non-interrogative comments.

CodeBert's relatively low rate of interrogative comments, especially for $k = 1$, can be attributed to its base model, which is primarily trained for generating code rather than natural language. Although pre-training CodeBert for 50K epochs enhances comment quality for larger k settings, when focusing only on the most likely solution ($k = 1$), the generated comments mainly consist of code fragments with limited natural language text. Consequently, only a few would have questions within the CodeBert-generated comments.

LLM-based RCGs pose questions more frequently than task-specific models, especially in the case of DBRX. We suspect that this is due to their limited contextual knowledge about the given change hunks, which lead to confusion. We explore the questions that LLMs pose in more detail in Section VI.

When we repeat the previous analysis focusing exclusively on discussion-inducing code changes, we find that the rate of interrogative comments ranges from 0.91% to 28.02%, with a median of 15.60% for all comments generated by task-specific RCGs. This indicates a slight increase in rates compared to our assessment of all comments. As for LLM-based RCGs, DBRX and LLaMA2 generate interrogative comments 67.62% and 46.64% of the time, indicating a +2.36 and -1.03 percentage point difference. As a benchmark, human-submitted comments in discussion-inducing code changes increased by 18.25 percentage points to reach 58.16% when all comments are considered. This result indicates that RCGs do not treat discussion-inducing changes like humans.

[10] An example of a discussion-inducing comment: https://gerrit-review.googlesource.com/c/gerrit/+/13126/1/gerrit-sshd/src/main/java/com/google/gerrit/sshd/commands/Receive.java#192

[11] https://gerrit-review.googlesource.com/c/gerrit/+/430619/comments/c74c5adb_ebc49a32

TABLE I
RATE OF GENERATED INTERROGATIVE COMMENTS FOR STUDIED REVIEW COMMENT GENERATORS (RCGS).

RCG \ Type	All	Discussion-inducing	k
AUGER	15.30%	15.52%	1
	16.14%	16.31%	3
	15.96%	15.89%	6
	15.60%	15.64%	10
CodeBert	0.91%	0.91%	1
	6.08%	6.22%	3
	14.21%	14.85%	6
	19.61%	20.57%	10
CodeReviewer	4.12%	4.18%	1
	15.62%	15.57%	3
	23.38%	24.00%	6
	27.54%	28.02%	10
DBRX	65.26%	67.62%	1
LLaMA2	47.67%	46.64%	1
Baseline/Human	*39.91%*	*58.16%*	N/A

Task-specific RCGs generate interrogative comments a median of 15.61% of the time and 65.26% and 47.67% for DBRX and LLaMA2-based RCGs. Since task-specific RCGs rarely ask questions and do not follow up, this limits discussion and reduces the effectiveness of code reviews.

Do Review Comment Generators and Human Reviewers Raise Interrogative Comments for Similar Code Changes? To shed light on RCG behaviour, we study the interrogative comments of RCGs and humans. Fisher's exact test [55] is used to check for nonrandom associations between the interrogative mood of RCG-generated and human comments. We repeat this for each k setting and apply the Bonferroni method [58] to correct for multiple comparisons.

CodeReviewer, for k settings of 3, 6, and 10, and CodeBert for k settings 6 and 10, show significant associations with human rates of interrogative comments, whereas AUGER shows no association. Table III presents p-values and odds ratios from this test, revealing that for two of the three studied task-specific RCGs, there is an association between generated and human comments in terms of sentence mood for different k settings. In all but one case, the significant associations have small odds ratios that are greater than one. This indicates that even when RCG-generated and human-submitted comment moods are correlated, the relationship is weak.

Unlike LLaMA2, the mood of DBRX-generated comments tends to correlate with that of human comments. Although enterprise LLM-based RCGs may pose questions in similar locations as humans, the similarity between those questions may determine whether these models should complement or replace human reviewers. We explore this in Section VI.

TABLE II
ENTROPY OF THE DENSITY OF INTERROGATIVE COMMENTS FOR ALL AND DISCUSSION-INDUCING CODE CHANGES.

RCG \ Type	All		Discussion-inducing		k
	entropy	p-value	entropy	p-value	
AUGER	0.297	0.00	0.337	<.001	1.0
	0.431	<0.001	0.456	<0.001	3.0
	0.520	<0.001	0.546	<0.001	6.0
	0.578	<0.001	0.593	<0.001	10.0
CodeBert	0.068	0.00	0.073	0.00	1.0
	0.353	0.00	0.389	0.00	3.0
	0.553	<0.001	0.576	<0.001	6.0
	0.641	<0.001	0.671	<0.001	10.0
CodeReviewer	0.169	0.00	0.154	0.00	1.0
	0.470	<0.001	0.497	<0.001	3.0
	0.613	<0.001	0.641	<0.001	6.0
	0.673	<0.001	0.707	<0.001	10.0
DBRX	0.555	<0.001	0.567	<0.001	1.0
LLaMA2	0.402	<0.001	0.399	<0.001	1.0

The association of comment mood between human-submitted and RCG-generated comments diminishes when only considering discussion-inducing changes. Table III shows that, compared to the same setting for all comments, either the significant association is lost or the odds ratio decreases. Therefore, comment moods differ more among discussion-inducing comments than among other review comments.

