

Understanding Developer-Analyzer Interactions in Code Reviews

Berk Cirisci, Linghui Luo, Muhammad Numair Mansur, Martin Schäf, Omer Tripp, Muhammad Bilal Zafar, Qiang Zhou
Amazon Web Services

Daniel Sanchez
Amazon Alexa

ABSTRACT

Static code analyzers are now a common part of the code review process. These automated tools integrate into the code review process by commenting on code changes and suggesting improvements, in the same way as human reviewers. The comments made by static analyzers often trigger a conversation between developers to align on if and how the issue should be fixed. Because developers rarely give feedback directly to the tool, understanding the sentiment and intent in the conversation triggered by the tool comments can be used to measure the usefulness of the static analyzer.

In this paper, we report on an experiment where we use large language models to automatically label and categorize the sentiment and intent of such conversations triggered by static analyzer comments. Our experiment demonstrates that LLMs not only classify and interpret complex developer-analyzer conversations, but can be more accurate than human experts.

CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging**; **Software maintenance tools**.

ACM Reference Format:

Berk Cirisci, Linghui Luo, Muhammad Numair Mansur, Martin Schäf, Omer Tripp, Muhammad Bilal Zafar, Qiang Zhou and Daniel Sanchez. 2024. Understanding Developer-Analyzer Interactions in Code Reviews. In *39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24)*, October 27-November 1, 2024, Sacramento, CA, USA. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3691620.3695257>

1 INTRODUCTION

Static code analyzers are increasingly becoming common in the code review process. Several studies have found that including static code analyzers in the code review process is an effective way to detect and eliminate software bugs early in the software development process (e.g., [1–7]). Typically, code reviews are conducted over a shared web interface where the developer, human reviewers, and static analyzers can comment on the proposed changes over several revisions until an agreement is reached and the changed code is merged or discarded.

In this code review workflow, static analyzers comment on changed lines with recommendations on how to improve quality or security of the proposed changes. The tool recommendations

Tool: Use a timezone-aware datetime object here.
Reviewer1: That's unnecessary, the naive datetime works fine.
Reviewer2: I agree, no need to overcomplicate this.

Figure 1: Example conversation between developers in response to a CodeGuru comment.

are presented in the same style as comments from human peer reviewers and can receive replies from humans involved in the review. Ideally, tool recommendations trigger a code change that address the highlighted issue, and the comment disappears in the next revision of the code review (this is sometimes called fix rate, or action rate). If a tool's recommendation is not addressed, we, as the tool developers, need to understand why. Is it due to the in-applicability of the suggestion, a misunderstanding of the advice, or simply because the issue is not deemed significant enough to warrant a change? Unraveling these nuances is crucial for refining tool effectiveness and enhancing user experience.

Previous research, such as a study in [8], has identified a variety of reasons why tool comments are not addressed, such as "too expensive to fix", "does not manifest in practice", or "warnings not understood". However, directly eliciting this sort of information through developer ratings on tool comments has proven ineffective. Studies in [4] and [9] show that developers rate comments made by static analyzers in less than 5% of cases. This makes the data too noisy and sparse for practical use.

The question arises: *How can we effectively capture developers' sentiments towards a tool comment?* Understanding developer sentiment would allow us to react quickly by either suppressing the rule that generated an unhelpful comment or flagging the rule for improvement.

In comparison to ratings given by developers, we found that a more reliable source of data is developer conversation in reaction to comments made by static analyzers. Figure 1 shows an example of such conversation. At Amazon Web Services (AWS), we run a static analyzer service called CodeGuru on internal code reviews, which generates comments for a variety of code quality and security issues. The static rules supported by CodeGuru are centrally maintained. We observe that over 15% comments generated by CodeGuru receive at least one reply from developers. For comments generated by the top 100 most often triggered rules supported by CodeGuru, this number increases to 30% on average.

To answer the question above, the idea we have is to map these conversations, consisting of tool comment and human replies, to sentiments and intents that help us understand how we can improve

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
ASE '24, October 27-November 1, 2024, Sacramento, CA, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1248-7/24/10.
<https://doi.org/10.1145/3691620.3695257>

the underlying analysis rules. Language models like RoBERTa [10] have successfully been used in many domains to classify the sentiment of text as positive, negative, or neutral. However, for our use case in CodeGuru it does not scale well since we have hundreds of different rules for different programming languages with unique recommendations, which all require some set of labeled data to fine-tune the RoBERTa model.

Further, we want to map conversations to more granular labels, e.g., distinguishing cases where developers don't understand unclear recommendations from cases where they reject false alarms, as both require different handling by the analyzer maintainers. The isolated sentiment of a comment alone might be ambiguous. A comment may express agreement with a previous comment, while containing a different sentiment polarity than that previous comment. Consider the conversation in Figure 1. Reviewer 2's comment has a different sentiment from Reviewer 1's preceding comment, but is actually expressing agreement with it. The individual comment's sentiment doesn't reveal what is being agreed with. Therefore, we need richer labels that go beyond classifying isolated comment sentiments to capture the full context and intent within the conversation. This allows us to disambiguate agreements and determine the true underlying feedback on both recommendations and tool performance.

Rather than mapping the conversation to a small set of sentiment labels, and providing the corresponding ground truth, we want to evaluate if we can describe the meaning of the labels that we want to distinguish to a large language model (LLM) and ask which of these summarizes a conversation the best. Summarizing, comprehending and reasoning about a given text is a task that large language models particularly excel at [11–13].

Unlike traditional machine learning models, large language models have extensive natural language understanding skills without needing explicit training data [14–17]. Their foundation of ingesting millions of texts across diverse domains and instruction tuning [17] equip them to interpret new conversations. By harnessing the embedded knowledge within the LLM, we avoid time-consuming and biased collection of labeled conversation data. The model can classify by understanding the provided label meanings rather than relying on manual annotations. This showcases how the innate capabilities of LLMs can be effectively utilized for natural language tasks without traditional training.

In this paper, we conduct an experiment to check the effectiveness of using LLMs to automatically understand and interpret human replies to the comments generated by CodeGuru. We want to assess if an LLM will allow us to map entire conversations to complex sentiments at an accuracy that is comparable to human labeling, without requiring us to provide labeled tuning data for each of the hundreds of rules that CodeGuru has. Our experiment involves following steps:

First, we identify a set of descriptive labels for sentiments and intents by having domain experts manually review conversations and agree on the labels. Instead of abstract labels, we use natural language descriptions. We then craft prompts to guide the LLM in mapping conversations to these descriptive labels.

