

Does AI Code Review Lead to Code Changes? A Case Study of GitHub Actions

Kexin Sun, Hongyu Kuang, Sebastian Baltes, Xin Zhou, He Zhang, Xiaoxing Ma, Guoping Rong, Dong Shao,
and Christoph Treude

Abstract—AI-based code-review tools automatically review and comment on pull requests to improve code quality. Despite their growing presence, little is known about their actual impact. We present a large-scale empirical study of 16 popular AI-based code-review actions for GitHub workflows, analyzing more than 22,000 review comments in 178 repositories. We investigate (1) how these tools are adopted and configured, (2) whether their comments lead to code changes, and (3) which factors influence their effectiveness. We develop a two-stage LLM-assisted framework to determine whether review comments are addressed. We then use interpretable machine learning to identify the influencing factors. We found that while adoption is growing, its effectiveness varies widely. Comments that are concise, contain code snippets, and are manually triggered, particularly those from hunk-level review tools, are more likely to result in code changes. These results highlight the importance of tool design and suggest directions for improving AI-based code review systems.

Index Terms—Code Review, GitHub Actions, Large Language Models, Empirical Software Engineering

I. INTRODUCTION

Code review is a widely adopted practice in modern software engineering, playing a central role in identifying bugs, improving code quality, and facilitating knowledge transfer within teams [1]. Traditionally, code reviews are performed manually through reviewing and commenting on code changes (submitted via pull requests) by developers [2]. However, recent advances in generative AI (GenAI) have created opportunities to automate at least part of this process by generating human-like code review comments [3].

On platforms such as GitHub, developers can now integrate GenAI into their workflows via GitHub Actions [4]. These AI-driven code review actions automatically analyse code changes and post suggestions as comments on pull requests. Some of these actions are now available and popular on the GitHub Marketplace, promising to reduce developers’ efforts and to increase review coverage [5]. Despite the growing presence, there is little empirical evidence of how these actions are actually used in practice and whether they can impact the development process meaningfully.

K. Sun, H. Kuang, X. Zhou, H. Zhang, X. Ma, G. Rong, and D. Shao are with State Key Lab for Novel Software Technology, Nanjing University, Nanjing, China (e-mail: kexinsun@smail.nju.edu.cn; khy@nju.edu.cn; zhouxin@nju.edu.cn; hezhang@nju.edu.cn; xxm@nju.edu.cn; ronggp@nju.edu.cn; dongshao@nju.edu.cn).

S. Baltes is with University of Heidelberg, Heidelberg, Germany (e-mail: sebastian.baltes@uni-heidelberg.de).

C. Treude is with School of Computing and Information Systems, Singapore Management University, Singapore (e-mail: ctreude@smu.edu.sg).

Corresponding author: Hongyu Kuang (e-mail: khy@nju.edu.cn).

Previous research has investigated human code review practices in detail [6], including (1) factors that make review comments useful [7]–[9], (2) the granularity of feedback [10], and (3) developer responsiveness [11]. In contrast, the dynamics of automated reviews powered by GenAI remain largely unexplored. Open questions include: Which AI-based tools are developers actually adopting? Do generated comments lead to meaningful code changes? Which characteristics influence whether such comments are taken seriously and acted upon?

This paper addresses these gaps through an empirical study of AI-based code review actions based on Large Language Models (LLMs) on GitHub. Our goal is to understand both the usage and the effectiveness of these tools, as well as the factors that influence their impact. Specifically, we propose the following three research questions:

RQ1: *How are LLM-based code review actions adopted in GitHub repositories?*

This question is motivated by the proliferation of AI-based review tools on GitHub. Although many tools exist, their real-world adoption and configuration practices are not well understood. Studying adoption patterns allows us to assess the maturity and utility of these tools in practical software development workflows.

RQ2: *To what extent do AI-generated review comments lead to code changes compared to human review comments?*

Our second question aims to understand whether AI-generated review comments can truly impact the code (i.e., lead to code changes) and how their effectiveness compares to human review comments. To enable large-scale analysis, we propose an LLM-based approach to automatically assess the validity of review comments and whether they had been addressed.

RQ3: *Which factors impact the likelihood that code review comments lead to code changes?*

Finally, we explore the contextual and technical factors that make comments more or less likely to result in code changes. By analyzing aspects such as comment content, contributor experience, and project characteristics, we aim at identifying patterns that can inform the design of more effective review tools.

To answer these questions, we conducted a large-scale empirical study of 16 popular GenAI-based code-review actions on GitHub. We analyze their adoption in 178 repositories, collect more than 22,000 review comments, and develop a two-stage LLM-based classification framework to assess whether these comments are addressed. Finally, we use interpretable

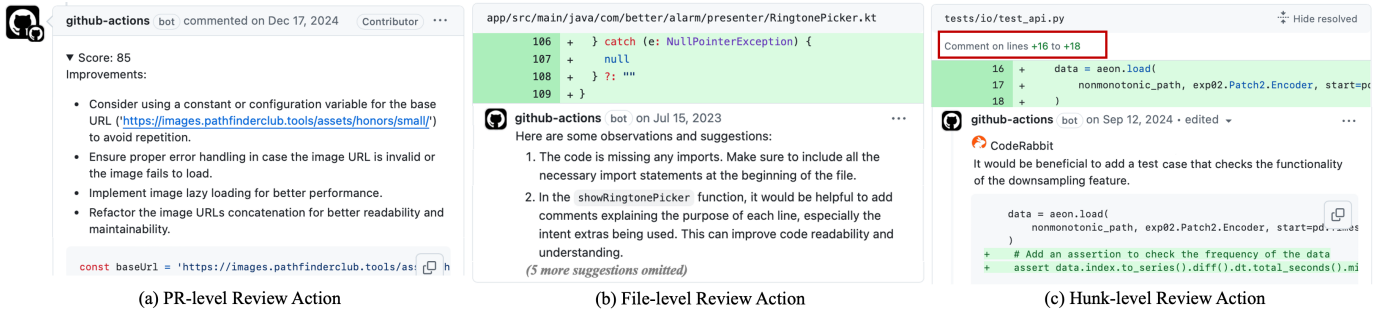


Fig. 1. Example comments from a **PR-level** review action (Integral-Healthcare/robin-ai-reviewer), a **file-level** review action (anc95/ChatGPT-CodeReview), and a **hunk-level** review action (coderrabbit/ai-pr-reviewer).

machine learning techniques to model and find out the factors that influence the effectiveness of comments.

Our contributions are as follows.

- We provide the **first systematic study** of the **adoption and usage** of **AI-based code review** actions on GitHub, categorizing tools by their review granularity (i.e., PR, file, or hunk) and behaviour (e.g., action triggers).
- We introduce an **LLM-assisted framework** for assessing whether code review comments are actionable and whether they have been **addressed**, achieving high accuracy compared to human annotations.
- We identify and explain the **factors** that influence whether **AI-generated comments** lead to **code changes**, offering design implications for code review tools.

Our findings offer insights into the role of GenAI in software engineering that are relevant to researchers and practitioners interested in automated software quality assurance.

II. BACKGROUND

GitHub Actions is a built-in automation platform on GitHub that allows developers to define custom workflows using reusable components called “actions”. This section uses anc95/ChatGPT-CodeReview, the most popular action in our dataset, as an example to demonstrate how AI-based code review can be integrated into a project’s workflow.

How to Configure an Action: Developers can define and manage project workflows through YAML files in the project’s `.github/workflows` directory. As Fig. 2 shows, in our example, we define a workflow named `Code Review`, containing a single job called `review` with one step that utilizes the main branch of anc95/ChatGPT-CodeReview for code analysis (`uses: anc95/ChatGPT-CodeReview@main`). Using the keyword ‘on’, we set this workflow to trigger automatically when pull requests are opened, reopened, or synchronized. The trigger conditions can be further restricted using ‘if’ conditions. For example, as shown in the commented-out section of Fig. 1, we can restrict the workflow to trigger only on pull requests labeled with ‘gpt review’. In our subsequent analysis, we refer to this conditional triggering as manual triggering because it requires developers to explicitly enable it. The example action requires two mandatory parameters: `GITHUB_TOKEN` for repository access and `OPENAI_API_KEY` for LLM authentication. Additionally, we can customize its

```
name: Code Review ← Name of the workflow
on: pull_request: types: [opened, reopened, synchronize]
jobs:
  review: ← Name of the job
    # Optional; to run only when a label is attached
    # if: ${{ contains(github.event.*.labels.*.name, 'gpt review') }}
    runs-on: ubuntu-latest # Target execution environment for the job
    steps:
      - uses: anc95/ChatGPT-CodeReview@main
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
          OPENAI_API_KEY: ${{ secrets.OPENAI_API_KEY }}
          # Optional
          OPENAI_API_ENDPOINT: https://api.openai.com/v1
          MODEL: gpt-3.5-turbo # LLM used for review
          temperature: 1 # Temperature for LLM
          PROMPT: "Please check if there are any confusions in the following code diff:"
          LANGUAGE: Chinese # Target natural language for review output
          IGNORE_PATTERNS: /node_modules/**/*.*.txt # Pattern to ignore files
```

Fig. 2. An example of **configuring** an AI-based code review action.

behavior through optional parameters, such as which LLM to use (`MODEL`), which prompt to apply (`PROMPT`), and which natural language to comment in (`LANGUAGE`).

How the Action Works: Once the trigger conditions are satisfied (e.g., a pull request is opened), ChatGPT-CodeReview conducts a code review. It compares the pull request’s base commit and the latest commit (the head commit at that time) to obtain the cumulative code changes, then reviews these changes file by file. During this process, certain files may be skipped based on predefined ignore patterns, and the large change blocks may be divided according to the configured token limits. After the LLM analyzes a file’s changes via the designed prompt, ChatGPT-CodeReview posts an inline comment to the pull request via the GitHub Action bot. This comment is typically attached to the last line of the final changed block within the file. Fig. 1(b) provides an example of a review comment generated by anc95/ChatGPT-CodeReview.

III. STUDY SETUP

To investigate AI-based code review on GitHub, we collected the 16 most popular actions as our study sample and conducted a categorization based on their review behaviors. We provide an online appendix [12], including our dataset, annotations, and scripts for LLM-assisted analysis of whether review comments have been addressed (for RQ2), and for the interpretation of how factors impact the likelihood of comment addressing (for RQ3). We follow the guidelines proposed by Wagner et al. [13], [14] for empirical studies in software engineering involving LLMs.

TABLE I
COMPARISON OF DESIGN CHOICES FOR SELECTED AI-BASED CODE REVIEW ACTIONS ACROSS EACH GRANULARITY CATEGORY (ID-11: INTEGRAL-HEALTHCARE/ROBIN-AI-REVIEWER FOR **PR-LEVEL**, ID-1: ANC95/CHATGPT-CODEREVIEW FOR **FILE-LEVEL**, ID-3: CODERABBITAI/AI-PR-REVIEWER FOR **HUNK-LEVEL**). LLM SELECTION IS THE DEFAULT MODEL IN THE LATEST VERSION.