A Mann-Whitney U test shows significant differences in distributions between RCG and human interrogative comments in all our experiments. Contrasting this observation with the comment moods, it appears that RCGs do behave differently than humans when generating interrogative comments.

When we repeat this test for discussion-inducing code changes specifically, we find that $p = 0.8852$, indicating that the null hypothesis (i.e., both samples are drawn from the same distribution) cannot be rejected. Based on this observation, RCGs appear to treat discussion-inducing code changes no differently than other changes, leading users to potentially miss out on useful discussion in code review.

None of our studied RCGs generate interrogative comments like humans. RCG-generated and human-submitted interrogative comments differ in terms of frequency of interrogative comment and their density per MR. Moreover, RCGs treat discussion-inducing changes similar to other changes, potentially diminishing the depth of review discussions.

When Do Review Comment Generators Generate Interrogative Comments? Given that RCG-generated interrogative comments differ from humans ones, we compute the entropy of normalized interrogative comments per MR, i.e., count of interrogative comments over the top-k, as a measure of predictability for task-specific RCGs. A higher entropy indicates that the number of RCG-generated interrogative comments

TABLE III

ODDS RATIOS AND FISHER'S EXACT TEST P-VALUES FOR REVIEW COMMENT GENERATORS (RCGS). A CORRECTED P-VALUE BELOW 0.0018 (*) INDICATES SIGNIFICANCE. RATIOS > 1 OR < 1 IMPLY POSITIVE OR NEGATIVE ASSOCIATIONS, RESPECTIVELY, WITH SIGNIFICANT ONES IN BOLD.

	odds ratios								p-values							
	All comments				Discussion-Inducing comments				All comments				Discussion-Inducing comments			
	top-1	top-3	top-6	top-10	top-1	top-3	top-6	top-10	top-1	top-3	top-6	top-10	top-1	top-3	top-6	top-10
AUGER	1.0178	1.0001	1.0020	1.0038	1.0762	1.0153	1.0367	1.0378	6.6931e-01	1.0000e+00	9.0257e-01	7.6428e-01	2.7679e-01	6.9533e-01	1.7836e-01	7.5635e-02
CodeBert	1.1597	1.0564	**1.1150**	**1.0716**	1.1979	**1.1031**	**1.0973**	**1.0550**	5.4031e-02	1.9305e-03	**3.5884e-37***	**1.9010e-32***	1.8091e-01	**7.7971-04***	**3.2262e-11***	**1.7494e-08***
CodeReviewer	1.0061	**0.9613**	**1.0546**	**1.0270**	1.0122	**0.9296**	1.0165	1.0187	8.6718e-01	**8.5961e-04***	**1.1110e-13***	**3.9885e-07***	8.5674e-01	**1.5052e-04***	1.5752e-01	3.0259e-02
DBRX-Based	**1.1366**	-	-	-	1.1249	-	-	-	**2.26e-08***	-	-	-	0.0024	-	-	-
LLaMA2-Based	0.9653	-	-	-	0.9873	-	-	-	0.1074	-	-	-	0.7305	-	-	-

varies more from one MR to another, making their overall behaviour less predictable. Table II presents the normalized entropy of interrogative comment density using Equation 1:

$$\text{Normalized Entropy} = \frac{-\sum_{i=1}^{n} rate_i \times \log_2 rate_i}{\log_2(\text{number of MRs})} \quad (1)$$

Table II presents the p-values obtained from the Mann-Whitney U test to explore whether the distribution of generated interrogative comments is different for RCGs and humans.

In Table II, we observe that the p-value is always less than the corrected $\alpha = 0.0018$, indicating that human-submitted and RCG-generated interrogative comments have different predictability. Moreover, we notice that an increase in k results in higher entropy for RCG-generated interrogative comments. Prior studies suggest that larger k settings yield better performance [12], [14]; however, in our case, it appears that it also makes generating interrogative comments less predictable.

We investigate the predictability of RCGs for discussion-inducing changes. Higher entropy in interrogative comments (shown in Table II) suggests less predictability in generating them, making it challenging to predict RCG behaviour. Plots that support these observations are available in our appendix.[8]

Considering larger top-k suggestions increases the entropy of interrogative comments in task-specific RCGs, making it increasingly harder to speculate about the RCG behaviour. Thus, RCGs not only pose questions with which authors might not be able to interact, but they also pose them with patterns that differ from humans. Additionally, while predicting the behaviour of task-specific Review Comment Generator*s (RCGs) in generating interrogative comments for discussion-inducing code changes is more important than normal code changes, a higher entropy shows that it is, in fact, more challenging.*

VI. QUALITATIVE ANALYSES

In this section, we analyze the content of interrogative comments and categorize them to shed light on the question-posing behaviour of RCGs with respect to human behaviour.

Approach. We adopt a catalogue of comment categories from prior work [19], [60] to compare the types and intentions of interrogative code review comments generated by RCGs with those of human reviewers. The catalogue is derived from two sources—one for comment types [60] and one for interrogative comment intentions [19]. The white rows of Table IV present the twelve comment categories proposed by Ochodek *et al.* [60] to describe different types of code review comments, whereas the grey rows show the three new comment categories that emerged from our dataset, but did not exist in the categories proposed by Ochodek *et al.* [60]. Similarly, Table V presents the five primary intents behind review queries that were identified by Ebert *et al.* [19]. We leverage the types of questions posed by RCGs to further uncover how RCGs can aid in code review.