Second, to evaluate the concordance between LLM-inferred and human-generate labeling of conversations, we manually annotate 100 conversations collected from CodeGuru, and compare the results

against the labels that are proposed by the LLM. We check how often the LLM agrees with a human provided ground truth.

Lastly, recognizing that LLMs might outperform human capabilities in text summarization, as mentioned by [11], we explore the adaptability of human judgment in the face of LLM-generated rationales. We investigate if the humans that labeled original conversations change their labels when presented with the reasoning of the LLM. To that end, we create a second prompt for the LLM to explain why it chooses a label. Like in original labeling process, we pair the LLM with a human to see if the human will change their ground truth when presented with the reasoning from the LLM.

The key takeaway from our experiment are:

- Human labeling of developer conversations suffers from inherent variability and subjectivity, where experts often diverged in their interpretations.
- LLMs can effectively categorize developer conversations in code reviews, outperforming human experts in depth and accuracy without requiring extensive labeled training data.
- Human experts frequently reconsidered and changed their initial labels after being presented with the LLM's summaries and rationale, highlighting the influence of LLM outputs on human judgment in such tasks.

2 RELATED WORK

In this paper, we explore the nuanced interactions between developers and static code analysis tools in code reviews. This investigation aligns with a growing interest in automating aspects of the software development process, particularly code quality assurance. The related work spans across various domains, including automated code-analysis, AI-driven code review, and the integration of machine learning in software development. Below we discuss related work in these areas in detail.

Static Code Analysis in Code Reviews. Static code analysis tools have become integral part of the code review process. Prior research has demonstrated their effectiveness in early detection and elimination of various classes of software bugs [18]. Singh *et al.* [19] evaluated how these tools can reduce code review efforts, emphasizing their utility in detecting potential coding issues before they escalate. Similarly, Panichella *et al.* [20] explored the role of static analysis tools in aiding developers during code reviews, highlighting their potential in improving code quality. Additionally, Tufano *et al.* [21] investigated the impact of these tools on the software development lifecycle, providing insights into their broader applicability in various coding scenarios.

Developer Interaction with Analysis Tools. The interaction between developers and static analysis tools has been a subject of interest in recent studies. Sadowski *et al.* [22] identified various reasons why developers might overlook tool recommendations, such as the perceived cost of fixing issues or a lack of understanding of the warnings. Additionally, Johnson *et al.* [23] and Calcagno *et al.* [24] have discussed the challenges developers face in interpreting and implementing tool suggestions, emphasizing the need for clearer communication between tools and developers.

Large Language Models in Software Engineering. The application of large language models in software engineering, particularly in sentiment and intent analysis, is a growing research area.

```

233 from datetime import datetime
234 def hello_world():
235     return datetime.now()
236

```

Tool: Naive datetime objects are treated by many datetime methods as local times. We recommend that you create timezone aware timestamps with the `now` or `fromtimestamp` functions, using the `tzinfo` attribute to represent times in UTC. Do not use `datetime.utcnow()` because it returns only a naive datetime object.

Reviewer1: Can we change this if it is not too much work? In our case we should be fine as even if we are in the wrong timezone we should have plenty of time to respond

Developer: Output without time zone information: 2023 – 03 – 15 10 : 10 : 12.102031

Output with time zone information: 2023 – 03 – 15 10 : 10 : 12.102031 + 00 : 00 I don't see changing this will be beneficial for our use case.

Reviewer1: ack, lgtm

Figure 2: Example conversation between developers and reviewers in response to a CodeGuru comment.

Lin *et al.* [25] demonstrated the efficacy of models like RoBERTa in classifying text sentiments, which is crucial for understanding developer responses to tool-generated comments. Veen *et al.* [11] further showcased the capability of large language models in summarizing complex texts and conversations, aligning with our approach of mapping developer-tool interactions to specific labels. Additional studies by Novielli *et al.* [26] and Mäntylä *et al.* [27] have further explored the application of natural language processing and machine learning techniques in understanding developer sentiments and automating software engineering tasks.

Furthermore, there is significant ongoing research leveraging LLMs in various aspects of software engineering. For instance, [28] explores the use of LLMs for code generation and completion. [29, 30] highlight the potential of LLMs in identifying and rectifying software bugs, enhancing software reliability. [31, 32] demonstrate how LLMs can automate documentation processes and assist in software maintenance.

3 MOTIVATING EXAMPLE

Developer interactions during code reviews, especially with static analyzers, can reveal intricate dynamics. Consider the scenario depicted in Figure 2. A static analyzer comments on a code change with a warning that a time format is used that does not include the time zone information. Missing time zone information can lead to consistency problems in (geographically) distributed systems, or cause anomalies during daylight saving adjustment. In this example, the first reviewer asks to apply the recommended change, if it is not too expensive. The developer responds by showing that the output does not change because the system already uses the UTC time zone, to which the reviewer acknowledges, implicitly agreeing that no further change is needed.

For our static analysis, we would like to learn two things from this conversation. First, there is a general consensus on the relevance and accuracy of the tool's recommendation, dismissing the possibility of a false alarm. Second, and more critically, the team concludes that addressing this specific issue is not essential for their current project. The rationale is that the system operates exclusively in the UTC time zone, rendering the recommended change unnecessary. This insight is important as it suggests a potential course of action for CodeGuru: suppressing similar recommendations for this project in future analyses.

Passing this conversation through a standard sentiment analysis model, such as `twitter-roberta-base-sentiment-latest`, yields an incomplete picture. The model broadly classifies the conversation as predominantly neutral with a slight negative bias (0.792 neutral, 0.160 negative). If we feed the comments individually to this language model, it will label the first comment as 0.705, then second as 0.560 neutral, and the last as 0.771 neutral, so without further fine-tuning, a Roberta model is not able to map this conversation to a positive or negative sentiment.

To get more fine-grained sentiments from a RoBERTa model, we would have to identify a set of labels that we want to distinguish and fine-tune the model with a sufficiently large amount of labeled training data.

The goal of this paper is to investigate if LLMs are capable of grasping the context and essence of such conversations without the need of extensive labeled datasets.

For example, an LLM will produce the following when asked to describe the sentiment and intent of this conversation:

The conversation indicates a consensus that, although the tool's advice is generally sound, it may not be necessary to implement in this particular case. The expected reaction is an acceptance of the current implementation without further changes, prioritizing practicality over strict adherence to the tool's recommendations.

This response already includes the information we are looking for and can be mapped to a set of sentiments, without further fine-tuning, using a carefully crafted prompt. If we follow up with a query that asks which of the summaries from Section 4.1 applies best to the above conversation, the LLM will respond:

The summary that best describes the above conversation is "Accept finding but reject remediation." In the conversation, the tool's finding about the use of naive datetime objects is acknowledged by the developers. However, the developer argues against the suggested remediation, demonstrating that the current approach without timezone information suffices for their specific use case. This indicates acceptance of the tool's finding (the issue with naive datetime objects) but rejection of the proposed solution (using timezone-aware timestamps).