ID	Action Description	Input Change Scope	Processing Granularity	LLM Selection	Prompt Context	Commenting Format
11	PR-level code review	Changes between PR’s base and latest commit.	PR-level	gpt-4-turbo	(<i>Only diff, no additional</i>)	General comment
1	File-level code review	Optional based on PR event: If <i>synchronize</i> : Only Changes in the latest commit. If <i>others</i> : Changes between PR’s base and latest commit.	File-level	gpt-4o-mini	(<i>Only diff, no additional</i>)	Inline comment
3	Hunk-level code review (also: generating PR descriptions, code change summaries, and direct interaction with the LLM)	Changes between PR’s the last reviewed commit and the latest commit.	File-level	gpt-4	PR title; PR description; LLM’s change summary	Inline comment

Action Selection: On January 7, 2025, we began our selection process by manually examining the names and descriptions of the top-ranked code review actions on the GitHub Marketplace (sorted by popularity). From a review of approximately 240 actions, we identified an initial set of 20 candidates related to AI-driven code review. We then carefully checked their project documentation and excluded four actions that focused on tasks other than comment generation (e.g., PR description generation). This process yielded a final set of 16 relevant actions. In this paper, we refer to each action by its repository’s full name and report its star count in Table II.

Action Categorization: Although designed for the same purpose, these actions exhibit distinct review behaviors. We group them into three categories based on the granularity of their feedback: **PR-level**, **file-level**, and **hunk-level**, as shown in Table II (“*hunk*” refers to a contiguous block of differing lines when comparing two versions of a file [15]). These levels reflect increasing ability to provide context-specific comments. **Hunk-level** review actions most closely resemble typical human code reviews. They can provide targeted comments on specific code lines within files. For example, in Fig. 1(c), coderabbitai/ai-pr-reviewer suggests adding an assertion to check data frequency after lines 16-18 in tests/io/test_api.py. **File-level** review actions are not explicitly associated with specific lines but can still associate comments with the certain file. As exemplified by the previously discussed anc95/ChatGPT-CodeReview, it attaches its reviews of the entire file’s changes to the end of the overall file diff. Consequently, developers must manually map suggestions to specific code blocks within files to comprehend the recommended modifications. **PR-level** review actions represent the coarsest granularity, lacking file-specific comment linking. Fig. 1(a) illustrates this with a comment from Integral-Healthcare/robin-ai-reviewer, which discusses changes spanning multiple files within a single review message.

Beyond review granularity, actions also differ in how they implement the review process. These differences include the input change scope (which commits are compared), the processing granularity (how the changes are segmented), the LLM selection (which model is used), the prompt context (what additional information is provided), and the commenting format (how the review is delivered). To illustrate the variation, we present an example action for each of the three

TABLE II
OVERVIEW OF 16 SELECTED AI-BASED CODE REVIEW ACTIONS.

ID	Action	Stars	Review Granularity		
			PR-level	File-level	Hunk-level
1	anc95/ChatGPT-CodeReview	4.1k		✓	
2	mattzcarey/code-review-gpt	1.7k		✓	
3	coderabbitai/ai-pr-reviewer	1.6k			✓
4	aidar-freede/ai-codereviewer	704			✓
5	kxxt/chatgpt-action	558	✓		
6	cirolini/genai-code-review	339	✓		
7	truongnh1992/gemini-ai-code-reviewer	101			✓
8	feiskyer/ChatGPT-Reviewer	70	✓		
9	adshao/chatgpt-code-review-action	69	✓		
10	tmokms/bedrock-pr-reviewer	61			✓
11	Integral-Healthcare/robin-ai-reviewer	59	✓		
12	presubmit/ai-reviewer	53			✓
13	gvasilei/AutoReviewer	42		✓	
14	unsafecoerce/chatgpt-action	39			✓
15	magnificode-ltd/chatgpt-code-reviewer	27		✓	
16	ca-dp/code-butler	23	✓		

granularity categories in Table I. A full comparison of 16 actions can be found in our online appendix. We observe that 6 out of the 16 actions incorporate related tasks to support code review, such as coderabbitai/ai-pr-reviewer’s PR summarization and interactive feedback. For the input scope, 11 actions review the full base-to-latest commit diff on each run, while 3 actions (e.g., coderabbitai/ai-pr-reviewer) track reviewed commits to avoid redundancy, and 2 actions, including anc95/ChatGPT-CodeReview, dynamically adjust the input scope based on PR events or pre-configured modes. Processing granularity does not always align with the review granularity. For instance, coderabbitai/ai-pr-reviewer submits the file-level diff into the LLM in one request, but can leverage prompt constraints (e.g., “*Comment with exact line number*”) to extract multiple line-specific comments. Most actions (13/16) rely on OpenAI’s model family [16] for review, with the remaining using alternatives such as Gemini [17] or Claude [18]. In addition, 11 actions enrich the prompts with contextual information, such as the PR title, description, or programming language, alongside code changes. Finally, actions offering file- or hunk-level review typically post feedback as inline comments (usually for detailed code review), while PR-level actions post as PR general comments (usually for broad discussion). These varying design decisions will shape the developer experience and responses to the generated code review comments.

TABLE III
CATEGORIES OF OPTIONAL CONFIGURATION PARAMETERS IN AI-BASED CODE REVIEW ACTIONS WITH DESCRIPTIONS AND EXAMPLES

Configuration Category	Description and Example
Task Triggers & Modes	Defines when and how the action executes. E.g., <code>review_per_file</code> in feiskyer/ChatGPT-Reviewer (enables file-level review if true).
Input Settings	Excludes/includes specific files or limits code diff size. E.g., <code>IGNORE_PATTERNS</code> in aidar-freedom/ai-codereviewer (specifies file exclusion patterns).
LLM Service Settings	Defines parameters for connecting to the LLM service. E.g., <code>OPENAI_API_BASE</code> in Integral-Healthcare/robin-ai-reviewer (URL of the OpenAI API interface).
LLM Selection	Specifies the LLM used for review. E.g., <code>MODEL</code> in anc95/ChatGPT-CodeReview (name of the LLM).
LLM Hyperparameters	Configures generation behavior (e.g., randomness, length). E.g., <code>temperature</code> in feiskyer/ChatGPT-Reviewer (controls output randomness).
Prompt Customization	Defines the instructions given to the LLM. E.g., <code>PROMPT</code> in ca-dp/code-butler (custom instructions for review).
Prompt Context Augmentation	Additional context for prompt construction. E.g., <code>NLP_LANGUAGE</code> in anc95/ChatGPT-CodeReview (specifies output language).
Output Settings	Controls review output formatting or logic. E.g., <code>review_comment_lgtm</code> in coderabbitai/ai-pr-reviewer (posts comments even if no issues are found if true).
Others	Miscellaneous settings unrelated to the core review process. E.g., <code>debug</code> in unsafecoerce/chatgpt-action (enables debug mode if true).

IV. ANALYSES & RESULTS

Based on the 16 collected actions, in this section, we detail the methodology and results for each of our research questions.

A. *RQ1: How are LLM-based code review actions adopted in GitHub repositories?*

Approach: To answer this question, we examined the adoption of these actions by measuring their usage across repositories, pull requests, and comments. Furthermore, we analyzed their configuration settings to understand how developers adapt them to their specific project needs.

We utilized the GitHub REST API to collect relevant data. First, we identified repositories containing configuration files that invoke the target actions. Specifically, we retrieved the `.yml` and `.yaml` files in `.github/workflows` directories of GitHub repositories that explicitly referenced the target actions (i.e., containing `uses:ACTION` in uncommented sections). These files were then grouped by repository. To ensure a unique mapping between repository, configuration file, and action, we filtered out 19 repositories that referenced multiple target actions or the same action across different files. Following prior work [19], our analysis was limited to repositories with at least 50 PRs to ensure a minimum level of project maturity and popularity of PRs. Next, we quantified action-generated comments by examining PRs in these repositories. Using GitHub’s search feature, we identified relevant PRs with the queries `repo:{repo_name} reviewed-by:github-actions[bot] is:pr` for inline comments and `repo:{repo_name} commenter:github-actions is:pr` for general comments. Subsequently, we retrieved specific comments posted by `github-actions[bot]` (the account associated with comments generated by a GitHub Action) via the REST API endpoints: `pulls/{pull_number}/comments` for inline comments and `issues/{pull_number}/comments` for general comments.

Beyond adoption, we investigated how developers configure these actions, particularly focusing on optional parameters that reflect developer intent beyond mandatory execution requirements, and how these configurations evolve. We extracted optional parameters from each action’s documentation, noting their considerable diversity (in both number and function). For example, `coderabbitai/ai-pr-reviewer` exposes 20 optional parameters to support its complex review workflows, whereas `presubmit/ai-reviewer` only defines three mandatory parameters. To facilitate comparison, the first author

TABLE IV

RQ1: ADOPTION STATISTICS OF 16 AI-BASED REVIEW ACTIONS, GROUPED BY PR/FILE/HUNK REVIEW ACTIONS. “TOTAL/MATURE REPOS” COUNTS REPOSITORIES CONFIGURING THE ACTION (MATURE: ≥ 50 PRS). THE OBSERVED ACTIVITY COLUMNS DETAIL THE ADOPTION DISTRIBUTION ACROSS REPOS, PRS, AND COMMENTS IN MATURE REPOS. ASTERISK (*) INDICATES GENERAL INSTEAD OF INLINE COMMENTS.

ID	Total Repos	Mature Repos	Observed Review Activity		
			Repos	PRs	Comments
PR-level review actions:					
5	7	2	0	0	0
6	4	1	0	0	0
8	3	1	1	24	37*
9	5	1	0	0	0
11	8	4	1	38	38*
16	13	5	5	74	133*
Total	40	14	7	136	208
File-level review actions:					
1	460	114	74	2,238	19,549
2	37	11	9	421	421*
13	3	1	0	0	0
15	4	1	1	1	3
Total	504	127	84	2,660	19,973
Hunk-level review actions:					
3	108	27	14	171	1,911
4	47	8	5	28	193
7	5	0	0	0	0
10	6	1	1	4	31
12	6	1	1	3	10
14	2	0	0	0	0
Total	174	37	21	206	2,145
All 16	718	178	112	3,002	22,326

manually grouped all optional parameters into nine functional categories, summarized in Table III. This grouping was reviewed by one co-author. For each repository, we considered the workflow configuration file at its latest commit as the final configuration and used PyYAML to parse configurations according to our nine categories. To explore the evolution of the configuration, we calculated the number of commits and the time interval between the initial commit (introducing the action) and the latest commit. If modifications occurred, we further compared specific settings in the initial and final configurations to identify frequently adjusted parameters.

Results: Table IV summarizes the usage of the 16 AI-based code review actions. Out of 718 matched repositories, 178 met the maturity criterion (≥ 50 PRs), and these contained a total of 22,326 AI-generated review comments. We found that 37.1% of the mature repositories declared an action but showed no generated comments, indicating a gap between declaration and actual use. Among the remaining repositories, usage was highly concentrated on the four most popular actions. Actions ID-1 to ID-4 (including two file-level and two hunk-level

TABLE V

RQ2: REFINED DATASET FOR COMMENT ADDRESSING ANALYSIS (N=5,652): POST-REVIEW FILE CHANGE DISTRIBUTION BY COMMENT SOURCES.