Since inspecting all available interrogative comments is impractical, we draw a random sample containing both groups of task-specific RCG- and human-submitted code review comments. We then apply blended coding [61] to the sample, initially using established categories for comment types [60] and intentions [19], while also integrating new categories to capture emergent trends. The following sections detail our sampling method and the blended coding approach.

Sampling: Similar to prior studies [12], [14], our analyses focus on initial review comments and exclude subsequent responses in the review threads. Given our goal of identifying the types and intentions of interrogative comments, our attention is narrowed to questions posed by either human reviewers or RCGs. In our dataset, we categorize reviews with interrogative comments into three groups based on the questioners: (1) only human reviewers, (2) only task-specific RCGs, or (3) both human reviewers and task-specific RCGs. We randomly select samples from each category and merge these into a composite sample set to ensure representation from all types of review comments. We then use GPT-4- and LLaMA2-based RCGs to generate LLM responses for the sampled set. Moreover, to ensure that LLMs have not been exposed to these code changes during their training, we augment this set with 30 additional changes that were introduced after the cutoff date for the training period of the studied LLMs.

Blended Coding: We use a blended coding approach [61] to label the sampled set of interrogative comments. This strategy allows us to leverage categories from prior studies while retaining the flexibility to create new categories.

Two authors (i.e., coders) inspect each entry in the sample, focusing on interrogative review comments by both reviewers and RCGs. The coders label false positives (i.e., non-interrogative comments) as a separate class. For consistency, in multi-question comments, the coders only label the first question. As a first step, in a preliminary session, for the task-

TABLE IV
CODE REVIEW COMMENT TYPES (OCHODEK *et al.* [60]), WITH THE INCLUSION OF NEW TYPES IDENTIFIED IN OUR ANALYSIS (IN GRAY).

Comment Type	Description
code design	Comments related to the structural organization of the code (e.g., class design).
code style	Comments pertaining to the code's layout and readability.
code naming	Comments focusing on the conventions used for naming variables, functions, classes, etc.
code logic	Comments that discuss the logic and operations within the code, such as algorithms.
code data	Comments that address the handling and usage of data (e.g., variables) within the code.
code api	Comments on the use and evolution of *Application Programming Interface*s (APIs) within the codebase.
code doc	Comments that concern documentation and commentary in the source code.
compatibility	Comments related to compatibility with operating systems, tools, and various versions.
config/.../review	Comments about the process of submitting and reviewing patches and commits.
code purpose	Comments about the necessity for changes, typically to clarify the intention behind code segments.
code exception	Comments related to exception handling within the code.
code testing	Comments related to existing tests or the need for new tests.

TABLE V
CODE REVIEW COMMENT INTENTIONS (EBERT *et al.* [19]).

Intention	Description
Suggestions	Inquiries that subtly propose a course of action.
Requests	Questions seeking details such as explanation related to the code under review.
Hypothetical Scenarios	Questions that construct a potential situation which may not have been previously considered.
Rhetorical Questions	Questions paired immediately with their answers, serving to emphasize a point.

specific RCGs, the coders label 100 interrogative comments, separate from the sampled set, for the initial categories. For LLM-based RCGs, coders label only 50 samples to refine guidelines, as these comments are more structured due to LLM outputs. Then, each coder independently labels the remaining comments in batches of 50, alternating between human and RCG comments. In addition to the comment content, coders use context, such as the referenced code changes, the type of RCG, and the responses to human comments. After each batch, the coders met to resolve discrepancies and refine the emerging categories. For disagreements, coders explain their rationale and reach a consensus, with the last author as arbitrator. While a fixed number of examples may yield reliable confidence intervals for binary tasks, saturation is more appropriate for multi-label problems [62]. Thus, coding continues until reaching our saturation [63], [64] criterion, i.e., no new codes identified across two consecutive batches. This criterion is met after labeling 150 samples.

To assess the reliability of the coding task, we measure inter-rater reliability using Cohen's Kappa for both distinct coding tasks for the four sources of comment generation, i.e., categorizing the type and intention of generated comments by task-specific RCGs, the GPT-based RCG, the LLaMA2-based RCG, as well as the human reviewers. For task pairs of comment type and interrogative intention, we obtain Kappa score pairs of (0.49, 0.45), (0.67, 0.66), (0.60, 0.50), and (0.61, 0.43) for task-specific, GPT-4, LLaMA2, and human reviewers, respectively. These scores reflect substantial agreement on comment type and thread response decisions for both LLMs. The scores for comment type tasks show moderate agreement for task-specific RCGs and substantial agreement for the other two RCGs [65]. For interrogative comments, the scores for human-submitted comments indicates substantial agreement, whereas the other RCGs have moderate agreement. Lower Kappa scores are likely due to the breadth of labels and noise from hard-to-parse generated comments.

TABLE VI
GENERATED COMMENT TYPES FOR INTERROGATIVE COMMENTS.