The LLM's innate summarization and comprehension skills allow mapping conversations to complex labels without specialized training¹. In the following we discuss an experiment to evaluate how well this works in practice.

4 APPROACH

We approach our experiment by sampling 100 conversations between developers in response to static analysis findings. We first

¹<https://chat.openai.com/share/efc05209-eb65-4e9f-a980-25887b93470a>

349 devise a set of labels that we want to distinguish, then we design a
350 set of experiments to compare human and LLM labeling results.

352 4.1 Label Identification and Data Preparation

353 Our initial task in this study was the development of a comprehen-
354 sive set of labels that encapsulate the diverse range of sentiments
355 and intents found in the interactions between developers and static
356 analyzers. To achieve this, we employed qualitative research meth-
357 ods, particularly coding techniques as introduced by Saldaña [33],
358 which were important in methodically dissecting and understand-
359 ing the complexities of these interactions.

360 Our internal code review system, where CodeGuru and other
361 static analyzers are integrated, served as the primary data source.
362 In this system, every tool-generated finding is presented as a com-
363 ment linked to the relevant line of code in a pull request. These
364 comments subsequently get responses from the developers and
365 reviewers involved, leading to a thread of discussion that captures a
366 spectrum of reactions and decision-making processes. To systemati-
367 cally categorize these conversations, we utilized both inductive and
368 deductive coding methods. We began by analyzing a representative
369 sample of 20 conversations initiated by CodeGuru comments. This
370 analysis involved constructing an initial set of codes, grounded in
371 our prior experiences with CodeGuru feedback, and then allowing
372 the emergence of new codes as patterns were identified in the data.
373 This dual approach ensured that our coding framework was both
374 grounded in real-world scenarios and flexible enough to adapt to
375 the nuanced nature of these interactions.

376 To enhance the reliability of the coding process, we divided the
377 20 conversations into two equal sets and engaged four CodeGuru
378 developers, organized into two groups, to independently code each
379 set. Post-coding, these groups convened to compare and reconcile
380 their findings, thereby enhancing the robustness of the coding
381 outcomes through inter-coder reliability techniques.

382 The culmination of this process was the establishment of the
383 following set of labels, each representing a distinct category of
384 developer response:

- 385 • **Accept both finding and remediation:** Cases where de-
386 velopers agree with the tool’s finding and its proposed
387 solution.
- 388 • **Accept finding but confused about remediation:** Cases
389 where developers acknowledge the issue but are unclear
390 about the recommended remediation.
- 391 • **Accept finding but reject remediation:** Cases where the
392 finding is deemed correct, but the suggested remediation is
393 not suitable.
- 394 • **Confused about finding:** Cases where developers are
395 uncertain or misunderstand the tool’s reported issue.
- 396 • **Reject finding in current context:** Cases where the tool’s
397 finding is considered irrelevant or inapplicable to the spe-
398 cific context.
- 399 • **Tool is wrong:** Cases that the tool’s finding is inaccurate
400 or a false positive.
- 401 • **Unrelated:** Discussions that do not pertain directly to the
402 tool’s finding or suggested remediation.

403 Armed with these labels, we then labeled a broader data set of
404 100 conversations. Each conversation was independently labeled by
405

407 two authors, who subsequently collaborated to reach a consensus
408 on the final label. This set of 100 labeled conversations formed
409 the basis for an initial dataset that we could compare our model’s
410 predictions against in subsequent experiments. However, as will be
411 discussed in our detailed experiments in the next section, human
412 judgment on these complex conversational intents is inconsistent.

413 In our experimental results section, we quantify the disagree-
414 ment rates between the authors’ independent labels prior to consen-
415 sus. The data also shows how the human experts frequently changed
416 their original labels when presented with the LLM’s explanations.
417 This demonstrates the inherent uncertainty and subjectivity in hu-
418 man labeling of such interpretive conversational data. Therefore,
419 we do not treat this benchmark dataset as absolute ground truth.
420 The 100 labels provide a preliminary comparison point, while ac-
421 knowledging the underlying difficulty humans encounter in reliably
422 labeling conversational intents and sentiments. We use the data
423 to evaluate model predictions but recognize its limitations as a
424 subjective human-labeled sample.

425 With this preliminary set of categorized conversations estab-
426 lished, we next set out to test Claude’s ability to accurately apply
427 these labels to unlabeled code review discussions.

429 4.2 Experimentation With Claude

430 In the experimental phase, we used Claude, a LLM developed by
431 Anthropic, to classify the same set of 100 conversations described
432 previously that were initially labeled by human experts to, first,
433 establish the extent of Claude’s agreement with human labeling, and
434 second, to examine the influence of Claude-generated summaries
435 on the human re-evaluation of these conversations. The experiment
436 was conducted in the following stages:

- 437 (1) Claude generated a summary for each of the 100 conversa-
438 tions, providing a condensed understanding of the dialogue.
- 439 (2) Utilizing these summaries, Claude then proceeded to clas-
440 sify each conversation, selecting from a set of predefined
441 labels that were deemed most appropriate for the discus-
442 sion.
- 443 (3) We then conducted a comparative analysis of Claude’s la-
444 bels against those determined by human experts. In in-
445 stances of discrepancy between Claude’s and human labels,
446 Claude was prompted to reevaluate its decision in light of
447 the human-assigned labels.
- 448 (4) After Claude’s reconsideration, we asked humans to re-
449 evaluate their original labels based on Claude’s justification
450 and summaries
- 451 (5) The final stage involved reassessing Claude’s classification
452 accuracy in relation to the revised human expert baseline.

453 The summaries generated by Claude in the first stage of our
454 experiment were important in providing insights into how the
455 model interpreted each conversation. These summaries was also
456 used in the later part of the experiment when human experts were
457 asked to reconsider their initial labels, as discussed further in the
458 next section.

459 Through this methodology, we aimed to evaluate classifying com-
460 plex conversations, but more importantly, we wanted to examine
461 the potential biases and variability in human labeling.

By comparing Claude’s classifications to the human expert baseline, and then asking humans to re-evaluate based on Claude’s justification, we can observe how susceptible human judgment is to influence.