ID	Action	Total Comments	Modified	Renamed-Modified	Renamed-Only	Deleted	Unchanged
1	anc95/ChatGPT-CodeReview	2,831	436 (15.40%)	9 (0.32%)	0 (0.00%)	2 (0.07%)	2,384 (84.21%)
2	mattzcarey/code-review-gpt	773	20 (2.59%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	753 (97.41%)
-	File-level Review Actions Total	3,604	456 (13.89%)	9 (0.27%)	0 (0.00%)	2 (0.06%)	3,137 (85.78%)
3	coderrabbitai/ai-pr-reviewer	713	303 (42.50%)	21 (2.95%)	3 (0.42%)	10 (1.40%)	376 (52.73%)
4	aidar-freedom/ai-codereviewer	169	84 (49.70%)	0 (0.00%)	0 (0.00%)	3 (1.78%)	82 (48.52%)
-	Hunk-level Review Actions Total	882	387 (43.88%)	21 (2.38%)	3 (0.34%)	13 (1.47%)	458 (51.93%)
-	Human Review Total	1,166	988 (84.75%)	44 (3.77%)	0 (0.00%)	38 (3.26%)	96 (8.23%)

reviewers) accounted for 91.1% of the reviewed repositories, 95.2% of the pull requests, and 98.9% of the generated comments. This was largely driven by the widespread use of ID-1 anc95/ChatGPT-CodeReview. In contrast, PR-level review actions saw limited adoption and impact.

Regarding configuration evolution, the majority of the mature repositories (147/178, 82.6%) customized at least one optional parameter. The most frequently configured options were *Prompt Context Augmentation* (103/147, 70.1%), primarily for specifying the natural language of review comments, *LLM Selection* (95, 64.6%), and *Input Settings* (70, 47.6%). On average, these 178 repositories made changes to the configuration file 4 times after the initial setup. Of these, 68 repositories (38.2%) retained their original configurations without further changes. Temporal analysis revealed that 50 projects (28.1%) finalized modifications within one week, and 20 (11.2%) within one month, while 40 repositories (22.5%) continued adjustments beyond one month. The most frequently modified parameters were *LLM Selection* (25, 22.7% of 110 repositories that made post-setup changes), *Prompt Customization* (21, 19.1%), and *LLM Hyperparameters* (13, 11.8%). Beyond action-specific options, we also observe that 50 repositories (45.5%) adjusted their workflow-level triggers by modifying the on or if conditions. Additionally, 12 repositories (10.9%) updated the action reference, such as replacing @main with a specific release tag.

Answer to RQ1: We analyzed 178 mature repositories and found a total of 22,326 AI-generated review comments. Among the 178 repositories, 82.6% customized at least one optional parameter, showing that developers often go beyond default settings when using these actions.

B. RQ2: To what extent do AI-generated review comments lead to code changes compared to human review comments?

Approach: To assess the practical impact of these actions, we leverage LLMs to extensively examine whether the generated review comments led to actual code changes (which we define as “addressed”). We use the term “comment addressing analysis” to refer to our analysis of whether comments have been addressed. We first created the “comment addressing dataset” (filtered from RQ1 and augmented with human-written review comments) to carry out this analysis. Next, we sampled a subset to manually annotate whether they had been addressed and evaluated LLM performance on this subset. Finally, we employed the best-performing LLM approach across the entire dataset. Figure 3 presents an overview of this workflow.

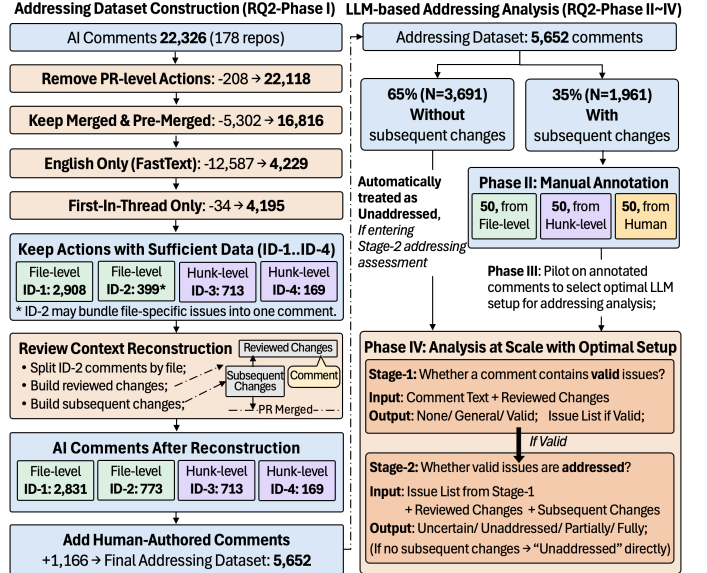


Fig. 3. Overview of the RQ2 comment addressing analysis pipeline.

Phase I: Refining the Subset for Comment Addressing Analysis. We first excluded comments from PR-level actions because of their significant differences from typical human code review practices and their low volume. For comments from file-level and hunk-level actions, we only retained those in merged PRs (and prior to merging) to capture finalized developer decisions, yielding 16,816 comments. To facilitate human analysis, we selected only English comments using FastText, reducing the dataset to 4,229 comments (a removal of 12,587 comments). ID-1 (anc95/ChatGPT-CodeReview) saw a significant reduction in comments at this stage, with 11,744 comments (mostly in Korean) filtered out. We further discuss potential threats introduced by this language-based filtering in Section VI. Given that some actions enable user-LLM dialogue within threads, we restricted our analysis to first-in-thread comments to focus on the original review suggestions rather than follow-up interactions (excluding an additional 34 comments). After these filters, only Actions ID-1 to ID-4 retained sufficient data for the addressing analysis, with 2,908, 399, 713, and 169 comments respectively.

Next, we constructed the necessary context for assessing comment addressing, which included the originally reviewed changes and subsequent code changes within the reviewed file. For hunk-level reviews, which were all submitted as inline comments, GitHub’s API provides sufficient metadata to reconstruct the reviewed code. Specifically, `original_commit_id` identifies the commit under

review, `original_start_line` and `original_line` specify the range of reviewed lines, and `diff_hunk` captures the change block where the comment was made. We extracted the reviewed code by slicing `diff_hunk` based on the recorded line range. For single-line comments (with only `original_line`), we additionally included the three preceding lines to align with GitHub’s display convention for diffs. File-level actions required additional handling, as they often cover multiple changes within a single comment. For Action ID-1 (anc95/ChatGPT-CodeReview), we simulated its two strategies to define the reviewed changes: (1) directly extracting the file’s changes from the `original_commit_id`, and (2) computing cumulative diffs between the PR’s base commit and `original_commit_id` via GitHub’s `/compare` endpoint. We retained only those comments whose `diff_hunk` could match as a subset of either reconstructed change set, and discarded inconsistent cases potentially caused by temporal shifts in the PR’s base commit. For Action ID-2 (mattzcarey/code-review-gpt), which posts general comments (without `original_commit_id`) reviewing multiple files, we first split comments by file, then approximated the reviewed commit as the one immediately preceding the comment’s creation. We reconstructed the reviewed changes by comparing this commit against the PR’s base commit.

For subsequent changes within reviewed files, we defined these as cumulative changes between the reviewed and merged versions. We retrieved file content at `original_commit_id` and at the merge commit using `contents/{file_path}?ref={sha}` endpoint, then calculated their textual differences using the `difflib` Python library. We chose this approach over the `/compare` endpoint to avoid exaggerating rebase-related changes. We also accounted for file renaming by resolving to their final paths. Subsequent changes were categorized as follows: (1) **Modified** (content changed under original filename); (2) **Renamed-Modified** (filename changed and content modified); (3) **Renamed-Only** (filename changed with no content modified); (4) **Deleted** (file removed); (5) **Unchanged** (no modifications or renaming occurred). Table V shows the distributions of subsequent changes by comment source. In total, we obtained 4,486 action-generated comments from 51 repositories.

For comparison, we collected human-authored review comments from the same 51 repositories for which we had obtained action-generated comments in Phase I. For each repository, we defined its AI review activation period as the time span between the first and last action-generated comment. We restricted our collection of human-written comments to this same time window to minimize potential effects on the addressing ratio (e.g., differences in the project’s development phase). Within this period, we collected human-authored inline comments that met the same filtering criteria as the action-generated comments: (1) from merged pull requests and posted prior to merging, (2) written in English, and (3) first in their comment threads (this filter excluded nearly half of the candidate comments). We also excluded comments from accounts whose login names contain the string “bot” [20]. This process yielded a final set of 1,166 human-authored review comments. Following the same procedure used for

hunk-level review actions, we reconstructed their originally reviewed changes and subsequent file modifications. Table V summarizes the statistics for the human comments. In total, our comment addressing dataset includes 5,652 review comments: 3,604 from file-level review actions, 882 from hunk-level actions, and 1,166 authored by human reviewers.

Phase II: Establishing Ground Truth via Human Annotation. In this phase, we sampled a subset and manually evaluated whether they had been addressed to serve as the ground truth. Specifically, we randomly sampled 50 comments from each category where subsequent file modifications occurred (i.e. 467 eligible from file-level actions, 424 eligible from hunk-level actions, and 1,070 eligible human-authored comments), yielding 150 comments in total. Before the full annotation, two authors first established an annotation guideline through consensus on 9 comments (the first 3 from each category). Through this process, we identified two challenges that prevent comment addressing from being a simple binary classification. First, some comments did not contain any suggestions, such as simple praise in human reviews (e.g., “Amazing!”), or mere code summaries in action-generated reviews. Some comments contained overly general suggestions lacking clear guidance (e.g., “Suggest to test thoroughly”), making it difficult to determine whether they were addressed. Second, a single comment may include multiple issues or suggestions, requiring a more granular assessment of their resolution. Therefore, we designed a two-stage annotation task. In **Stage-1**, we examined the comment text and the reviewed changes to determine whether a comment contained valid (i.e. specific and actionable) issues or suggestions, resulting in the following classifications:

- **None:** Does not contain any issues/suggestions.
- **General:** Only contains overly general issues/suggestions.
- **Valid:** Contains at least one valid issues/suggestions.

In **Stage-2**, for the comments classified as “Valid” in Stage-1, we further analyzed subsequent code changes to determine the extent to which the identified issues or suggestions were addressed. The possible outcomes were:

- **Valid-Uncertain:** Insufficient information to determine (mainly caused by file deletion or large-scale refactoring).
- **Valid-Unaddressed:** None of identified items addressed.
- **Valid-Partially:** Some but not all items addressed (at least one of them was completely addressed).
- **Valid-Fully:** All items addressed.

Following this guideline, the remaining 141 comments (47 per category) were independently labeled by one author and an external graduate student in software engineering who is not a co-author of this paper. Inter-rater agreement (Cohen’s κ) reached substantial levels over the full 6-class annotation scheme (i.e., “None”, “General”, “Valid-Uncertain”, “Valid-Unaddressed”, “Valid-Partially”, “Valid-Fully”): 0.674 for file-level actions, 0.734 for hunk-level actions, and 0.764 for human reviews. A third author resolved disagreements by selecting between two annotations or making a new judgment when necessary. Table VI shows the final label distribution across 150 annotated comments.

TABLE VI

RQ2: ANNOTATION LABEL DISTRIBUTION BY COMMENT SOURCES.

