

Issue-Oriented Agent-Based Framework for Automated Review Comment Generation

SHUOCHUAN LI, Tianjin University, China

DONG WANG*, Tianjin University, China

PATANAMON THONGTANUNAM, The University of Melbourne, Australia

ZAN WANG, Tianjin University, China

JIUQIAO YU, University of California Berkeley, USA

JUNJIE CHEN, Tianjin University, China

Code review (CR) is a crucial practice for ensuring software quality. Various automated review comment generation techniques have been proposed to streamline the labor-intensive process. However, existing approaches heavily rely on a single model to identify various issues within the code, limiting the model's ability to handle the diverse, issue-specific nature of code changes and leading to non-informative comments, especially in complex scenarios such as bug fixes. To address these limitations, we propose RevAgent, a novel agent-based issue-oriented framework, decomposes the task into three stages: (1) Generation Stage, where five category-specific commentator agents analyze code changes from distinct issue perspectives and generate candidate comments; (2) Discrimination Stage, where a critic agent selects the most appropriate issue-comment pair; and (3) Training Stage, where all agents are fine-tuned on curated, category-specific data to enhance task specialization. Evaluation results show that RevAgent significantly outperforms state-of-the-art PLM- and LLM-based baselines, with improvements of 12.90%, 10.87%, 6.32%, and 8.57% on BLEU, ROUGE-L, METEOR, and SBERT, respectively. It also achieves relatively higher accuracy in issue-category identification, particularly for challenging scenarios. Human evaluations further validate the practicality of RevAgent in generating accurate, readable, and context-aware review comments. Moreover, RevAgent delivers a favorable trade-off between performance and efficiency.

CCS Concepts: • **Software and its engineering** → **Software maintenance tools**.

Additional Key Words and Phrases: Automated Code Review, Large Language Models

ACM Reference Format:

Shuochuan Li, Dong Wang, Patanamon Thongtanunam, Zan Wang, Jiuqiao Yu, and Junjie Chen. 2025. Issue-Oriented Agent-Based Framework for Automated Review Comment Generation. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 24 pages. <https://doi.org/XXXXXXX.XXXXXXX>

*Corresponding Author.

Authors' Contact Information: Shuochuan Li, lishuochuan@tju.edu.cn, Tianjin University, Tianjin, China; Dong Wang, dong_w@tju.edu.cn, Tianjin University, Tianjin, China; Patanamon Thongtanunam, The University of Melbourne, Victoria, Australia; Zan Wang, Tianjin University, Tianjin, China; Jiuqiao Yu, University of California Berkeley, Berkeley, California, USA; Junjie Chen, Tianjin University, Tianjin, China.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Conference acronym 'XX, Woodstock, NY

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-XXXX-X/2018/06

<https://doi.org/XXXXXXX.XXXXXXX>

1 Introduction

Code review (CR) is a fundamental practice in software quality assurance, serving to uncover defects, suboptimal design choices, and potential bugs while safeguarding the long-term quality and maintainability of a codebase [8, 53, 54]. Despite its benefits in detecting issues early, promoting knowledge sharing, and enforcing coding standards, code review is labor-intensive, time-consuming, and highly dependent on reviewers' expertise [32]. To address these challenges, automated code review approaches have gained increasing attention as a means to enhance efficiency and consistency [50]. A central component of such automation is review comment generation, which involves detecting issues in code changes and providing actionable suggestions. The quality of these comments is essential, as they directly influence the effectiveness of subsequent code refinements. Therefore, generating accurate and relevant review comments is critical to ensuring that issues are thoroughly addressed and properly documented.

Recently, increasing efforts have been devoted to automating review comment generation, resulting in a variety of techniques [22, 29, 31, 34, 45, 51, 52, 61], including approaches based on information retrieval (IR), pre-trained language models (PLMs), and large language models (LLMs). For instance, Li et al. [29] proposed CodeReviewer, leveraging CodeT5 for pre-training on large-scale code review datasets. To address the limited generalizability of pre-trained models, recent studies have increasingly adopted LLMs with task-specific fine-tuning. For instance, Lu et al. [31] introduced LLaMA-Reviewer by fine-tuning a base LLaMa model, while Nashaat et al. [34] and Yu et al. [61] fine-tuned LLMs on a benchmark dataset for specific requirements of code review, all of them showing the promise. These efforts collectively demonstrate the promise of LLM-based approaches in enhancing the quality and relevance of generated review comments.

However, existing approaches typically rely on a single model to analyze code and detect diverse issue types. This lack of specialization blurs focus and often yields superficial, non-informative comments, undermining review effectiveness. As noted by Tufano et al. [50], such methods also overlook the heterogeneity of code-change issues. Specifically, well-established PLM- and IR-based approaches primarily skew toward easy refactorings (e.g., simple variable or constant changes) and perform poorly on more critical scenarios, like bug fixes, testing, and logging. A major driver of this disparity is dataset bias: complex issue types are underrepresented in existing benchmarks, pushing models toward frequent, simpler patterns and leaving them without the issue-specific knowledge or contextual sensitivity required to generate meaningful comments for high-impact cases. Recent industrial research [17] further emphasizes that LLM-based reviewers should balance comment types while improving clarity and relevance.