In more detail, the motivation for this experiment stems from several key challenges and considerations:

- (1) **Complexity of the Labeling Task:** The task of labeling developer conversations in the context of code reviews is inherently complex. It requires a thorough understanding of various factors including the code itself, the specific context of the code review, and the nuanced exchanges between developers. This complexity makes it difficult for humans to consistently apply labels accurately.
- (2) **Human Bias and Variability:** Human labelers are subject to biases and variability in their interpretations. Different individuals might perceive and understand the same conversation differently, leading to inconsistencies in labeling. This variability is a natural aspect of human judgment but poses a challenge in achieving objective and reliable labeling.
- (3) **Advancements in LLM Capabilities:** Due to recent advancements, LLMs have shown remarkable proficiency in understanding natural language in a variety of contexts. They are not only capable of understanding the nuances of human language but also demonstrate a strong capability in comprehending code.

5 EXPERIMENTAL EVALUATION

Building on the experimentation with Claude, we detail the experimental evaluation of our approach to understanding developer-analyzer interactions in code reviews using LLMs. The primary goal of this evaluation is to assess the effectiveness and dependability of LLMs, in our case Claude, in categorizing dialogues between developers and automated code analysis tools. Additionally, we aim to quantify the level of agreement (inter-rater reliability) between human experts and Claude. To provide a structured approach to our investigation, we have formulated several key research questions:

RQ1: Human Consensus How frequently do human experts reach a consensus when independently labeling developer conversations in code reviews, and what patterns of agreement or disagreement emerge across different categories of responses?

RQ2: Human confidence To what extent are human experts confident in their labeling decisions, and how does their confidence get affected when they are presented with Claude’s summaries and rationale for its labeling choices?

RQ3: Claude confidence What level of consistency does Claude demonstrate in its labeling choices when presented with initial labels assigned by human experts, and does it show a tendency to maintain or revise its decisions upon reevaluation?

RQ4: Stability of Claude’s labeling result How stable are the labeling results generated by Claude over multiple iterations?

RQ5: Usability of Predefined Labels How effective are the predefined labels used in our study for Claude’s labeling

process, and is Claude able to propose new, distinct labels that better capture the nuances of the conversations?

5.1 Experimental Setup

For our evaluation, we collect conversations between developers in response to CodeGuru comments from real code reviews. We only select conversations that have at least two human participants. Further, we only include conversations for CodeGuru rules if there are at least 3 relevant conversations for recommendations of that rule. This is to limit bias from small samples. Out of the conversations that match this criteria, we sampled 100 conversations randomly based on the relative frequency of the CodeGuru rule that sparked the conversation.

For our experiments, we use Anthropic’s Claude version 2.1² (anthropic.claude-v2:1) with temperature 0.0. As discussed in Section 4.1, each of the 100 conversations is labeled by two domain experts (authors of this paper) independently, and an agreement is reached in a joint meeting.

From Claude’s end, the labelling process began with Claude first generating a summary of each conversation. This summary was crucial as it provided insights into Claude’s interpretative reasoning, which was later utilized in RQ2 to potentially influence the human experts’ perspectives. The following prompt guided Claude in summarizing the conversations:

```
Your task is to understand the conversation between human developers and reviewer(s), summarize the conversation. The conversation is from a code review process which is between the automated code analysis tool named CodeGuru, the developer and reviewer(s). The conversation starts with a recommendation from the tool. Then the developer and reviewer(s) discuss about the recommendation.
```

```
A recommendation consists of:
* Finding: Coding issue such as security vulnerability or coding standard violation.
* Remediation: Recommended solution by the tool to fix the coding issue.
```

```
In their discussion, the developer and reviewer(s) try to decide if the finding is correct. Sometimes the developer and reviewer(s) can discuss about something unrelated to the finding.
```

```
Summarize the following conversation:
« conversation »
```

Following the summarization, we asked Claude to match its summary to the most fitting label from our predefined list (as detailed in Section 4.1), using this prompt:

```
From the following list of categories, which describes your summary the best?
« List of labels »
Explain your reasoning why you pick a category. End your
```

²<https://www.anthropic.com/index/claude-2>

label	Agreement count	Disagreement count	Overlap percentage
Accept both finding and remediation	49	4	92.4%
Accept finding but confused about remediation	3	4	42.8%
Accept finding but reject remediation	4	4	50%
Confused about finding	2	4	33.3%
Reject finding in current context	11	4	73.3%
Tool is wrong	2	1	66.6%
Unrelated	7	1	87.5%
Sum	78	22	78%

Table 1: Count of agreements and disagreements among human experts for different labels during the labeling of developer-analyzer conversations. The table highlights the areas where consensus was easily reached and where experts had differing opinions, showing the overall trend of expert agreement in labeling these interactions.

answer with the number of the summary that best describes the conversation.

The list of labels and their descriptions were inserted into the prompt, as derived from Section 4.1.

RQ1 takeaway: Human experts display strong agreement in straightforward cases but diverge in their opinions on more complex issues. This divergence underscores the subjective and context-dependent nature of human labeling, particularly in nuanced developer conversations.

5.2 Experimental Results

We now discuss the experimental results for each of our outlined research questions.

RQ1: Human consensus. We evaluated how often human experts reach consensus by having two experts independently label each conversation. Table 1 provides a breakdown of agreement and disagreement counts across different labels between human experts. The label *Accept both finding and remediation* saw the most consensus, with 49 instances, indicating a common understanding among experts in scenarios where developers agreed with both the issue and the suggested remedy by CodeGuru.

In contrast, labels like *Accept finding but confused about remediation* and *Accept finding but reject remediation* showed an even split of agreement and disagreement. This balance highlights the varied interpretations by humans in complex scenarios. More nuanced labels like *Confused about finding* and *Reject finding in current context* exhibited moderate disagreement, shedding light on the complexities in categorizing developer responses that are either confused or context-specific rejections of the tool’s findings. Straightforward labels such as *Tool is wrong* and *Unrelated* had fewer disagreements, likely due to their less subjective nature.

Table 2 presents a pair-wise comparison of expert label selections among human experts, offering deeper insights into the areas where human interpretations tend to diverge. The results reflect both the strengths and limitations of human judgment in classifying developer-analyzer interactions. While there is significant consensus in cases with clear-cut scenarios, the variability in interpretations becomes more apparent in categories that involve more nuanced or complex conversations. This variability highlights the inherent challenges in human-based labeling processes, especially when dealing with subjective and context-dependent content.

RQ2: Human confidence. To evaluate human confidence in their labeling decisions, we conducted a two-step evaluation. Initially, we asked the experts to form a consensus on the labels for each conversation and come up with final labels. Following this, the human experts were shown the summaries and rationales generated by Claude for its labeling choices. The experts were then asked to reconsider their original decisions in the light of the Claude’s reasoning and justifications. For each conversation, the human experts chose if they agree with the Claude’s justification and are willing to change their label, or if they believe their own label is correct. We only count it, if at least two out of three human experts are willing to change their label. This method aimed to measure the degree to which human confidence in their initial choices.