Source	Total	Invalid		Valid, Not Addressed		Valid, Addressed	
		None	General	Valid-Uncertain	Valid-Unaddressed	Valid-Partially	Valid-Fully
File-level Action	50	2.0%	30.0%	0.0%	42.0%	22.0%	4.0%
		32.0%		42.0%		26.0%	
Hunk-level Action	50	8.0%	8.0%	6.0%	42.0%	2.0%	34.0%
		16.0%		48.0%		36.0%	
Human	50	12%	0.0%	6.0%	2.0%	2.0%	78.0%
		12.0%		8.0%		80.0%	

TABLE VII

RQ2: PERFORMANCE ON 150 ANNOTATED COMMENTS USING THE BEST AUTOMATED SETUP, WITH *openai-gpt-4.1* FOR STAGE-1 AND *openai-o3-mini* FOR STAGE-2. THE LAST COLUMN REPORTS THE ABILITY TO DISTINGUISH “VALID, ADDRESSED” COMMENTS FROM ALL OTHERS. “AVG.”: AVERAGE ACROSS THREE SOURCES.

Source	Full 6-class		Stage-1 (Validity)		Stage-2 (Addressing)		Valid, Addressed vs. All Others	
	OA	Macro-F1	OA	Macro-F1	OA	Macro-F1	OA	Macro-F1
File-level Action	83.6%	78.1%	94.4%	93.5%	88.5%	88.4%	92.0%	90.5%
Hunk-level Action	89.2%	83.4%	95.2%	90.8%	93.2%	93.1%	91.6%	91.2%
Human	85.6%	62.4%	94.0%	88.0%	93.7%	83.9%	90.8%	87.6%
Avg.	86.1%	74.6%	94.5%	90.8%	91.8%	88.5%	91.5%	89.8%

TABLE VIII

RQ2: PERFORMANCE OF DISTINGUISHING “VALID, ADDRESSED” COMMENTS FROM OTHERS ACROSS THE ENTIRE DATASET. UNCHANGED GROUP IS LOGICALLY CLASSIFIED AS NON-ADDRESSED (100% ACC.).

Source	No Code Change		With Code Change		Overall	
	Total	Acc.	Total	Acc.	Total	Acc.
File-level Action	3,137	100%	467	92.0%	3,604	99.0%
Hunk-level Action	458	100%	424	91.6%	882	96.0%
Human	96	100%	1,070	90.8%	1,166	91.6%
Total	3,691	100%	1,961	91.3%	5,652	97.0%

Phase III: Evaluating LLM Effectiveness for Comment Addressing Analysis. Based on our human-annotated dataset, we evaluated the performance of three families of LLMs for addressing analysis: OpenAI [16] (*gpt-4.1*, *gpt-4o*, and *o4-mini* / *o3-mini* with medium reasoning effort), Claude [18] (*claude-3-sonnet* and *claude-3-haiku*), and DeepSeek [21] (*deepseek-r1* and *deepseek-v3*). The specific API endpoints and model versions used are documented in our online appendix scripts; the evaluations were performed in April and May 2025. Following the human annotation process, we designed separate prompts for Stage-1 and Stage-2. Stage-1 aimed to classify comments as “None”, “General”, or “Valid”, and extract a list of valid issues or suggestions for “Valid” comments. When Stage-1 identified a comment as “Valid”, Stage-2 was triggered to determine its final addressed status. Both stages utilized the same contextual information (i.e. reviewed changes and subsequent file modifications) and the classification scheme as the human annotators. The details of the LLM-assisted framework with specific prompts are available in the online appendix for other researchers to use.

We decoupled the stages to allow different model selection based on stage-specific performance. With temperature set to 0, we ran each model five times for robust evaluation. Performance was measured using overall accuracy (OA) and Macro-F1 scores. In the first round of evaluation, we used the

TABLE IX

RQ2: LLM-ASSIGNED ADDRESSING LABELS FOR COMMENTS BY SOURCE (ID-1: ANC95/CHATGPT-CODEREVIEW, ID-2: MATTZCAREY /CODE-REVIEW-GPT, ID-3: CODERABBITAI/AI-PR-REVIEWER, ID-4: AIDAR-FREED/AI-CODEREVIEWER, HUMAN REVIEWS). DISTRIBUTION SHIFT COMPARED WITH TABLE VI RESULTS FROM A DIFFERENT SAMPLING STRATEGY (SEE RQ2’S RESULTS).

Source	Total	Invalid		Valid, Not Addressed		Valid, Addressed	
		None	General	Valid-Uncertain	Valid-Unaddressed	Valid-Partially	Valid-Fully
File-level Review Action:							
ID-1	2,831	15.7%	24.8%	0.2%	55.2%	3.7%	0.5%
		40.4%		55.4%		4.2%	
ID-2	773	1.6%	18.1%	0.1%	79.3%	0.8%	0.1%
		19.7%		79.4%		0.9%	
Hunk-level Review Action:							
ID-3	713	15.4%	7.4%	2.2%	55.7%	4.2%	15.0%
		22.9%		57.9%		19.2%	
ID-4	169	3.6%	2.4%	2.4%	85.2%	0.6%	5.9%
		5.9%		87.6%		6.5%	
Human	1,166	15.9%	3.7%	4.2%	16.3%	4.0%	56.0%
		19.6%		20.5%		60.0%	

same LLM for both Stage 1 and Stage 2. Beyond assessing performance on the final 6-class scheme, we also evaluated the models’ effectiveness in each stage independently. For Stage-1 (detecting valid comment), we grouped “None” and “General” into a “Not Valid” category, and reported the models’ ability to distinguish between “Not Valid” and “Valid” comments. For Stage-2 (addressed assessment), focusing on comments correctly identified as “Valid”, we grouped “Valid-Uncertain”, “Valid-Unaddressed” into “Valid, Not Addressed,” and “Valid-Partially” and “Valid-Fully” into “Valid, Addressed”, then reported the models’ ability to differentiate these two outcomes. In the second round of evaluation, we cross-combined the top three models from each stage to find the optimal combination.

Phase IV: LLM-Based Labeling of Comment Addressing. Finally, we apply the best-performing setup determined in Phase III to conduct comment addressing analysis on the entire dataset of 5,652 review comments.

Results: We first present the evaluation results from Phase III. In our first round of evaluation, we observed that general-purpose models usually performed well in Stage-1, while reasoning-enhanced models excelled in Stage-2. The top performers for Stage-1 were *gpt-4.1* (94.5% average overall accuracy), *deepseek-v3* (94.0%), and *claude-3-sonnet* (93.2%). For Stage-2, *deepseek-r1* led with 95.4%, followed by *openai-o3-mini* (92.4%) and *o4-mini* (91.3%). Table VII reports the performance of our optimal cross-combined setup: *gpt-4.1* for Stage-1 and *o3-mini* for Stage-2, which achieved strong and balanced performances across three comment sources, with an average of 86.1% overall accuracy and 74.6% Macro-F1 score under the full 6-class scheme. The lower macro-F1 for human-authored comments (62.4%) reflects stronger class imbalance within that subset (see Table VI), which penalizes macro-averaged metrics. Importantly, performance on the primary downstream distinction (“Valid, Addressed” vs. all others) remains high, with an average of 91.5% overall accuracy and 89.8% macro-F1. Furthermore, when analyzing the complete dataset, we found that a significant portion of

comments (87.0% of file-level, 51.9% of hunk-level, and 8.2% of human comments) had no subsequent modifications. We can thus trivially classify them as non-“Valid, Addressed” with certainty. Considering the natural class imbalance in real-world review data, the overall classification accuracy across the full dataset is 97%, ensuring the reliability of our subsequent large-scale analysis in Phase IV (detailed breakdown in Table VIII).

Following the best-performing setup on the annotated dataset, we used *gpt-4.1* to assess the validity of all 5,652 comments, identifying 3,955 as valid, and then used *openai-o3-mini* to assess whether these valid comments had been addressed. Table IX details the results across the entire dataset. Comparing Table VI (manual labels) and Table IX (automated labels), we observe a discrepancy between their annotation distributions. This stems from our sampling strategy. For the manual annotation, we purposefully sampled only those comments where the reviewed file was subsequently modified. This was a necessary precondition for assessing if a comment was addressed, but it naturally biased the sample toward “Valid” and “Addressed” classifications. In contrast, the automated annotation was applied to the entire comment dataset, which included a large volume of comments (65.3%) with no subsequent file changes. Following our procedure, if these comments were identified as “Valid” in Stage-1, they were automatically classified as “Not Addressed” in Stage-2. Consequently, the proportion of “Valid, Addressed” comments is lower in the automated setting. Beyond this discrepancy, both tables indicate that the addressing rate of AI-generated review comments (0.9%–19.2% in Table IX) still lags behind human review comments (60%). Overall, hunk-level review actions exhibit a higher addressing rate (6.5%–19.2%) compared to file-level actions (0.9%–4.2%).

Answer to RQ2: We found that **hunk-level review actions (6.5%–19.2%)** exhibit a higher addressing rate compared to **file-level actions (0.9%–4.2%)**, yet the addressing rate of AI-generated review comments still lags behind **human review comments (60%)**.

C. RQ3: Which factors impact the likelihood that code review comments lead to code changes?

Approach: To answer RQ3, we conducted an interpretable analysis to identify the factors influencing whether review comments lead to code changes. We focus specifically on the 3,879 valid comments from RQ2 that have a definitive addressing label (excluding 76 uncertain cases), examining the factors that determine whether they are addressed. For invalid comments, we discuss their potential causes and mitigation strategies in Section V. To quantify the impact of different factors, we engineered a structured feature set covering various dimensions that potentially affect response behavior. Since direct interpretation of LLM decision-making is challenging, we trained a Random Forest classifier to fit the addressing results derived in RQ2 and employed SHAP (SHapley Additive exPlanations) [22] analysis to interpret the influence and directionality of each feature on the model’s predictions.

TABLE X
RQ3: 36 SELECTED FEATURES ACROSS FOUR DIMENSIONS USED FOR ANALYZING FACTORS IMPACTING COMMENT ADDRESSING.