Comment Type	Human	RCGs	GPT-4	LLaMA2
code logic	54.37%	40.00%	42.17%	38.64%
code design	14.56%	7.37%	7.23%	6.82%
code naming	9.71%	8.42%	3.01%	2.27%
code data	3.88%	5.26%	4.82%	6.82%
code testing	3.88%	1.05%	3.61%	0.00%
code api	2.91%	2.11%	3.61%	0.00%
code exceptions	2.91%	7.37%	14.46%	20.45%
config/../review	1.94%	3.16%	0.60%	0.00%
code purpose	1.94%	22.11%	13.86%	15.91%
code style	1.94%	3.16%	3.61%	4.55%
code doc	0.97%	0.00%	2.41%	4.55%
compatibility	0.97%	0.00%	0.60%	0.00%

Results. Tables VI and VII summarize the results of the coding task. We discuss these observations below.

What Types of Comments Can Be Observed Within the Scope of Generated Interrogative Comments? Table IV highlights three new emergent categories of comment types (cells with a gray background). Conversely, our analysis does not reveal any examples of types *code io*, *code doc*, *compatibility*, *rule def*, and *config building/installing*.

TABLE VII
DISTRIBUTION OF QUESTION INTENTIONS FOR GENERATED INTERROGATIVE COMMENTS.

Comment Intention	Human	RCGs	GPT-4	LLaMA2
Suggestions	63.11%	40.00%	14.46%	13.64%
Requests	23.30%	56.84%	72.89%	84.09%
Rhetorical questions	8.74%	3.16%	0.00%	2.27%
Hypothetical scenario	4.85%	0.00%	12.65%	0.00%

LLM-based RCGs pose more documentation-related questions (2.41% and 4.55% of all generated interrogative comments for GPT-4 and LLaMA2, respectively) than both human reviewers (0.97% of all human-submitted interrogative comments) and task-specific RCGs, which do not ask this type of question. This suggests that LLM-based RCGs could serve as vigilant overseers for code documentation quality.

RCGs pose more questions about exceptions than human reviewers. While task-specific RCGs are adept at generating such interrogative comments (7.37% of their generated interrogative comments), LLMs ask about exceptions more frequently (14.46% and 20.45% for GPT-4 and LLaMA2). These RCGs appear especially adept at raising exception-handling concerns for common APIs, such as file I/O operations, whereas human reviewers excel at pinpointing complex issues, such as neglected edge cases. This findings aligns with recent studies on LLM-based code review performance [34].

Human reviewers more frequently question the logic behind code changes (54.37%), such as conditional placement or potential oversights in handling edge cases, compared to RCGs (38.64%–42.17%). In contrast, RCG- (22.11%) and LLM-posed (13.86% and 15.91% for GPT-4 and LLaMA2, respectively) questions often concentrate on the purpose of the changes, possibly reflecting their more limited grasp of the broader context of the code. Human reviewers are often among the core developers of the projects [5] or have past involvement with the modified files and subsystem in the code change [66]. By using this prior context, they can provide more in-depth interrogative comments on improving the code logic. RCGs lack this context. As a result, they often question the rationale behind a code change rather than the logic of the code being changed. Providing all of the available code and documentation may not resolve this problem, since the additional context may be misleading, yielding poorer results, while imposing higher computation costs [67]. Based on this observed behaviour, our findings suggest that humans are better suited to review code changes involving complex logic. They often question the logic of the changes, leading to potentially useful discussions. On the other hand, RCGs, probe the logic underlying a change considerably less frequently and ask more frequently about the purpose.

RCGs excel at identifying unhandled exceptions; however, they fall short in contextual understanding, frequently using interrogative comments to get code change context (54.37%).

What Are the Perceived Intentions Behind Generated Interrogative Comments? During labeling, we encounter all intent categories [19] except *Attitudes and emotions*.

Human reviewers primarily (63.11% of sampled comments) use interrogative comments to provide suggestions, whereas task-specific (56.84%) and LLM-based (72.89% for GPT-4 and 84.09% for LLaMA2) RCGs primarily ask questions to gain more information or justifications for code changes. For example, in a specific code change,[12] a reviewer suggests, "*Cannot be list.sort(comparing(GpgKeyInfo::id))?*", while CodeBert comments, "*This is a bit confusing. Does this work for a single GpgKeyInfo?[...]*", reflecting confusion possibly due to limited context. This outcome supports our previous observation concerning the comment types. Because current RCGs do not engage in discussions and do not use the information provided by the author, these comments hinder productive use of RCGs. LLMs do not face this limitation, but this may still hinder their usefulness since, if they do not provide a suggestion, humans must still do so.

Indeed, *a considerable portion of LLM-based RCGs inquiries request more information about the code change (72.89% for GPT-4 and 84.09% for LLaMA2), prompting authors to propose solutions rather than offering them like human reviewers.* This trend may be partially attributed to a relative lack of context compared to human reviewers, who request additional information in 23.30% of interrogative comments. It also exceeds the rate at which task-specific RCGs seek information (56.84% of times). Although this trait is beneficial, authors may have to respond to many questions when addressing reviews, potentially prolonging the code review process. Even when authors provide the requested information in their responses, LLMs do not always engage with discussion threads, resulting in wasted time and unresolved comments.