Table 3 shows the results for this experiment, which outlines the labels agreed upon by human experts (human count), the labels assigned by Claude (LLM count), and the degree of overlap between the two. Claude and human experts agreed in 57% of the cases. The most significant agreement was observed in the category *Accept both finding and remediation*, where the Claude’s labels coincided with the human experts’ decisions in 70% of the cases, and vice versa, the humans agreed with the Claude in 90% of these cases. Table 4 further dissects the 43 instances of divergence between human and Claude labels, highlighting the most common areas of disagreement. We can see that the most frequent disagreement is related to if the remediation of an accepted finding is rejected or if developers are confused about the remediation. There is also frequent disagreement if the tool is wrong, or if developers just reject the finding in its current context.

A significant shift in human experts’ confidence was observed after considering Claude’s justifications. The column *human reconsiders* in Table 3 shows that in 31 instances, human experts were willing to change their original labels after considering Claude’s reasoning. This shift was particularly pronounced in the category *Accept finding but reject remediation*, where 11 out of 16 cases saw a change in the human experts’ decision. It indicates that Claude’s justifications were compelling enough to cause the human experts

	Accept both finding and remediation	Accept finding but confused about remediation	Accept finding but reject remediation	Confused about finding	Reject finding in current context	Tool is wrong	Unrelated
Accept both finding and remediation	49	2	1	-	-	-	1
Accept finding but confused about remediation	1	3	2	-	-	1	-
Accept finding but reject remediation	-	-	4	-	2	-	2
Confused about finding	-	1	-	2	2	-	1
Reject finding in current context	1	-	-	1	11	2	-
Tool is wrong	-	-	-	-	1	2	-
Unrelated	1	-	-	-	-	-	7

Table 2: Pair-wise comparison of label selections among human experts. The horizontal axis represents the label selected by the first expert and the vertical axis represents the label selected by another expert. The table shows how often experts selected the same or different labels for the same conversation.

label	human count	LLM count	overlap	LLM reconsiders	human reconsiders
Accept both finding and remediation	54	42	38	0	1
Accept finding but confused about remediation	8	10	2	1	5
Accept finding but reject remediation	6	16	2	1	11
Confused about finding	3	4	1	0	3
Reject finding in current context	15	18	11	1	7
Tool is wrong	6	5	1	0	2
Unrelated	8	5	2	0	2
Sum			57	3	31

Table 3: A breakdown of how often each label was assigned by the human expert and by the LLM prompt, and how often they agreed on a label for the same conversation. The last two columns show how often the LLM was willing to reconsider its label when shown the label chosen by the human expert, and vice versa. To remove bias is *human reconsiders* we had three human experts review the LLMs justification and we only count if two of three humans are willing to change their label.

to revisit and alter their initial assessments. Combining instances of initial agreement with those revised post-justification, the agreement between Claude and Human reaches to 88 out of 100 cases.

Figure 3 shows an example of a real conversation where human experts changed their stance after considering Claude’s summary and justification. Initially labeled as Accept both finding and remediation by humans, Claude’s justification for Accept finding but confused about remediation led to a revised

Tool: The naive datetime objects are treated by many datetime methods as local times, it is preferred to use aware datetimes to represent times in UTC. The recommended way to create an aware datetime object representing a specific timestamp in UTC is by passing ‘tzinfo’ as an argument to the method.

Reviewer1: 1. Move this to a separate method so that it is abstracted out. 2. What if execution name already has ‘-dateTime’ in it when it got restarted by this script? Should we remove the existing dateTime from the workflow name?

Developer: 1. Sure. Will take a look. 2. Hmm. Good point!

Developer: What I will do here instead is I will append a number - ‘1, 2, 3,... etc’ after a restart.

Reviewer1: Yup, this sounds good.

decision by the human experts. Claude’s summary highlights the implicit acceptance of the tool’s finding but points out the lack of discussion or implementation of the recommended remediation, thereby categorizing the conversation as Accept finding but confused about remediation. Here is the summary generated by Claude:

Here is a summary of the conversation:

The tool flagged that naive datetimes without timezone information are being used, and recommended using aware datetimes in UTC instead.

Reviewer 1 had two suggestions:

1. Move the datetime logic to a separate method to abstract it out.
2. Handle the case where the execution name already contains a datetime when restarting. They asked if the existing datetime should be removed first.

The developer agreed to move the logic to a separate method. For handling restarts, instead of removing any existing datetime, they will append a number (1, 2, 3, etc) each time to distinguish multiple restarts.

Reviewer 1 agreed that appending a number on restart

Figure 3: Example conversation between developers and reviewers in response to a CodeGuru recommendation.

is a good solution.

In summary, the tool identified an issue with naive datetimes which was not explicitly addressed. The discussion focused more on refactoring suggestions from the reviewer related to abstracting out the datetime logic and properly handling workflow restarts. The developer agreed to make those changes, but there was no clear resolution related to using aware datetimes.

Following is the justification by Claude for choosing Accept finding but confused about remediation:

Based on the summary, I would categorize this conversation as Accept finding but confused about remediation. The rationale is:

- The tool's finding about using naive datetimes was implicitly accepted as the developer did not dispute or question it.
 - However, the proposed remediation of using aware UTC datetimes was not addressed or discussed. The conversation focused entirely on the refactoring suggestions from the reviewer.
- So while the datetime issue seems to be acknowledged, there is no indication that the developer implemented or understood how to implement the recommendation to use aware objects. The remediation piece remains unclear.

Therefore, Summary 02 best reflects this scenario where the finding is agreed to but the resolution path is uncertain, indicating confusion on how to resolve the tool's suggestion.

RQ2 takeaway: This experiment indicates a significant influence of Claude on human labeling decisions, revealing the challenges humans face in labeling nuanced conversations. The ease with which experts were swayed by Claude's reasoning underscores the complexities of objective labeling in intricate discussion contexts.

RQ3: LLM confidence. We evaluated Claude's confidence in its labeling decisions by presenting it with labels initially assigned by human experts and asking it to reconsider its own labels. This approach aimed to measure whether Claude would maintain its initial labeling decision or alter it based on the human-proposed labels. The findings, detailed in Table 3, show the instances where Claude reconsidered its labels in the *LLM reconsiders* column. Claude only revised its decision in 3 cases (accounting for 7% of cases where there was a discrepancy between human and LLM labels), indicating a higher consistency or confidence in its judgments compared to human experts.