Source Features (10): Who generated the comment and how it was generated	
Is_Human	1 if written by a human; 0 if AI-generated
Is_File_Level_Action	1 if generated by file-level action; 0 otherwise
Is_Action_[4]	One-hot (4): Which action produced the comment
Trigger_[auto/manual]	One-hot (2): Action trigger type: Auto/Manual (all-zero if human)
LLM_[GPT-3.5/4]	One-hot (2): LLM used for reviewing (all-zero if human)
Repository Features (3): Repository-level context where the review occurred	
Repo_File_Size	Total size of files in the repository (in bytes)
Repo_Issue_Count	Total number of issues in the repository
Repo_Contributor_Count	Number of non-bot contributors in the repository
Modification Features (12): Characteristics of the code changes under reviewed	
Author_Is_Bot	1 if the author of reviewed commit is a bot; 0 otherwise
Author_Is_Anon	1 if the author is anonymous; 0 otherwise
Author_Prior_Commits	Number of previous commits by the author to this repository
Commit_Del	Total lines deleted in changed files
Commit_Base_Lines	Total lines in all changed files at base commit
File_Is_Code	1 if a certain reviewed file is a programming file (via Linguist)
File_Depth	Directory nesting level of the file
File_Add/Del	Lines added/deleted in the reviewed file
File_Base_Lines	Line count of the file at base commit
Comment_Add/Del	Lines added/deleted the comment scope
Comment Features (11): Structure and content of the review comment itself	
Prior_Comment_Len	Total character length of all prior comments in the PR
Inline_Code	1 if contains inline code snippets (`code`)
Multiline_Code	1 if contains multiline code blocks (```code```)
Text_Length	Total Length of the current comment (in characters)
Code_Text_Ratio	Ratio of code length to total comment length
	Six LDA topic probabilities with GPT-4.1 labeled themes:
	0: Dependency Version Update and Compatibility;
	1: UI Usability and Accessibility Enhancements;
	2: Concurrency Control and Locking Robustness;
	3: Module Import Optimization;
	4: Variable Naming and Code Readability;
	5: Error Handling and Code Documentation;
LDA_Topic_[6]	

Phase I: Structured Feature Engineering. Inspired by previous works [9], [23], We initially built a 45-feature set across four dimensions to comprehensively capture the possible impacting factors. After checking for multicollinearity, we removed 9 redundant features, yielding a final set of 36 features presented in Table X. The complete list of candidate features can be found in our online appendix. Specifically, **Source Features** capture who generated the comment (a human developer or one of four actions) and two key aspects of action configuration: the trigger mode (automatic vs. manual) and the LLM used (GPT-3.5 vs. GPT-4). These configurations were extracted by parsing the workflow file at the comment’s associated commit. If the workflow included an ‘if’ conditional trigger, we labeled the comment as manually triggered; otherwise, it was considered automatic. To determine the LLM used for comment generation, we first checked for explicitly specified models in the workflow configuration. If unspecified ($n=1,043$), we manually inferred the default model by examining 9 associated action versions. Specifically, 7 were fixed versions (e.g., @v1) with constant defaults, while 2 were rolling main branch; for the latter, we identified the default switch commit (e.g., GPT-3.5→GPT-4) and labeled comments pre-/post-switch accordingly. All four actions in our final analysis employed OpenAI models, which we mapped to either the GPT-3.5 or GPT-4 family. **Repository Features** reflect the project context, including overall size and activity level (e.g., number of issues). **Modification Features** characterize the code changes under review, including the authorship and change properties. We used *Author_Prior_Commits* to estimate the author’s familiarity with the project. Edge

TABLE XI

RQ3: TOP 10 FEATURES INFLUENCING TOTAL COMMENT ADDRESSING.
GRAY-SHADED ROWS DENOTE FEATURES THAT ALSO RANK AMONG THE TOP 10 FOR AI COMMENT ADDRESSING (TABLE XII).

Feature	Rank	Importance ($ \phi $)	Directionality (ρ)
Source Feature (10): $\sum \phi =0.1589; \mu \phi =0.0159$			
Is_Human	1	0.0449	0.99
Trigger_auto	2	0.0410	-0.96
Is_File_Level_Action	3	0.0346	-0.95
Repository Feature (3): $\sum \phi =0.0342; \mu \phi =0.0114$			
Repo_File_Size	5	0.0184	0.70
Modification Feature (12): $\sum \phi =0.0801; \mu \phi =0.0067$			
Author_Prior_Commits	4	0.0235	-0.69
Commit_Del	9	0.0105	-0.15
Comment Feature (11): $\sum \phi =0.0959; \mu \phi =0.0010$			
Code_Text_Ratio	6	0.0155	0.89
Text_Length	7	0.0133	-0.24
LDA_Topic_2	8	0.0116	0.50
LDA_Topic_1	10	0.0102	0.41

cases, such as when the author was identified as a bot or an anonymous user (1.8% of valid comments), were flagged using *Author_Is_Bot/Anon*, with *Author_Prior_Commits* set to 0 accordingly. We further described change scope at the commit, file, and comment levels, along with the volume of the involved files before changes were made. **Comment Features** reflect the characteristics of the comment itself, including structural features (e.g., its position in the PR discussion thread) and textual features (e.g., code ratio). To capture comment intent, we applied Latent Dirichlet Allocation (LDA) [24] to the 3,879 comments. Following hyperparameter tuning strategies in prior work [25], we selected six as the optimal number of topics. We then used *gpt-4.1* to summarize the top ten comments for each topic to generate interpretable labels (as shown in Table X). Finally, we applied AutoSpearman correlation analysis [26] ($\rho > 0.7$ threshold) to address potential multicollinearity among features, and removed 9 redundant features (e.g., *Repo_File_Count*, *Commit_Changed_File_Count*).

Phase II: Model Fitting and Interpretation. Since our feature set spans four dimensions, we chose a Random Forest classifier with SHAP explanations to better model and interpret the complex, non-linear interactions between features and comment addressing. We further discuss this choice in Section VI. To enable focused analysis, we mapped the 3,879 comments into a binary classification task: comments labeled as “Valid-Partially” or “Valid-Fully” were grouped as the “Addressed” class (25.1%), with “Valid-Unaddressed” comments forming the “Not Addressed” class (74.9%). Upon our refined 36 features, we trained a Random Forest classifier using stratified 5-fold cross-validation. This approach ensures that our performance metrics and interpretability results are robust and not biased by a single random data split. The model achieved an average of 88.6% overall accuracy (Macro-F1=0.846) across the five test folds, indicating that our engineered features are effective predictors of comment addressing. Since Random Forest’s native feature importance metrics lack directional insights, we applied SHAP to quantify each feature’s contribution towards “Addressed” predictions. Specifically, we computed SHAP values for every instance in the test set of each fold and aggregated them to form a comprehensive explanation for the

TABLE XII

RQ3: TOP 10 FEATURES INFLUENCING AI COMMENT ADDRESSING.
GRAY-SHADED ROWS DENOTE FEATURES THAT ALSO RANK AMONG THE TOP 10 FOR TOTAL COMMENT ADDRESSING (TABLE XI).

Feature	Rank	Importance ($ \phi $)	Directionality (ρ)
Source Features (7): $\sum \phi =0.0418; \mu \phi =0.0060$			
Is_Action_3	2	0.0177	0.92
Is_File_Level_Action	7	0.0096	-0.88
Repository Features (3): $\sum \phi =0.0203; \mu \phi =0.0068$			
Repo_File_Size	6	0.0101	0.42
Modification Features (12): $\sum \phi =0.0767; \mu \phi =0.0064$			
Author_Prior_Commits	1	0.0212	-0.67
Commit_Base_Lines	4	0.0111	0.11
Commit_Del	5	0.0107	-0.17
Comment_Add	8	0.0095	0.81
Comment Features (11): $\sum \phi =0.0709; \mu \phi =0.0064$			
Code_Text_Ratio	3	0.0124	0.78
LDA_Topic_5	9	0.0085	0.61
Prior_Comment_Len	10	0.0075	0.65

entire dataset. SHAP is a widely-used technique to interpret learning-based models. It explains individual predictions by assigning each feature a Shapley value, which represents its average marginal contribution to the prediction across all possible feature combinations. Based on the aggregated SHAP values, we calculated two interpretability metrics for each feature: 1) **SHAP Importance**: the mean absolute SHAP value, reflecting global predictive importance; and 2) **SHAP Directionality**: the Pearson correlation between feature values and their SHAP values, indicating the association direction (positive or negative) with “Addressed” predictions.

Phase III: AI-Specific Modeling and Interpretation. In addition to modeling the entire dataset, we also conducted an AI-specific modeling and interpretation to ensure that the specific factors influencing the effectiveness of AI-generated comments were not overshadowed by the human-authored ones. To achieve this, we re-trained the Random Forest classifier using only the 2,990 AI-generated comments, removed the *Is_Human* feature, and mapped two sets of one-hot features (*Trigger_[auto/manual]* and *LLM_[GPT-3.5/4]*) into two binary features (*Is_Trigger_Manual* and *Is_Model_GPT4*). The resulting AI-Specific Model achieved an overall accuracy of 91.7% and a Macro-F1 score of 0.647 in 5-fold cross-validation. Given the highly imbalanced nature of the dataset (only 9.2% addressed comments), we consider this performance reasonable for supporting the subsequent interpretation analysis. Finally, we applied the same SHAP-based analysis to the remaining 33 features to interpret the model’s predictions.

Results: Table XI presents the interpretability results based on the complete dataset (including both human and AI comments). We list the top 10 most important features, organized by their feature groups. For each group, we also report the total SHAP Importance ($\sum |\phi|$) and average SHAP Importance ($\mu|\phi|$). In our subsequent analysis, we mainly focus on features that show a meaningful correlation (i.e. $|\rho| > 0.3$). Notably, “Source Features” and “Comment Features” rank first and second in total importance ($\sum |\phi|$), suggesting that both *who* provides the feedback (source credibility) and *what* the feedback contains (content quality) are strongly associated with whether comments are addressed.

Source Features represent the most significant group, showing a clear difference in developers’ responses based on the comment origins (human vs. actions; file-level vs. hunk-level). The negative correlation for *Is_File_Level_Action* ($\rho = -0.95$) aligns with the trend in Table IX, where hunk-level review actions generally outperform file-level counterparts. Specifically, coderabbitai/ai-pr-reviewer achieved the highest valid and addressed rate (19.2%). Automatic triggers also show a negative correlation with addressing (*Trigger_auto*, $\rho = -0.96$). To detail this effect, we analyzed data distribution by trigger modes across the four actions (Table XIII). We found that in the two actions with manually triggered comments (Action ID-1 and ID-2), manually triggered comments consistently showed higher addressing rates. This suggests that the massive feedback resulting from unconditional triggering might reduce developers’ willingness to respond. Interestingly, the LLM choice did not show high predictive importance (LLM_GPT-3.5 ranked 19/36 and LLM_GPT-4 ranked 21/36 in the feature list of the overall model; *Is_Model_GPT4* ranked 23/33 in the feature list of the AI-specific model), indicating that simply employing a more advanced model does not decisively guarantee comment addressing. However, when comparing addressing rates between two model families (Table XIV), GPT-4 generated comments did show better addressing rates.

For **Comment Features**, we find the comments were more likely to be addressed if they were concise (*Text_Length*, $\rho = -0.24$) or contained a higher proportion of code (*Code_Text_Ratio*, $\rho = 0.89$), especially when they included multi-line code blocks (*Has_Multiline_Code*, $\rho = 0.67$ not shown in the table). As shown in Table XV, the addressing rate for both human and AI-generated comments increased noticeably when *Code_Text_Ratio* exceeded 0.5. This may be because multi-line code is often used to illustrate concrete solutions, allowing developers to follow comments with minimal effort through copy-pasting. Regarding topics, comments were addressed more often when they pertained to “Concurrency Control and Locking Robustness” (e.g., “using a ‘set-if-not-exists’ pattern”) (*LDA_Topic_2*, $\rho = 0.50$), as well as “UI Usability and Accessibility Enhancements” (e.g., “adding aria-label for screen reader support”) (*LDA_Topic_1*, $\rho = 0.41$).

Within **Modification Features**, review comments targeting commits by experienced contributors were less likely to be addressed (*Author_Prior_Commits*, $\rho = -0.69$), suggesting that AI-generated reviews may be more helpful for newcomers. As shown in Table XVI, valid comments directed at project newcomers (*Author_Prior_Commits* ≤ 124) achieved 16% addressing rate, compared to just 3.3% for the most experienced contributors. Notably, human reviews mainly focused on these newcomers, with 79% of comments directed at them, aligning with typical community practices. In contrast, automated actions reviewed code changes regardless of author experience, which may help mitigate potential experience blindspots in the review process. Finally, positive correlations for **Repository Features** demonstrated that comments were more likely to be addressed in large and active projects (*Repo_File_Size*, $\rho = 0.70$; *Repo_Issue_Count*, $\rho = 0.64$, ranked 13/36).