Human reviewers occasionally (4.85%) propose hypothetical scenarios to probe potential issues—a practice that task-specific RCGs do not replicate, with the exception of GPT-4 as an LLM-based RCG (12.65%). For example, in a specific code change, a reviewer inquires, "*Should we reload plugins in dependency order if the caller gave us more than one and one depends on the other??*"[13] This question highlights a scenario potentially missed by the code author. This illustrates the limited capacity of RCGs to consider the relevant project context to draw attention to defects that authors may have overlooked. Task-specific RCGs lack the relevant project context and the capacity to assess the code for potential hypothetical scenarios, thus they cannot ask hypothetical questions that can help draw attention to potential bugs or future issues. Our qualitative analysis reveals that the LLaMA2-based RCG similarly lacks the capability for this task. Meanwhile, GPT-4-based RCG is capable of asking such types of questions, asking them more often than humans. This showcases the need to carefully select the type of RCGs based on the desired outcomes.

[12]https://gerrit-review.googlesource.com/c/gerrit/+/194420
[13]https://gerrit-review.googlesource.com/c/gerrit/+/54428

Our intention-based analysis highlights that most of the studied RCGs have a limited understanding of the code's context. LLM-based RCGs could mitigate this issue using transfer learning techniques [68], but still require advancements in reasoning [69] to simulate the nuanced thought processes of human reviewers [70], [71], [72]. Thus, human reviewers provide unique benefits for code changes that rely on historical context or external information, such as bug fixes or integration with private APIs. Using RCGs for such reviews may lead to an ineffective cycle of rationale-seeking review comments.

Rhetorical questions are used by humans (8.74%) to draw author attention [73], strengthen arguments, and compel action [74]; however, task-specific (3.16%) and LLM-based (0% for GPT-4 and 2.27% in LLaMA2) RCGs seldom employ this technique. Even if RCGs could express the same concepts in a declarative fashion, this could hamper the communication of ideas and increase author resistance to perceiving mistakes. This finding, though subtle, suggests that task-specific RCGs do not fully leverage the expressive capacity of language. This could diminish RCG effectiveness in highlighting the significance of some comments over others through careful word choice. LLM-based RCGs may perform better in this regard at the cost of increased latency and resource consumption [75].

Human reviewers predominantly use interrogative comments (63.11%) to offer suggestions, highlighting the subtle differences in review strategies. Their second most common intention is to request information about the change (23.30%). Conversely, task-specific RCGs comments mainly request more information (56.84%), with recommendations being the next most common purpose (40%). LLM-based RCGs follows the same pattern with GPT-4 and LLaMA requesting more information 72.89% and 84.09% of the time and making suggestions 14.46% and 13.64% of the time, respectively.

VII. Threats to Validity

Construct Validity threats undermine measurement effectiveness [76]. One construct threat is imposed by the heuristic that we use to identify interrogative comments. To assess its accuracy, we manually label 377 review comments and observe a 0.894 kappa score, indicating almost perfect agreement.

Internal Validity threats relate to uncontrolled confounding factors [76]. Of concern is the subjective judgment of coders in qualitative tasks. To mitigate this, we adhered to best practices [77], [78], [79] for qualitative analysis. Disagreements between coders were resolved through collaborative discussion until a consensus was reached. Cohen's Kappa inter-rater reliability scores indicate moderate to perfect agreement. Given the complexity of selecting labels from lists of 15 and 5 categories for the type and intent of comments, respectively, such agreement levels are often deemed acceptable [80], [81], [82]. Also, to balance coder subjectivity and allow new categories to emerge, we employed a blended coding approach [61].

External Validity threats impact the generalizability of the findings. One such threat is our focus on a single community. Although this limits generalizability, we chose the Gerrit community for its well-established review practices and prior use in related studies [50]. Its consistency makes it a strong representative of communities with sustained code review culture. Also, our study relies on simulated data rather than developer interactions with RCG. Thus, the findings may not fully generalize to settings where human developers respond to RCG-generated interrogative comments. Future work should validate these effects using actual developer discussions.

VIII. Conclusions and Lessons Learned

In this paper, we study interrogative comments generated by RCGs. We use three task-specific RCGs [13], [14], [36] and three LLMs to generate code review comments and quantitatively and qualitatively analyze the interrogative ones. Below, we distill lessons for development and research communities.

On Development. ***RCGs should be used to focus on specific, technical, interrogative comments during code review.*** When RCGs pose questions, they primarily inquire about the rationale behind changes (56.84%, 72.89%, 84.09% for task-specific, GPT-4-, and LLaMA2-based RCGs, respectively), unlike human reviewers who more often propose solutions (63.11%). We conjecture that projects can benefit from presenting RCG-generated comments to human reviewers during the review process. This integration could include a pre-review stage where RCGs generate comments to guide human reviewers' attention. RCG-generated comments can also help with weak spots missed by humans, such as exception handling or missing documentation. Finally, reviewers can use RCGs to articulate concerns when an issue is hard to pinpoint, potentially unblocking their reasoning process. This is supported by our results where RCGs more frequently ask questions related to code exceptions (7.37%, 14.46%, and 20.45% for task-specific, GPT-4-, and LLaMA2-based RCGs, respectively) and documentation (2.41% for GPT-4 and 4.55% for LLaMA2) than humans (0.97% and 2.91% for documentation and exceptions). This approach can complement human reviewers, leading to a broader review process. Similar approaches have been effective in enhancing code reviews [3], [42].