Continuing the example from RQ2, when Claude was asked to reconsider its answer for the conversation in Figure 3, it upheld its initial label, indicating strong adherence to its interpretative logic:

I do not believe "Accept both finding and remediation" would accurately apply here. My reasoning is:

There is no clear evidence that the recommended remediation of using aware UTC datetimes was implemented or even discussed. The conversation was focused entirely on the separate refactoring suggestions from the reviewer.

Since there is no signal confirming the remediation was adopted, "Accept finding but confused about remediation" remains more fitting. While the finding seemed to be implicitly agreed to, the lack of resolution around the suggested fix indicates continued confusion or uncertainty there.

Outright acceptance of both the issue and its resolution requires more clarity that the recommendation was understood and addressed. Such confirmation is lacking based on the summary.

Therefore, I believe keeping the original label of "Accept finding but confused about remediation" better represents the scenario where agreement on the risk exists but the path forward is still unclear.

These results indicate consistency in Claude's labeling decisions, even when presented with differing human opinions. This steadfastness highlights its reliance on internal interpretive reasoning.

RQ3 takeaway: Claude demonstrates confidence in its labeling decisions, often maintaining its original labels even when presented with alternative human-assigned labels. This underscores its potential as a reliable labeling tool in tasks such as labeling conversations in developer-analyzer interactions.

RQ4: Stability of Claude's labeling results. To evaluate the consistency of Claude's labeling, we conducted three rounds of labeling for each conversation using a temperature setting of 0.0. In the first round, Claude labeled all 100 conversations. It then repeated this labeling process two more times. We compared the labels across rounds on a pair-wise basis. In 48 of the 100 conversations, Claude assigned identical labels in all three rounds, demonstrating high repeatability. Furthermore, the labels matched across at least two of the three rounds for 88 conversations, providing additional evidence of stable labeling behavior from Claude.

A closer examination of the discrepancies between labels in different runs is detailed in Table 5. The most significant disagreement was observed between the labels Accept both finding and remediation and Accept finding but reject remediation, followed by a disagreement between Accept both finding and remediation and Accept finding but confused about remediation. These disagreements align with the trends noted in Table 4, where human experts and Claude showed divergences in opinion.

	Accept both finding and remediation	Accept finding but confused about remediation	Accept finding but reject remediation	Confused about finding	Reject finding in current context	Tool is wrong	Unrelated
Accept both finding and remediation	-	5	7	0	1	1	2
Accept finding but confused about remediation	0	-	4	1	1	0	0
Accept finding but reject remediation	1	1	-	0	1	0	1
Confused about finding	0	1	1	-	0	0	0
Reject finding in current context	0	1	1	0	-	2	0
Tool is wrong	1	0	0	0	4	-	0
Unrelated	2	0	1	2	0	1	-

Table 4: Breakdown of the the pairs of labels for which human experts and LLMs chose different labels. The horizontal axis represents the label selected by the LLM and the vertical axis represents the label selected by the human. Each cell shows how often human and LLM disagreed on those labels.

Another notable disagreement was between *Reject finding in current context* and *Accept finding but reject remediation*. This particular disagreement, however, was not as frequent in Table 4. We hypothesize that enhancing the clarity of these labels in the prompt could potentially reduce such disagreements, thereby aiding Claude in making more distinct distinctions between these categories.

RQ4 takeaway: The experiment demonstrates Claude’s high level of stability in labeling results over multiple iterations. This stability is crucial for reliable and consistent classification of developer-analyzer interactions, affirming Claude’s suitability for such tasks.

RQ5: Usability of predefined labels. This research question aimed to evaluate the effectiveness of the predefined labels used in our study and whether Claude could propose new, distinct labels that better capture the nuances of the conversations. To investigate this, we asked Claude to categorize conversations using our predefined list of labels. Claude was also given the option to propose new labels if it found them more suitable. The following prompt was used for this purpose:

From the following list of categories, which describes your summary the best?

« list of categories »

Explain your reasoning why you pick a category. End your answer with the number of the summary that best describes the conversation. If there is a better summary than the ones above, propose a new Summary 08 and describe why we should use this summary instead.

In 98 out of 100 cases, the Claude favored one of the seven labels we had initially provided. This suggests that our selection of labels is comprehensive enough to encompass the variety of conversations in our dataset. However, in two instances, Claude proposed a new label, *Insufficient information highlighting conversations* where the existing labels did not sufficiently capture the context or details were lacking for an accurate assessment.

In another to discover other interesting labels, we reran the prompt with a higher temperature of 0.7, which is supposed to make the model more *creative*. However, Claude only generated a single new label, *Incomplete information* for one conversation. Adjusting the prompt to stimulate Claude into selecting new labels did not yield different outcomes. This leads us to believe that our existing set of labels sufficiently encapsulates the range of conversations in our dataset.

Lastly, we asked Claude to autonomously determine labels without relying on our predefined set. The prompt was as follows:

Define a category to classify the above conversation by sentiment and intent. The category should capture the general intent of the reviewers and developers towards the tool comment, and their likely intent in response to the comment. The category should consist of short sentence describing the intent, and a short paragraph that explains the category

Explain why you pick a category, and how the above conversation falls into this category.

Using this approach, Claude generated 100 unique labels, with each conversation receiving a distinct label. Some labels bore similarities, such as *Collaborative Exception Handling Enhancement* and *Collaborative exception context balancing*, but none were identical. Despite various modifications to the prompt, Claude did not consistently reuse labels across conversations.

Therefore, while Claude efficiently mapped conversations to our predefined labels and was satisfied with the labels provided (suggesting only one additional label), it was not successful in autonomously generating a set of reusable labels.

RQ5 takeaway: Claude’s effective use of predefined labels for most conversations suggests their comprehensive coverage. Our labels generally capture conversation dynamics well, yet there is room for refinement to address specific scenarios more accurately.

Summary. Our experiments aimed to determine the effectiveness and dependability of large language models in categorizing

	Accept both finding and remediation	Accept finding but confused about remediation	Accept finding but reject remediation	Confused about finding	Reject finding in current context	Tool is wrong	Unrelated
Accept both finding and remediation	90	15	22	1	5	2	9
Accept finding but confused about remediation	-	4	11	9	1	0	8
Accept finding but reject remediation	-	-	32	0	13	2	0
Confused about finding	-	-	-	7	3	0	1
Reject finding in current context	-	-	-	-	33	8	4
Tool is wrong	-	-	-	-	-	2	2
Unrelated	-	-	-	-	-	-	16

Table 5: Pair-wise comparison of the labels produced by the LLM in three different runs. The number in a cell means how often the two respective labels were used for the same conversation by two different runs of the LLM. So the top right cell means that, 9 times, one run of the LLM labels a conversation as *Accept both finding and remediation* and another run labeled it as *Unrelated*.

developer dialogues during, exploring areas like the consensus among human experts, their confidence in labeling decisions, and the stability of LLM’s results. Our findings revealed that while human experts frequently agreed on straightforward cases, they diverged in more complex scenarios, highlighting the subjective and context-dependent nature of human judgment.