Table XII presents the interpretability results for AI-

TABLE XIII
RQ3: ADDRESSING RATES BY TRIGGER MODE ACROSS REVIEW ACTIONS (p -VALUES FROM FISHER’S EXACT TEST).

Action	Auto Trigger		Manual Trigger		p -value
	Total	Addr. (%)	Total	Addr. (%)	
Total	2,886	9.0%	104	14.4%	> 0.05
ID-1	1,595	6.8%	86	12.8%	≤ 0.05
ID-2	602	0.5%	18	22.2%	≤ 0.05
ID-3	534	25.7%	—	—	—
ID-4	155	7.1%	—	—	—

TABLE XIV
RQ3: ADDRESSING RATES BY LLM SERIES ACROSS REVIEW ACTIONS (p -VALUES FROM FISHER’S EXACT TEST).

Action	GPT-3.5 Series		GPT-4 Series		p -value
	Total	Addr. (%)	Total	Addr. (%)	
Total	1,435	5.4%	1,555	12.6%	≤ 0.05
ID-1	1,431	5.5%	250	16.4%	≤ 0.05
ID-2	—	—	620	1.1%	—
ID-3	4	0.0%	530	25.8%	> 0.05
ID-4	—	—	155	7.1%	—

TABLE XV
RQ3: COMMENT ADDRESSING RATES BY BINNED CODE_TEXT_RATIO (ALL FOUR ACTIONS VS. HUMAN).

Bins of Code_Text_Ratio	Four Actions		Human	
	Total	Addr. (%)	Total	Addr. (%)
(-0.00, 0.00]	642	4.2%	320	71.2%
(0.00, 0.11]	669	7.2%	61	65.6%
(0.11, 0.26]	635	5.7%	95	75.8%
(0.26, 0.52]	621	10.5%	107	78.5%
(0.52, 1.00]	423	23.2%	306	89.9%

TABLE XVI
RQ3: COMMENT ADDRESSING RATES BY BINNED AUTHOR_PRIOR_COMMITS (ALL FOUR ACTIONS VS. HUMAN).

Bins of Author_Prior_Commits	Four Actions		Human	
	Total	Addr. (%)	Total	Addr. (%)
(-0.00, 30]	218	16.1%	558	80.1%
(30, 124]	639	16.1%	140	82.1%
(124, 319]	642	12.1%	131	71.8%
(319, 1,013]	763	4.5%	32	81.2%
(1013, 4,316]	728	3.3%	28	60.7%

generated comments. As indicated by the gray-shaded rows, features that rank highly in both the total and AI-specific models demonstrate the same influence direction (ρ). These consistent influence directions indicate that our main findings remain stable when restricting the analysis to AI-generated comments only. Although trigger-mode features become less prominent here, our earlier comparison across trigger modes (Table XIII) still supports higher addressing rates for manually triggered comments. Interestingly, when focusing solely on AI-generated comments, suggestions related to “Error Handling and Code Documentation” (e.g., “Instead of printing stack, use proper logging for exceptions”) (*LDA_Topic_5*, $\rho = 0.58$) become more influential and are positively associated with addressing. Finally, the rise of *Author_Prior_Commits* to the top rank with a negative correlation ($\rho = -0.67$) highlights that there is still significant room for AI reviewers to evolve to become valuable for experienced developers.

Answer to RQ3: Comment addressing is strongly associated with **Source Features** (human vs. AI and specific action type) and **Comment Features** (conciseness and code-richness are better), highlighting the importance of thoughtful automated review design.

V. DISCUSSION

Does AI-based code review lead to code changes? In general, human-authored review comments are much more likely to be addressed than AI-generated ones: 60% of valid human review comments led to code changes, compared to only 0.9%–19.2% for valid AI-generated comments depending on the tool (Table IX). However, the answer is not simply “no”. Our results show that there is a clear potential for AI-generated code review to influence development if these tools are designed well.

A. Design Matters

Our study reveals that *not all AI-based review tools are equal*. Among the four actions we analyzed in depth, one stood out: `coderrabbitai/ai-pr-reviewer` achieved a 19.2% rate of valid comments addressed, outperforming other tools by a large margin. In contrast, some tools had near zero impact. For example, `mattzcarey/code-review-gpt` had only 0.9% of valid comments addressed. This suggests that *tool design choices directly affect the developer’s response*.

Through our interpretability analysis (Table XI), we identified several characteristics that consistently increased the likelihood of AI-generated comments being addressed.

- **Hunk-level granularity:** Comments attached to specific code locations perform better than general or file-level comments. Tools such as `coderrabbitai/ai-pr-reviewer` that offer hunk-level feedback are more effective.
- **Manual triggering:** Tools explicitly triggered by developers produce higher addressing rates. For example, `anc95/ChatGPT-CodeReview` showed a 12.8% addressing rate for manually triggered comments versus 6.8% for automatically triggered ones (Table XIII).
- **Code-rich, concise comments:** Comments with a high code-to-text ratio, especially multiline suggestions, are more likely to be acted upon. We observed a clear increase in the addressing rate when more than half of the comment content was code (Table XV).
- **Focus on newcomers:** Comments were more likely to be addressed when targeting contributors with little prior commit history. For the least experienced developers, the addressing rate was five times higher than for the most experienced (Table XVI).

To further investigate the effectiveness of review granularity, we randomly investigated 14.4%, i.e., 30 comments, of the 208 PR-level review comments as follows: 5 from Action ID-8, 5 from ID-11, and 20 from ID-16. Only 1 comment was partially addressed, which provided a specific performance improvement: “*For better performance, preallocate the ‘architectures’ slice by using ‘make([]types.Architecture, 0, len(fm.Spec.Architectures))’*”. This aligns with our finding that concrete code suggestions are more likely to be adopted. However, the majority of PR-level comments were ignored, supporting our conclusion that fine-grained (i.e., hunk-level) reviews are more effective for AI-generated code review.

We also highlight that prompt design plays an important role in generating high-quality comments. For example,

```

tests/io/test_api.py
2 2 def test_load_encoder_with_downsampling():
3 3     data = aeon.load (...)
4 4     assert len(data) < len(raw_data) # Current check

```

github-actions
The length check is good, but we should also verify that the data range is preserved (start and end timestamps) to ensure no data is lost at boundaries.
AI's Initial Code (Strict Check)
+ assert data.index[0] == raw_data.index[0]
+ assert data.index[-1] == raw_data.index[-1]

Senior Developer
Rejection: Downsampling naturally drops data points. It is not required to preserve the exact first and last points. The current test is sufficient.

github-actions
Clarification: Understood. However, reducing points shouldn't mean altering the time range. We need to ensure the dataset isn't inadvertently "trimmed" from the edges.

Senior Developer
Technical Rejection: Preserving the "range" makes sense, but your suggested strict check (==) will fail due to rounding errors (binning effects) when downsampling.

github-actions
Refinement: You are correct about the rounding. Here is a robust fuzzy check that validates the range stays within one sampling period.
AI's Refined Code (Fuzzy / Robust Check)
+ assert abs(data.index[0] - raw_data.index[0]).total_seconds() <= 0.02
+ assert abs(data.index[-1] - raw_data.index[-1]).total_seconds() <= 0.02

[Assert expected data ranges and add test comments](#) a45b16a

Fig. 4. Illustrative case ([link](#)): Effective tool design (**hunk-level review**, **concrete code suggestions**, and **interactive dialogue**) successfully persuading a senior developer to adopt a change.

`anc95/ChatGPT-CodeReview` uses a short, generic prompt such as “*Below is a code patch, please help me do a brief code review on it. Any bug risks and/or improvement suggestions are welcome.*”, which often results in vague comments. In contrast, `coderrabbitai/ai-pr-reviewer` provides a long and detailed prompt (over 400 words), explicitly instructing the model to avoid generalities and provide focused feedback. It also includes logic to suppress comments such as “*LGTM!*” unless no issues are found, further reducing noise.

These differences are reflected in the types of output generated. We observed many vague or unhelpful comments from tools with limited prompt context. For example, a frequent output was: “*Without more context, it is difficult to provide further suggestions*” – a clear signal that the model lacked sufficient information to provide value. Other typical outputs included overly generic summaries of code functionality or hallucinated style warnings unrelated to the actual changes.

To help reduce noise, some tools also implemented logic to avoid redundant reviews. For example, `coderrabbitai/ai-pr-reviewer` logs reviewed commits to avoid a full base-to-head review every time. This helps prevent developer frustration caused by redundant feedback, a concern also noted by Cihan et al. [27] in their study on AI-assisted code review tools in industrial settings.

We present an illustrative case in Figure 4 to show why these design choices matter. This case drawn from `coderrabbitai/ai-pr-reviewer`, demonstrates how concrete modifications and interactive feedback can persuade even an experienced developer (`Author_Prior_Commits=186`) to accept a suggestion they initially rejected. After the developer

identified potential issues with rounding errors, the AI adapted its proposal from a strict equality check to a robust fuzzy comparison, successfully transforming the initial rejection into an actionable improvement.

Beyond post-comment refinement, we also argue that AI tools could be more cautious before posting. We found that most current tools rely on a simplistic one-in-one-out paradigm, where each code change automatically leads to a review comment, regardless of whether a comment is warranted. This leads to low-precision output and reviewer fatigue. Instead, we advocate for a multi-stage approach, similar to practices in traditional machine learning pipelines. The tool should first determine whether a change needs review, then decide whether to generate a comment, and finally, where appropriate, offer a code suggestion. This would bring structure and discipline to AI-based review and help ensure that comments are meaningful and justified.

B. Beyond Code Changes: Impact on Closed PRs and Feedback on Unaddressed Comments

While we mainly focused on merged PRs to observe definitive code improvements, we recognize that AI-generated review comments can provide value even when they do not lead to a merge or any direct code changes. Therefore, we conducted two supplementary analyses on the comments outside our main dataset.

1) *Analysis of AI Comment Impact in Closed PRs:* We extended our analysis to closed (non-merged) pull requests for the four primary actions to investigate if developers engage with AI feedback before a PR is abandoned. Following the methodology proposed in RQ2, we applied the LLM-assisted framework to assess whether comments were valid and addressed in the latest commits of these closed PRs. We observed a significantly lower addressing rate for AI comments in closed PRs compared to merged ones: only 3.0% (17/570) for Action ID-1, and 0% for Action ID-2 ($n = 14$), ID-3 ($n = 18$), and ID-4 ($n = 0$). Our qualitative analysis suggests that even when a comment is valid, the PR may still be abandoned because the overall approach “doesn’t solve the root cause”. Therefore, we focused on merged PRs in our main results to ensure we capture actual code improvements that successfully transitioned into the project’s codebase.

2) *Analysis of Feedback on Unaddressed AI Comments:* To capture impact beyond immediate code changes, we analyzed developer replies to AI comments that were not labeled as “Valid, Addressed” in Table IX. We identified a total of 50 replies and manually categorized them into four types:

- **Defending the Change (64%):** Developers defended their current implementation as correct or preferred and explained why the AI suggestion was not adopted, e.g., “I don’t want to bring back ‘GetParameter’ as using type parameters is a lot nicer” (link).
- **Prompting Reflection (14%):** AI comments triggered technical debates, which may lead to code changes depending on the outcome. For instance, a developer questioned the suggested optimization: “Is ‘iterrows’ really more efficient than vectorized operations? The golden rule is benchmarks” (link).