On Research. ***Exploring ways to control the generation of interrogative comments and proposing methods to improve discussion thread responses are promising directions.*** The share of interrogative comments for task-specific RCGs (median 15.61%) highlights their importance. Since task-specific RCGs do not currently participate in follow-up discussions, the usefulness of this portion of generated comments, especially for discussion-inducing changes, is hindered. Prior work has shown promise in cleaning datasets to improve LLM-based performance at specific code review tasks [29]. Thus, we recommend that research further focus on either reducing the current conversation-impeding interrogative comments from these models or integrating additional context into the model inputs. This enhancement could enable RCGs to respond to the various facets of a change more effectively. Future research should also determine how to efficiently choose (and further improve) RCGs to respond to code review discussions and interrogative comments, a previously infeasible task.

REFERENCES

[1] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "An empirical study of the impact of modern code review practices on software quality," *Empirical Software Engineering*, vol. 21, pp. 2146–2189, 2016.

[2] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 712–721.

[3] F. Kazemi, M. Lamothe, and S. McIntosh, "Exploring the notion of risk in code reviewer recommendation," in *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2022, pp. 139–150.

[4] E. Tempero and Y.-C. Tu, "Assessing understanding of maintainability using code review," in *Proceedings of the 23rd Australasian Computing Education Conference*, 2021, pp. 40–47.

[5] E. Mirsaeedi and P. C. Rigby, "Mitigating turnover with code review recommendation: Balancing expertise, workload, and knowledge distribution," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 1183–1195.

[6] O. Kononenko, O. Baysal, and M. W. Godfrey, "Code review quality: How developers see it," in *Proceedings of the 38th international conference on software engineering*, 2016, pp. 1028–1038.

[7] A. Bosu and J. C. Carver, "Impact of peer code review on peer impression formation: A survey," in *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. IEEE, 2013, pp. 133–142.

[8] Y. Murakami, M. Tsunoda, and H. Uwano, "Wap: Does reviewer age affect code review performance?" in *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2017, pp. 164–169.

[9] Y. Hong, C. Tantithamthavorn, P. Thongtanunam, and A. Aleti, "Commentfinder: a simpler, faster, more accurate code review comments recommendation," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 507–519.

[10] M. Staron, M. Ochodek, W. Meding, and O. Söder, "Using machine learning to identify code fragments for manual review," in *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2020, pp. 513–516.

[11] R. Tufano, S. Masiero, A. Mastropaolo, L. Pascarella, D. Poshyvanyk, and G. Bavota, "Using pre-trained models to boost code review automation," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 2291–2302.

[12] R. Tufano, L. Pascarella, M. Tufano, D. Poshyvanyk, and G. Bavota, "Towards automating code review activities," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 163–174.

[13] Z. Li, S. Lu, D. Guo, N. Duan, S. Jannu, G. Jenks, D. Majumder, J. Green, A. Svyatkovskiy, S. Fu *et al.*, "Automating code review activities by large-scale pre-training," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 1035–1047.

[14] L. Li, L. Yang, H. Jiang, J. Yan, T. Luo, Z. Hua, G. Liang, and C. Zuo, "Auger: automatically generating review comments with pre-training models," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 1009–1021.

[15] G. Viviani, C. Janik-Jones, M. Famelis, X. Xia, and G. C. Murphy, "What design topics do developers discuss?" in *Proceedings of the 26th Conference on Program Comprehension*, 2018, pp. 328–331.

[16] Y. Liu, C. Tantithamthavorn, Y. Liu, P. Thongtanunam, and L. Li, "Automatically recommend code updates: Are we there yet?" *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 8, Dec. 2024.

[17] H. Y. Lin, C. Liu, H. Gao, P. Thongtanunam, and C. Treude, "Codereviewqa: The code review comprehension assessment for large language models," 2025. [Online]. Available: https://arxiv.org/abs/2503.16167

[18] X. Zhou, K. Kim, B. Xu, D. Han, J. He, and D. Lo, "Generation-based code review automation: How far are we," in *2023 IEEE/ACM 31st International Conference on Program Comprehension (ICPC)*. IEEE, May 2023. [Online]. Available: https://doi.org/10.1109/icpc58990.2023.00036

[19] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik, "Communicative intention in code review questions," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2018, pp. 519–523.

[20] H.-Y. Li, S.-T. Shi, F. Thung, X. Huo, B. Xu, M. Li, and D. Lo, "Deepreview: automatic code review using deep multi-instance learning," in *Advances in Knowledge Discovery and Data Mining: 23rd Pacific-Asia Conference, PAKDD 2019, Macau, China, April 14-17, 2019, Proceedings, Part II 23*. Springer, 2019, pp. 318–330.

[21] V. J. Hellendoorn, J. Tsay, M. Mukherjee, and M. Hirzel, "Towards automating code review at scale," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 1479–1482.

[22] S. Mahbub, M. E. Arafat, C. R. Rahman, Z. Ferdows, and M. Hasan, "Reviewranker: A semi-supervised learning based approach for code review quality estimation," *arXiv preprint arXiv:2307.03996*, 2023.

[23] P. Thongtanunam, C. Pornprasit, and C. Tantithamthavorn, "Autotransform: Automated code transformation to support modern code review process," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 237–248.

[24] H. Y. Lin and P. Thongtanunam, "Towards automated code reviews: Does learning code structure help?" in *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2023, pp. 703–707.

[25] Y. Yin, Y. Zhao, Y. Sun, and C. Chen, "Automatic code review by learning the structure information of code graph," *Sensors*, vol. 23, no. 5, p. 2551, 2023.