We also examined human confidence in labeling decisions, revealing a significant shift in expert opinions after they were presented with LLM’s summaries and reasoning. In cases like Accept finding but reject remediation, experts often revised their initial labels, indicating that LLM’s justifications were persuasive enough to sway human decision-making. Conversely, the LLM in our study demonstrated high confidence in its labeling, showing a tendency to maintain its initial decisions even when alternative human-assigned labels were presented. This underscored the potential of LLMs as reliable tools for the categorization of developer conversations, especially in complex dialogue scenarios.

Overall, our experimental evaluation positions large language models as effective and promising tools for the automated classification of developer conversations in code reviews. This capability not only enhances efficiency but also improves accuracy in this labeling task, which can be laborious and susceptible to bias when conducted manually. With an acceptance of 88%, even our simple experimental setup is sufficiently accurate to be used to propose follow up actions, such as suppression of warnings by the static analyzer in a code review setting.

Threats to validity. We demonstrated the effectiveness of LLMs at evaluating the sentiment of conversations in a code review, but cannot make claims about the generalizability of this approach or the results because of the model and taxonomy specific to these experiments. In particular, we used a specific language model and acknowledge that these results may be different for other models (in fact, we expect these models to only become more performant over time). We also did not compare how the same approach would perform with fine tuning or in-context learning.

The taxonomy used for labeling and categorizing items is inherently subjective. Changes in the overall taxonomy, the labels, or the instructions could lead to changes in performance for both the language model and human evaluators. Further work is needed to examine these parameters to better understand the root causes

of variability and drivers of performance in both human and LLM evaluations.

However, this does not change our observation that LLMs can be used classify conversations between developers and static analysis tools. For Claude we obtained usable results and we can see that the LLM is more stable in its labels (less willing to change) than the human. Results may be better (or worse) with other models and can likely be improved if the model is fine tuned to this task.

6 CONCLUSION

We presented an experiment to assess the effectiveness, reliability, and adaptability of LLMs in categorizing complex dialogues during developer-analyzer interactions during code reviews. Our findings highlight the capability of LLMs in accurately categorizing developer conversations, offering a level of depth and precision that surpasses traditional sentiment analysis methods. This advantage is underscored by the fact that LLMs achieve this without necessitating extensive labeled training data.

The study also revealed the inherent variability and subjectivity in human labeling of developer conversations. While human experts often aligned on clear-cut scenarios, their interpretations diverged in more complex or nuanced dialogues. This variability was further emphasized when human judgments were compared with the classifications made by the LLM. In many instances, human experts reconsidered and changed their initial labels after being presented with the model’s summaries and rationale. This aspect of our findings sheds light on the potential biases and uncertainties inherent in manual labeling processes and underscores the benefits of utilizing LLMs in such tasks.

In conclusion, our research positions LLMs as effective tools for automating the classification of developer conversations in code reviews. They enhance efficiency and accuracy, offering consistency difficult to achieve manually. The capability of LLMs to interpret complex dialogues with minimal training opens new avenues in code quality assurance and software development processes.

REFERENCES

- [1] D. Singh, V. R. Sekar, K. T. Stolee, and B. Johnson, “Evaluating how static analysis tools can reduce code review effort,” in *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2017, pp. 101–105.
- [2] S. Panichella, V. Arnaudova, M. Di Penta, and G. Antoniol, “Would static analysis tools help developers with code reviews?” in *2015 IEEE 22nd International*