- **Assessing Tool Capabilities (12%):** Beyond replying to a specific suggestion, developers evaluated the overall performance of the AI or action. For instance, one developer noted it was “interesting to learn where AI stands” (link) despite the specific suggestion being of limited value.
- **Willingness to Follow (10%):** Developers expressed an intention to apply the suggestion in future iterations, e.g., “Templates could be enhanced later on to use markdown instead of raw text” (link).

Specifically, **Willingness to Follow (10%)** and **Prompting Reflection (14%)** imply that even if not addressed immediately, a comment may still trigger code changes in the future. These findings motivate future work on impact beyond code changes (e.g., guiding developer reasoning or shaping long-term code quality) to better understand the contribution of AI code reviewers to software engineering practice.

C. Recommended Features for Effective AI Code Review

Based on our findings, we recommend that designers of AI-based code review tools prioritize the following.

- **Use hunk-level review granularity:** Inline comments tied to specific code changes are more actionable than general or file-level comments.
- **Enable and encourage manual triggering:** Give developers control over when the tool runs to avoid flooding pull requests with unhelpful comments.
- **Include specific code suggestions:** Multi-line code blocks are especially effective as they can often be copy-pasted with minimal effort.
- **Filter out vague comments:** Use a pre-check (like our Stage-1 classification model) to suppress non-actionable suggestions before they reach developers.
- **Support adaptive reviewing:**
 - Target less experienced contributors who may benefit more from feedback.
 - Adjust the level of detail according to the change in size or file criticality.
- **Provide richer context to the LLM:** Supplying the PR title, description, summaries of changes, or even user’s feedback helps generate more useful review comments.

By adopting these strategies, AI-based review tools can become more than noise – they can become useful collaborators in the development process, both in open-source communities and in industrial practice.

D. Implications for AI Code Review Adopters

We also derive the following implications for developers considering or currently adopting AI-based code review tools.

Choose and tune the tool appropriately: Given the large performance differences among AI-based review tools, we recommend prioritizing well-designed and highly customizable actions such as `coderabbitai/ai-pr-reviewer` as a starting point. Second, do not rely solely on the default settings. In our study, 82.6% of the repositories analyzed customized their action configurations, and 22.5% continued to

adjust the configurations beyond one month. This suggests that integrating AI code review tools requires careful tuning and ongoing optimization. Based on our observations in RQ1, this includes practices such as keeping the action and its underlying LLMs up to date, and addressing project-specific needs by customizing prompts, refining file inclusion/exclusion patterns, and adjusting trigger modes.

Clarify the role of AI code review: Even the best-performing AI tool achieved only a 19.2% addressing rate, far below the human reviewers’ 60%. Therefore, for now, AI code review is better framed as a *complement* to human review, not a replacement. This requires understanding AI code review’s capability boundaries and deploying it in appropriate scenarios. Based on our LDA topic analysis (RQ3), AI reviewers are good at providing defensive programming suggestions, and developers are relatively more responsive to their comments on error handling. Accordingly, we recommend positioning AI reviewers as *defenders*: tightly scope them to this mandate (with targeted prompts and other recommended features), thereby suppressing category-sprawling yet low-quality comments. This allows human reviewers to focus on higher-level issues, such as overall architecture (e.g. “Try sorting at the DB level, it will be 100x faster”), development context (e.g., “Remove this as project requires Python ≥ 3.11 ”), and business logic (e.g., “This will cause issues on ARM64”). Mapping the boundaries of AI code review’s effectiveness is also an ongoing task for researchers.

Monitor effectiveness over time: We recommend using the LLM-assisted framework proposed in this study to monitor comment validity and addressing rates over time. These metrics provide a crucial feedback loop to continually tune and customize the AI reviewer.

E. Implications for GitHub

For GitHub, we recommend the following to better support the development and analysis of automated code review tools.

Enhance the traceability of Actions’ behaviors: Currently, GitHub retains action execution logs for up to 400 days, and all actions operate under a shared `github-action[bot]` account. This makes it difficult to attribute specific comments or behaviors to individual actions—a threat also noted in Section VI. Therefore, we suggest refining action identification to the particular action within a repository, which enables precise source attribution for users and researchers. Additionally, we suggest that GitHub establish a “usage list” (maybe labeled by a successful run) for Marketplace actions, similar to repository forks. This would also enable action developers, potential users, and researchers to assess their true popularity.

Improve support for identifying actual code changes: Currently, most tools re-review the full base-to-head diff every time, since there’s no easy way to trace the reviewed commits to calculate incremental changes. Therefore, we suggest GitHub provide a platform-level solution for this, rather than leaving it to individual action developers. This may be implemented through extending the existing “viewerViewedState” flag (a boolean indicating file view status) to the specific

commit hash of the last reviewed version. Additionally, a true “compare file content” endpoint (that ignores noise from rebases) could also ease implementing code review tools.

VI. THREATS TO VALIDITY

Construct Validity: We identify AI generated comments through the `github-action[bot]` account, which may include comments from unrelated actions. Although our manual checks during annotation revealed no such cases, misclassification remains possible. Additionally, we determined comment addressing based on file-level code changes, which may be insufficient for certain edge cases. However, “uncertain” cases constituted only 1.3% of our addressing dataset, and we excluded these from our impact factor analysis.

Internal Validity: In RQ2, we extracted 150 comments from over 22,000 AI-generated and 1,166 human-authored comments and obtained gold-standard labels via multi-rater annotation. To assess representativeness, the first author examined an additional 250 samples and observed similar addressing patterns as reported in our main findings. Therefore, we believe the gold standard reflects the broader dataset. For RQ3, our addressing analysis focuses on four popular actions, which may limit the generalizability of our findings. However, within the scope of our dataset, usage was highly concentrated on these four actions, making them representative of real-world deployments observed in our study. Moreover, the final analysis uses a binary classification of whether comments were addressed, which does not capture the degree or impact of the resulting code changes. Nevertheless, our evaluation shows substantial inter-rater agreement and strong LLM performance on this task, ensuring the reliability of our findings. Our feature engineering for impact factor analysis may also be incomplete or miss latent feature interactions. However, our features was comprehensively designed across four dimensions and inspired by prior works [9], [23]. Another threat is that different interpretation methods may yield different analysis results. Since our feature set spans four dimensions, we chose Random Forest to better model the complex and non-linear interactions between features, rather than the simpler model like logistic regression, which assumes a linear relationship between features. We also tried logistic regression in our experiments and it achieved a lower performance (86.1% accuracy and 0.826 Macro-F1 in 5-fold cross-validation) than our Random Forest model (88.6% and 0.846 respectively). This model choice also aligns with work [9], [23]. And the use of SHAP is a standard approach for interpreting such models as well. Therefore, our interpretation pipeline is well-justified. We also acknowledge that these interpretations describe associations between features and the predicted probability of “Addressed”, not causal effects. If causal identification is desired, further studies (e.g., controlled experiments or developer interviews) would be needed.

External Validity: Our study includes 16 AI-based code review actions identified through a systematic Marketplace search and manual verification. Rather than sampling a subset, we included all actions that met our relevance criteria and collected the complete set of observable review comments

generated by these tools in mature repositories. Although four tools account for the majority of comments (most notably `anc95/ChatGPT-CodeReview`), this concentration reflects actual ecosystem usage patterns at the time of data collection. We acknowledge that less popular or newly released tools are underrepresented in absolute volume. However, their limited usage inherently results in fewer observable review behaviors. Thus, our dataset captures the near-complete population of AI-based review activity occurring on GitHub Actions at the time of the study, particularly among tools with meaningful adoption. To further verify dataset coverage, we conducted a follow-up Marketplace search on July 23, 2025. We observed no substantial changes among the top-ranked actions. The only notable newcomer, `hustcer/deepseek-review` (ranked #7, 324 stars), had its first commit on January 29, 2025, i.e., after our action selection date (January 7, 2025). This confirms that our dataset accurately reflects the state of the ecosystem during the study period. Consequently, our findings, such as the benefits of hunk-level granularity and concrete code suggestions, are derived from analysis across multiple widely used tools and human reviews, rather than from a single-tool artifact. We will continue to monitor the evolution of the AI-based code review ecosystem in future work. Second, to mitigate the impact of inactive repositories which may not yet have established stable review workflows, we followed prior research [19] and collected data only from repositories with at least 50 PRs. However, this threshold does not fully capture the diversity of project scales. Since our dataset primarily consists of small- to medium-sized projects (typically with ≤ 50 non-bot contributors), our findings may not generalize to very large-scale industrial systems. Moreover, our main analysis focuses on review comments generated by file-level and hunk-level review actions and examines whether they are addressed in merged PRs. We did not consider PR-level actions. This design choice simplifies the analysis by ensuring that we can reliably observe whether comments have been addressed, but it may also narrow the generalizability of our findings. To broaden our understanding beyond this core scope, we additionally examined a sample of PR-level review comments and conducted the supplementary analyses on closed PRs and feedback on unaddressed AI comments, presented in Section V-B. Additionally, our analysis focused on English-language comments. As described in RQ2 Phase I, we began with 16,762 comments from 100 repositories. A language filter reduced this to 4,229 comments from 53 repositories, excluding about 75% of comments and 47 repositories. Among them, 32 repositories were excluded because they only contained Korean comments (e.g., 1,828 comments from `KEA-DoKebi/dalkom-backend`). Despite this reduction, we retained over half of the original projects. And given that English serves as the main communication language in global development activities [28], with even non-native speakers commonly adopting it, we believe our findings remain relevant to the broader open-source community.

VII. RELATED WORK

We organize related work into two areas relevant to our study: (a) the automation of code review using AI, and (b)

developer response to automated feedback.

a) AI for Code Review Automation: The automation of code review has attracted growing attention with the rise of GenAI. Modern large language models, such as GPT-4 and LLaMA, have shown strong performance on code-related tasks, including vulnerability detection [29], program repair [30]–[32], and bug fixing [33]–[35], enabled by the naturalness of source code [36]–[38].

Review automation builds on the observation that many review comments are repetitive and can be learned. Prior work has proposed systems for three core tasks [39]: (1) determining whether a code change needs review [40], [41], (2) generating natural language comments [40], [42]–[44], and (3) generating revised code based on comments [42], [45], [46].

Comment generation is particularly complex, requiring the synthesis of context-sensitive suggestions. Early approaches used encoder-decoder models such as T5 [42], and subsequent work explored specialized pretraining [40], [43], cross-task learning [47], and benchmark datasets [44], [48]. Other work has demonstrated that general-purpose models like LLaMA can be adapted to this task through fine-tuning [3].

Recent work explored how LLMs can be more effectively applied to code review, e.g. by incorporating pull request context [49] or through mixture-of-prompts architectures [50]. Meanwhile, benchmark-oriented studies also developed more realistic evaluation settings for AI-based code review [51], [52]. Additionally, Cihan et al. studied the impact of AI-assisted code review in industrial settings [27].