[26] Z. Cao, S. Lv, X. Zhang, H. Li, Q. Ma, T. Li, C. Guo, and S. Guo, "Structuring meaningful code review automation in developer community," *Engineering Applications of Artificial Intelligence*, vol. 127, p. 106970, 2024.

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[28] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 5485–5551, 2020.

[29] C. Liu, H. Y. Lin, and P. Thongtanunam, "Too noisy to learn: Enhancing data quality for code review comment generation," 2025.

[30] N. Wadhwa, J. Pradhan, A. Sonwane, S. P. Sahu, N. Natarajan, A. Kanade, S. Parthasarathy, and S. Rajamani, "Core: Resolving code quality issues using llms," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 789–811, 2024.

[31] Y. Yu, G. Rong, H. Shen, H. Zhang, D. Shao, M. Wang, Z. Wei, Y. Xu, and J. Wang, "Fine-tuning large language models to improve accuracy and comprehensibility of automated code review," *ACM transactions on software engineering and methodology*, vol. 34, no. 1, pp. 1–26, 2024.

[32] G. Sridhara, S. Mazumdar *et al.*, "Chatgpt: A study on its utility for ubiquitous software engineering tasks," *arXiv preprint arXiv:2305.16837*, 2023.

[33] Q. Guo, J. Cao, X. Xie, S. Liu, X. Li, B. Chen, and X. Peng, "Exploring the potential of chatgpt in automated code refinement: An empirical study," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, 2024, pp. 1–13.

[34] T. Sun, J. Xu, Y. Li, Z. Yan, G. Zhang, L. Xie, L. Geng, Z. Wang, Y. Chen, Q. Lin *et al.*, "Bitsai-cr: Automated code review via llm in practice," 2025. [Online]. Available: https://arxiv.org/abs/2501.15134

[35] Z. Rasheed, M. A. Sami, M. Waseem, K.-K. Kemell, X. Wang, A. Nguyen, K. Systä, and P. Abrahamsson, "Ai-powered code review with llms: Early results," *arXiv preprint arXiv:2404.18496*, 2024.

[36] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, "Codebert: A pre-trained model for programming and natural languages," *arXiv preprint arXiv:2002.08155*, 2020.

[37] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[38] V. Balachandran, "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, 2013, pp. 931–940.

[39] A. Gupta and N. Sundaresan, "Intelligent code reviews using deep learning," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'18) Deep Learning Day*, 2018.

[40] J. K. Siow, C. Gao, L. Fan, S. Chen, and Y. Liu, "Core: Automating review recommendation for code changes," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2020, pp. 284–295.

[41] B. Lin, S. Wang, Z. Liu, Y. Liu, X. Xia, and X. Mao, "Cct5: A code-change-oriented pre-trained model," *arXiv preprint arXiv:2305.10785*, 2023.

[42] M. Vijayvergiya, M. Salawa, I. Budiselić, D. Zheng, P. Lamblin, M. Ivanković, J. Carin, M. Lewko, J. Andonov, G. Petrović *et al.*, "Ai-assisted assessment of coding practices in modern code review," *arXiv preprint arXiv:2405.13565*, 2024.

[43] U. Cihan, V. Haratian, A. İçöz, M. K. Gül, Ö. Devran, E. F. Bayendur, B. M. Uçar, and E. Tüzün, "Automated code review in practice," 2024. [Online]. Available: https://arxiv.org/abs/2412.18531

[44] J. Lu, L. Yu, X. Li, L. Yang, and C. Zuo, "Llama-reviewer: Advancing code review automation with large language models through parameter-efficient fine-tuning," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2023, pp. 647–658.

[45] C. Pornprasit and C. Tantithamthavorn, "Gpt-3.5 for code review automation: How do few-shot learning, prompt design, and model fine-tuning impact their performance?" *arXiv preprint arXiv:2402.00905*, 2024.

[46] A. Elnaggar, W. Ding, L. Jones, T. Gibbs, T. Feher, C. Angerer, S. Severini, F. Matthes, and B. Rost, "Codetrans: Towards cracking the language of silicon's code through self-supervised deep learning and high performance computing," *arXiv preprint arXiv:2104.02443*, 2021.

[47] Y. Wang, W. Wang, S. Joty, and S. C. Hoi, "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," in *EMNLP*, 2021.

[48] J. Siegmund, N. Siegmund, and S. Apel, "Views on internal and external validity in empirical software engineering," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 9–19.

[49] M. Chouchen, A. Ouni, J. Olongo, and M. W. Mkaouer, "Learning to predict code review completion time in modern code review," *Empirical Software Engineering*, vol. 28, no. 4, p. 82, 2023.

[50] I. X. Gauthier, M. Lamothe, G. Mussbacher, and S. McIntosh, "Is historical data an appropriate benchmark for reviewer recommendation systems? : A case study of the gerrit community," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2021, pp. 30–41.

[51] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–35, 2023.

[52] A. Fan, M. Lewis, and Y. Dauphin, "Hierarchical neural story generation," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018, pp. 889–898.

[53] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt, "A prompt pattern catalog to enhance prompt engineering with chatgpt," *arXiv preprint arXiv:2302.11382*, 2023.

[54] H. Dang, L. Mecke, F. Lehmann, S. Goller, and D. Buschek, "How to prompt? opportunities and challenges of zero-and few-shot learning for human-ai interaction in creative applications of generative models," *arXiv preprint arXiv:2209.01390*, 2022.