- 1161 *Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2015, pp. 161–170.
- 1162 [3] M. Harman and P. O’Hearn, “From start-ups to scale-ups: Opportunities and open
- 1163 problems for static and dynamic program analysis,” in *2018 IEEE 18th International*
- 1164 *Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2018, pp. 1–23.
- 1165 [4] C. Sadowski, J. van Gogh, C. Jaspan, E. Söderberg, and C. Winter, “Tricorder:
- 1166 Building a program analysis ecosystem,” in *Proceedings of the 37th International*
- 1167 *Conference on Software Engineering - Volume 1*, ser. ICSE ’15. IEEE Press, 2015,
- 1168 p. 598–608.
- 1169 [5] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros,
- 1170 A. Kamsky, S. McPeak, and D. Engler, “A few billion lines of code later: Using
- 1171 static analysis to find bugs in the real world,” *Commun. ACM*, vol. 53, no. 2, p.
- 1172 66–75, Feb. 2010. [Online]. Available: <https://doi.org/10.1145/1646353.1646374>
- 1173 [6] C. Calcagno, D. Distefano, J. Dubreil, D. Gabi, P. Hooimeijer, M. Luca, P. O’Hearn,
- 1174 I. Papakonstantinou, J. Purbrick, and D. Rodriguez, “Moving fast with software
- 1175 verification,” in *NASA Formal Methods*, K. Havelund, G. Holzmann, and R. Joshi,
- 1176 Eds. Cham: Springer International Publishing, 2015, pp. 3–11.
- 1177 [7] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, “Why don’t software
- 1178 developers use static analysis tools to find bugs?” in *Proceedings of the 2013*
- 1179 *International Conference on Software Engineering*, ser. ICSE ’13. IEEE Press,
- 1180 2013, p. 672–681.
- 1181 [8] C. Sadowski, E. Aftandilian, A. Eagle, L. Miller-Cushon, and C. Jaspan, “Lessons
- 1182 from building static analysis tools at google,” *Commun. ACM*, vol. 61, no. 4, p.
- 1183 58–66, Mar. 2018. [Online]. Available: <https://doi.org/10.1145/3188720>
- 1184 [9] L. Luo, R. Mukherjee, O. Tripp, M. Schäff, Q. Zhou, and D. Sanchez, “Long-term
- 1185 static analysis rule quality monitoring using true negatives,” in *2023 IEEE/ACM*
- 1186 *45th International Conference on Software Engineering: Software Engineering in*
- 1187 *Practice (ICSE-SEIP)*, 2023, pp. 315–326.
- 1188 [10] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis,
- 1189 L. Zettlemoyer, and V. Stoyanov, “Roberta: A robustly optimized BERT
- 1190 pretraining approach,” *CoRR*, vol. abs/1907.11692, 2019. [Online]. Available:
- 1191 <http://arxiv.org/abs/1907.11692>
- 1192 [11] D. V. Veen, C. V. Uden, L. Blankemeier, J.-B. Delbrouck, A. Aali, C. Bluethgen,
- 1193 A. Pareek, M. Polacin, W. Collins, N. Ahuja, C. P. Langlotz, J. Hom, S. Gatidis,
- 1194 J. Pauly, and A. S. Chaudhari, “Clinical text summarization: Adapting large
- 1195 language models can outperform human experts,” 2023.
- 1196 [12] P. Liang, R. Bommasani, T. Lee, D. Tsipras, D. Soylu, M. Yasunaga, Y. Zhang,
- 1197 D. Narayanan, Y. Wu, A. Kumar, B. Newman, B. Yuan, B. Yan, C. Zhang, C. Cos-
- 1198 grove, C. D. Manning, C. Ré, D. Acosta-Navas, D. A. Hudson, E. Zelikman, E. Dur-
- 1199 mus, F. Ladhak, F. Rong, H. Ren, H. Yao, J. Wang, K. Santhanam, L. Orr, L. Zheng,
- 1200 M. Yuksekgonul, M. Suzgun, N. Kim, N. Guha, N. Chatterji, O. Khattab, P. Henderson,
- 1201 Q. Huang, R. Chi, S. M. Xie, S. Santurkar, S. Ganguli, T. Hashimoto, T. Icard,
- 1202 T. Zhang, V. Chaudhary, W. Wang, X. Li, Y. Mai, Y. Zhang, and Y. Koreeda,
- 1203 “Holistic evaluation of language models,” 2023.
- 1204 [13] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee,
- 1205 Y. T. Lee, Y. Li, S. Lundberg *et al.*, “Sparks of artificial general intelligence: Early
- 1206 experiments with gpt-4,” *arXiv preprint arXiv:2303.12712*, 2023.
- 1207 [14] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakant-
- 1208 tan, P. Shyam, G. Sastry, A. Askell *et al.*, “Language models are few-shot learners,”
- 1209 *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.
- 1210 [15] OpenAI, “Gpt-4 technical report,” *arXiv preprint arXiv:2303.08774*, 2023.
- 1211 [16] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov,
- 1212 S. Batra, P. Bhargava, S. Bhosale *et al.*, “Llama 2: Open foundation and fine-tuned
- 1213 chat models,” *arXiv preprint arXiv:2307.09288*, 2023.
- 1214 [17] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang,
- 1215 S. Agarwal, K. Slama, A. Ray *et al.*, “Training language models to follow instruc-
- 1216 tions with human feedback,” *Advances in Neural Information Processing Systems*,
- 1217 vol. 35, pp. 27 730–27 744, 2022.
- 1218 [18] C. Vassallo, S. Panichella, F. Palomba, S. Proksch, H. Gall, and A. Zaidman, “How
- 1219 developers engage with static analysis tools in different contexts,” *Empirical*
- 1220 *Software Engineering*, vol. 25, 03 2020.
- 1221 [19] D. Singh, V. R. Sekar, K. T. Stolee, and B. Johnson, “Evaluating how static analysis
- 1222 tools can reduce code review effort,” in *VL/HCC*. IEEE, 2017, pp. 101–105.
- 1223 [20] S. Panichella, V. Arnaoudova, M. D. Penta, and G. Antoniol, “Would static analysis
- 1224 tools help developers with code reviews?” in *SANER*. IEEE, 2015, pp. 161–170.
- 1225 [21] R. Tufano, L. Pascarella, M. Tufano, D. Poshyvanik, and G. Bavota, “Towards
- 1226 automating code review activities,” in *ICSE*. IEEE, 2021, pp. 163–174.
- 1227 [22] A. Bacchelli and C. Bird, “Expectations, outcomes, and challenges of modern
- 1228 code review,” in *ICSE*. IEEE, 2013, pp. 712–721.
- 1229 [23] B. Johnson, Y. Song, E. R. Murphy-Hill, and R. W. Bowdidge, “Why don’t software
- 1230 developers use static analysis tools to find bugs?” in *ICSE*. IEEE, 2013, pp. 672–
- 1231 681.
- 1232 [24] C. Calcagno, D. Distefano, J. Dubreil, D. Gabi, P. Hooimeijer, M. Luca, P. W.
- 1233 O’Hearn, I. Papakonstantinou, J. Purbrick, and D. Rodriguez, “Moving fast with
- 1234 software verification,” in *NFM*, vol. 9058. Springer, 2015, pp. 3–11.
- 1235 [25] B. Lin, F. Zampetti, G. Bavota, M. D. Penta, M. Lanza, and R. Oliveto, “Sentiment
- 1236 analysis for software engineering: how far can we go?” in *ICSE*, M. Chaudron,
- 1237 I. Crnkovic, M. Chechik, and M. Harman, Eds. ACM, 2018, pp. 94–104.
- 1238 [26] N. Novielli and A. Serebrenik, “Sentiment and emotion in software engineering,”
- 1239 *IEEE Software*, vol. 36, no. 5, pp. 6–9, 2019.
- 1240 [27] M. Mäntylä, B. Adams, G. Destefanis, D. Graziotin, and M. Ortu, “Mining valence,
- 1241 arousal, and dominance: possibilities for detecting burnout and productivity?”
- 1242 in *MSR*. ACM, 2016, pp. 247–258.
- 1243 [28] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu,
- 1244 D. Jiang, and M. Zhou, “Codebert: A pre-trained model for programming and
- 1245 natural languages,” in *EMNLP*. Association for Computational Linguistics, 2020,
- 1246 pp. 1536–1547.
- 1247 [29] M. Pradel and K. Sen, “Deepbugs: a learning approach to name-based bug detec-
- 1248 tion,” *Proc. ACM Program. Lang.*, vol. 2, no. OOPSLA, pp. 147:1–147:25, 2018.
- 1249 [30] G. Pang, C. Shen, L. Cao, and A. van den Hengel, “Deep learning for anomaly
- 1250 detection: A review,” *ACM Comput. Surv.*, vol. 54, no. 2, pp. 38:1–38:38, 2022.
- 1251 [31] S. Haque, A. LeClair, L. Wu, and C. McMillan, “Improved automatic summariza-
- 1252 tion of subroutines via attention to file context,” in *MSR*, S. Kim, G. Gousios,
- 1253 S. Nadi, and J. Hejderup, Eds. ACM, 2020, pp. 300–310.
- 1254 [32] H. Alsolai and M. Roper, “A systematic literature review of machine learning
- 1255 techniques for software maintainability prediction,” *Inf. Softw. Technol.*, vol. 119,
- 1256 2020.
- 1257 [33] J. Saldaña, *The coding manual for qualitative researchers*. sage, 2021.