Although some studies include automatic and human evaluations of generated comments [42], most of this research has focused on comment quality rather than downstream impact. Our study complements these efforts by analyzing how AI-generated comments are used in open-source development workflows, whether they lead to code changes, and which factors influence their effectiveness.

b) Developer Response to Automated Feedback: Several studies have explored how developers respond to code review comments, with a focus on human-authored feedback. Prior work has investigated factors that make comments useful [7]–[9], developer responsiveness [11], and differences in comment granularity and presentation [6], [10].

Building on this foundation, more recent work has turned to developers’ reactions to AI-generated artifacts. For example, Vaithilingam et al. [53] examined developers’ expectations and experiences with GitHub Copilot, and Huang et al. [54] studied perceptions of AI-generated inline versus block comments.

However, little is known about whether developers actually act on AI-generated code review comments. Our study addresses this gap by analyzing over 22,000 such comments across 178 repositories, assessing whether they were addressed, and identifying the factors that influence their effectiveness.

VIII. CONCLUSION AND FUTURE WORK

This paper presents a large-scale empirical analysis of AI-based code review actions on GitHub. We examine how these tools are adopted, whether their comments lead to

code changes, and what factors influence their effectiveness. Our findings reveal that adoption is highly concentrated on four popular actions. Many AI-generated comments are not addressed, especially when they are vague or lack context. However, comments that are concise, specific are more likely to be addressed—particularly by less experienced contributors. Moreover, even when AI comments do not lead to immediate code changes, we find that they can still prompt developer reflection and inform future improvements. Based on our findings, we offer implications for code review tool designers,

adopters, and GitHub.

We recommend treating AI code reviewers as collaborators with human reviewers. Future work should further enhance review quality by exploring richer context, better prompts, and multi-stage strategies to generate more specific and actionable feedback. It is also essential to clarify AI’s capabilities to identify optimal ways for human-AI collaboration. We hope our findings contribute to the development of more practical and developer-friendly AI-based code review tools.

ACKNOWLEDGMENTS

This work is supported by the Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (No. JYB2025XDXM118), the National Natural Science Foundation of China (No. 62572237, No. 62302210, No. 72371125), the 2024 Development and Testing Tools Project (CEIEC-2024-ZM02-0066), the Frontier Technologies R&D Program of Jiangsu (No. BF2024059), and the Natural Science Foundation of Jiangsu Province (No. BK20241195).

REFERENCES

- [1] C. Sadowski, E. Söderberg, L. Church, M. Sipko, and A. Bacchelli, “Modern code review: a case study at google,” in *Proceedings of the 40th international conference on software engineering: Software engineering in practice*, 2018, pp. 181–190.
- [2] G. Gousios, M.-A. Storey, and A. Bacchelli, “Work practices and challenges in pull-based development: The contributor’s perspective,” in *Proceedings of the 38th international conference on software engineering*, 2016, pp. 285–296.
- [3] 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.
- [4] M. Wessel, J. Vargovich, M. A. Gerosa, and C. Treude, “Github actions: the impact on the pull request process,” *Empirical Software Engineering*, vol. 28, no. 6, p. 131, 2023.
- [5] A. Decan, T. Mens, P. R. Mazrae, and M. Golzadeh, “On the use of github actions in software development repositories,” in *2022 IEEE International Conference on Software Maintenance and Evolution (IC-SME)*. IEEE, 2022, pp. 235–245.
- [6] 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.
- [7] A. Bosu, M. Greiler, and C. Bird, “Characteristics of useful code reviews: An empirical study at microsoft,” in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 146–156.
- [8] 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.
- [9] M. M. Rahman, C. K. Roy, and R. G. Kula, “Predicting usefulness of code review comments using textual features and developer experience,” in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 215–226.
- [10] H. Y. Lin, P. Thongtanunam, C. Treude, M. W. Godfrey, C. Liu, and W. Charoenwet, “Leveraging reviewer experience in code review comment generation,” *arXiv preprint arXiv:2409.10959*, 2024.
- [11] L. MacLeod, M. Greiler, M.-A. Storey, C. Bird, and J. Czerwonka, “Code reviewing in the trenches: Challenges and best practices,” *IEEE Software*, vol. 35, no. 4, pp. 34–42, 2017.
- [12] “Does AI code review lead to code changes? A case study of GitHub actions (Online Appendix),” <https://zenodo.org/records/19562450>, 2026, zenodo, accessed 14-04-2026.
- [13] S. Wagner, M. M. Barón, D. Falessi, and S. Baltes, “Towards evaluation guidelines for empirical studies involving llms,” in *2nd International Workshop on Methodological Issues with Empirical Studies in Software Engineering (WSESE 2025)*, 2025.
- [14] S. Baltes, F. Angermeir, C. Arora, M. M. Barón, C. Chen, L. Böhme, F. Calefato, N. Ernst, D. Falessi, B. Fitzgerald, D. Fucci, M. Kalinowski, S. Lambiase, D. Russo, M. Lungu, L. Prechelt, P. Ralph, C. Treude, and S. Wagner, “Evaluation guidelines for empirical studies in software engineering involving llms,” 2025. [Online]. Available: <https://arxiv.org/abs/2508.15503>
- [15] “Definition of ‘hunk’ in the gnu diffutils manual,” https://www.gnu.org/software/diffutils/manual/html_node/Hunks.html, 2025.
- [16] “Openai,” <https://chat.openai.com>, 2026.
- [17] “Gemini,” <https://gemini.google.com/>, 2026.
- [18] “Claude,” <https://claude.ai>, 2026.
- [19] P. Reich and W. Maalej, “Testability refactoring in pull requests: Patterns and trends,” in *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 2023, pp. 1508–1519.
- [20] M. Golzadeh, D. Legay, A. Decan, and T. Mens, “Bot or not? detecting bots in github pull request activity based on comment similarity,” in *Proceedings of the IEEE/ACM 42nd international conference on software engineering workshops*, 2020, pp. 31–35.
- [21] “Deepseek,” <https://deepseek.com>, 2026.
- [22] S. M. Lundberg and S. Lee, “A unified approach to interpreting model predictions,” in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, 2017, pp. 4765–4774.
- [23] T. Dey and A. Mockus, “Effect of technical and social factors on pull request quality for the NPM ecosystem,” in *ESEM ’20: ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, Bari, Italy, October 5-7, 2020*. ACM, 2020, pp. 11:1–11:11. [Online]. Available: <https://doi.org/10.1145/3382494.3410685>
- [24] D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent dirichlet allocation,” *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.
- [25] C. Treude and M. Wagner, “Predicting good configurations for github and stack overflow topic models,” in *Proceedings of the 16th International Conference on Mining Software Repositories, MSR 2019, 26-27 May 2019, Montreal, Canada*. IEEE / ACM, 2019, pp. 84–95.
- [26] J. Jiarpakdee, C. Tantithamthavorn, and C. Treude, “Autospearman: Automatically mitigating correlated software metrics for interpreting defect models,” in *2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Madrid, Spain, September 23-29, 2018*. IEEE Computer Society, 2018, pp. 92–103.
- [27] 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,” in *2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2025, pp. 425–436.
- [28] Y. Wang, “Language matters,” in *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2015, pp. 1–10.
- [29] M. Fu and C. Tantithamthavorn, “Linevul: A transformer-based line-level vulnerability prediction,” in *Proceedings of the 19th International Conference on Mining Software Repositories*, 2022, pp. 608–620.
- [30] M. Fu, C. Tantithamthavorn, T. Le, V. Nguyen, and D. Phung, “Vulrepair: a t5-based automated software vulnerability repair,” in *Proceedings of the 30th ACM joint european software engineering conference and symposium on the foundations of software engineering*, 2022, pp. 935–947.
- [31] X. Zhou, K. Kim, B. Xu, D. Han, and D. Lo, “Out of sight, out of mind: Better automatic vulnerability repair by broadening input ranges and sources,” in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.
- [32] M. Fu, V. Nguyen, C. Tantithamthavorn, D. Phung, and T. Le, “Vision transformer inspired automated vulnerability repair,” *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 3, pp. 1–29, 2024.
- [33] M. Tufano, C. Watson, G. Bavota, M. D. Penta, M. White, and D. Poshyanyk, “An empirical study on learning bug-fixing patches in the wild via neural machine translation,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 28, no. 4, pp. 1–29, 2019.
- [34] N. Jiang, T. Lutellier, and L. Tan, “Cure: Code-aware neural machine translation for automatic program repair,” in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1161–1173.
- [35] M. Jin, S. Shahriar, M. Tufano, X. Shi, S. Lu, N. Sundaresan, and A. Svyatkovskiy, “Inferfix: End-to-end program repair with llms,” in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 1646–1656.
- [36] B. Ray, V. Hellendoorn, S. Godhane, Z. Tu, A. Bacchelli, and P. Devanbu, “On the ‘naturalness’ of buggy code,” in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 428–439.
- [37] A. Hindle, E. T. Barr, M. Gabel, Z. Su, and P. Devanbu, “On the naturalness of software,” *Communications of the ACM*, vol. 59, no. 5, pp. 122–131, 2016.
- [38] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, “A survey of machine learning for big code and naturalness,” *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–37, 2018.
- [39] H. Y. Lin, C. Liu, H. Gao, P. Thongtanunam, and C. Treude, “The code review comprehension assessment for large language models,” in *Findings of the Association for Computational Linguistics: ACL 2025*, 2025.

- [40] 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.
- [41] 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.
- [42] 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.
- [43] 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.
- [44] H. Y. Lin, P. Thongtanunam, C. Treude, and W. Charoenwet, “Improving automated code reviews: Learning from experience,” in *Proceedings of the 21st International Conference on Mining Software Repositories*, 2024, pp. 278–283.
- [45] 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.
- [46] 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.
- [47] O. Ben Sghaier and H. Sahraoui, “Improving the learning of code review successive tasks with cross-task knowledge distillation,” *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 1086–1106, 2024.
- [48] B. Lin, S. Wang, Z. Liu, Y. Liu, X. Xia, and X. Mao, “Cct5: A code-change-oriented pre-trained model,” in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 1509–1521.
- [49] Y. Zhang, Y. Zhang, Z. Sun, Y. Jiang, and H. Liu, “Laura: Enhancing code review generation with context-enriched retrieval-augmented llm,” in *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2025, pp. 2983–2995.
- [50] Y. Peng, K. Kim, L. Meng, and K. Liu, “icodereviewer: Improving secure code review with mixture of prompts,” in *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2025, pp. 3204–3215.
- [51] Y. Zhang, Z. Pan, I. N. B. Yusuf, H. Ruan, R. Shariffdeen, and A. Roychoudhury, “Code review agent benchmark,” *arXiv preprint arXiv:2603.23448*, 2026. [Online]. Available: <https://arxiv.org/pdf/2603.23448>
- [52] K. Pereira, N. Sinha, R. Ghosh, and D. Dutta, “Cr-bench: Evaluating the real-world utility of ai code review agents,” *arXiv preprint arXiv:2603.11078*, 2026. [Online]. Available: <https://arxiv.org/pdf/2603.11078>
- [53] P. Vaithilingam, T. Zhang, and E. L. Glassman, “Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models,” in *Chi conference on human factors in computing systems extended abstracts*, 2022, pp. 1–7.
- [54] Y. Huang, H. Guo, X. Ding, J. Shu, X. Chen, X. Luo, Z. Zheng, and X. Zhou, “A comparative study on method comment and inline comment,” *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 5, pp. 126:1–126:26, 2023. [Online]. Available: <https://doi.org/10.1145/3582570>