[55] G. J. Upton, "Fisher's exact test," *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, vol. 155, no. 3, pp. 395–402, 1992.

[56] T. W. MacFarland, J. M. Yates, T. W. MacFarland, and J. M. Yates, "Mann–whitney u test," *Introduction to nonparametric statistics for the biological sciences using R*, pp. 103–132, 2016.

[57] C. E. Shannon, "A mathematical theory of communication," *ACM SIGMOBILE mobile computing and communications review*, vol. 5, no. 1, pp. 3–55, 2001.

[58] O. J. Dunn, "Multiple comparisons among means," *Journal of the American statistical association*, vol. 56, no. 293, pp. 52–64, 1961.

[59] F. E. Zanaty, T. Hirao, S. McIntosh, A. Ihara, and K. Matsumoto, "An empirical study of design discussions in code review," in *Proceedings of the 12th ACM/IEEE international symposium on empirical software engineering and measurement*, 2018, pp. 1–10.

[60] M. Ochodek, M. Staron, W. Meding, and O. Söder, "Automated code review comment classification to improve modern code reviews," in *International Conference on Software Quality*. Springer, 2022, pp. 23–40.

[61] M. E. Graebner, J. A. Martin, and P. T. Roundy, "Qualitative data: Cooking without a recipe," *Strategic Organization*, vol. 10, no. 3, pp. 276–284, 2012.

[62] A. J. Fugard and H. W. Potts, "Supporting thinking on sample sizes for thematic analyses: a quantitative tool," *International journal of social research methodology*, vol. 18, no. 6, pp. 669–684, 2015.

[63] H. R. Bernard, A. Wutich, and G. W. Ryan, *Analyzing qualitative data: Systematic approaches*. SAGE publications, 2016.

[64] G. Guest, E. Namey, and M. Chen, "A simple method to assess and report thematic saturation in qualitative research," *PloS one*, vol. 15, no. 5, p. e0232076, 2020.

[65] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *biometrics*, pp. 159–174, 1977.

[66] P. Thongtanunam, C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida, and K.-i. Matsumoto, "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, 2015, pp. 141–150.

[67] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, "Lost in the middle: How language models use long contexts," *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 157–173, 2024.

[68] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar *et al.*, "Inner monologue: Embodied reasoning through planning with language models," in *Conference on Robot Learning*. PMLR, 2023, pp. 1769–1782.

[69] M. Zečević, M. Willig, D. S. Dhami, and K. Kersting, "Causal parrots: Large language models may talk causality but are not causal," *Transactions on Machine Learning Research*, 2023. [Online]. Available: https://openreview.net/forum?id=tv46tCzs83

[70] Y. Bang, S. Cahyawijaya, N. Lee, W. Dai, D. Su, B. Wilie, H. Lovenia, Z. Ji, T. Yu, W. Chung *et al.*, "A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity," *arXiv preprint arXiv:2302.04023*, 2023.

[71] L. Berglund, M. Tong, M. Kaufmann, M. Balesni, A. C. Stickland, T. Korbak, and O. Evans, "The reversal curse: LLMs trained on "a is b" fail to learn "b is a"," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=GPKTIktA0k

[72] H. Liu, R. Ning, Z. Teng, J. Liu, Q. Zhou, and Y. Zhang, "Evaluating the logical reasoning ability of chatgpt and gpt-4," *arXiv preprint arXiv:2304.03439*, 2023.

[73] L. Varpio, "Using rhetorical appeals to credibility, logic, and emotions to increase your persuasiveness," *Perspectives on medical education*, vol. 7, pp. 207–210, 2018.

[74] R. E. Petty, J. T. Cacioppo, and M. Heesacker, "Effects of rhetorical questions on persuasion: A cognitive response analysis." *Journal of personality and social psychology*, vol. 40, no. 3, p. 432, 1981.

[75] J. Kaddour, J. Harris, M. Mozes, H. Bradley, R. Raileanu, and R. McHardy, "Challenges and applications of large language models," *arXiv preprint arXiv:2307.10169*, 2023.

[76] H. K. Wright, M. Kim, and D. E. Perry, "Validity concerns in software engineering research," in *Proceedings of the FSE/SDP workshop on Future of software engineering research*, 2010, pp. 411–414.

[77] K. Charmaz, "Constructing grounded theory," 2014.

[78] K. Krippendorff, *Content analysis: An introduction to its methodology*. Sage publications, 2018.

[79] A. Scheopner Torres, J. Brett, J. Cox, and S. Greller, "Competency education implementation: Examining the influence of contextual forces in three new hampshire secondary schools," *AERA Open*, vol. 4, no. 2, p. 2332858418782883, 2018.

[80] A. K. Turzo and A. Bosu, "What makes a code review useful to opendev developers? an empirical investigation," *Empirical Software Engineering*, vol. 29, no. 1, p. 6, 2024.

[81] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik, "Confusion detection in code reviews," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2017, pp. 549–553.

[82] A. Krutauz, T. Dey, P. C. Rigby, and A. Mockus, "Do code review measures explain the incidence of post-release defects? case study replications and bayesian networks," *Empirical Software Engineering*, vol. 25, pp. 3323–3356, 2020.