On the other hand, while LLM-based approaches have exhibited superior performance in terms of textual similarity metrics (e.g., BLEU), their capability to generate issue-specific and contextually relevant comments remains largely unexplored. Tufano et al. [50] made an initial attempt to address this limitation by introducing a prompt-based approach that guides an LLM using a taxonomy of five root code-related categories. However, their evaluation revealed that even with such prompts, a single LLM still struggled to grasp the nuanced logic of each issue category on the code-to-comment generation task, underscoring the need for more structured and context-sensitive strategies. Therefore, we argue that leveraging diverse comment generation experts, each focusing on a distinct issue category, could overcome the limitations of one-size-fits-all approaches and produce more accurate, issue-oriented review comments for practitioners.

To enable LLMs to better focus and capture a broader range of issues, we propose **RevAgent**, a novel multi-issue-oriented framework designed to enhance the diversity and depth of generated review comments. To effectively handle different issue types, RevAgent leverages an agent-based architecture that transforms the traditional single-LLM pipeline into a collaborative multi-agent

system, each specialized in a specific issue type. Inspired by the recent success of agent-based systems for natural-language-to-code tasks (e.g., code generation [25, 62]), this work adopts a multi-agent system for this code-to-natural-language task. Rather than designing a one-stage review comment generation pipeline, RevAgent has a three-stage process (i.e., the *Generation Stage*, *Discrimination Stage*, and *Training Stage*), to enhance performance through specialization and targeted augmentation. RevAgent addresses a central challenge in identifying the appropriate perspective for a given code change, especially in the absence of explicitly labeled issue categories. In particular, RevAgent introduces three coordinated stages: (1) *Generation Stage*, where five category-specific commentator agents act as specialized reviewers. Each agent analyzes the code change from a distinct issue-category perspective (e.g., refactoring, bug-fixing, and logging) and produces a corresponding candidate review comment. (2) *Discrimination Stage*, where a critic agent evaluates the candidate comments, infers the most relevant issue category, and selects the most appropriate final comment. To equip each agent involved in the aforementioned stages with the necessary expertise, RevAgent incorporates a foundational stage: (3) *Training Stage*, where each commentator agent is fine-tuned on data specifically curated for its assigned issue category. The critic agent, responsible for cross-category evaluation, is trained on a diversified dataset in which similar code changes are paired with review comments from multiple issue perspectives.

To evaluate RevAgent, we conducted extensive experiments on the Curev [46], which contains 20,000 review comment instances categorized into five issue types: Refactoring, Bugfix, Testing, Logging, and Documentation. The results demonstrate that RevAgent outperforms both PLM-based and LLM-based state-of-the-art baselines, achieving average improvements of 12.90%, 10.87%, 6.32%, and 8.57% in BLEU, ROUGE-L, METEOR, and SBERT scores, respectively. In terms of issue-category identification, RevAgent achieves an overall accuracy of 60.20%, including 21.69% accuracy in the challenging Bugfix category, where existing approaches typically perform below 5%. To assess the contributions of individual components, we performed ablation experiments by replacing the category-specific commentator agents and the critic agent with alternative implementations. Results show that using multiple specialized commentator agents significantly improves the generation of category-aligned comments compared to a single unified agent, and the critic agent plays the most crucial role in accurately identifying issue categories. Furthermore, human evaluations on a total of 1,920 samples, with 384 samples evaluated for each of the five approaches, confirm that the review comments generated by RevAgent are indeed helpful for developers, offering improved readability, accuracy, and alignment with the intended issue category. From an efficiency standpoint, although the agent-based design introduces a modest additional latency of 0.038 seconds per generation compared to a single LLM, the performance gains justify this trade-off and remain within a practically acceptable range. Finally, we manually analyze the root causes of low-quality generated comments to guide future research, revealing that 48% of failure cases are attributed to the lack of project-specific knowledge (e.g., coding standards).

Contributions. To summarize, this paper makes the following key contributions:

- We propose a novel agent-based framework for automated review comment generation that enhances issue-oriented feedback by equipping category-specific experts and coordinating their reasoning to handle diverse issue types.
- We conduct a comprehensive evaluation of our framework on a large-scale dataset, demonstrating that RevAgent significantly outperforms state-of-the-art approaches in both quantitative metrics and qualitative assessments.
- We extensively evaluate various advanced open-source LLMs within our framework, examining models of different sizes to assess their effectiveness and scalability within our framework.

- To support future research, we publicly release our replication package [1], which includes all necessary code and resources to reproduce the results.

Paper Organization. The remainder of this paper is organized as follows. Section 2 positions our work in relation to prior research. Section 3 details our proposed agent-based framework. Section 4 describes the experimental setup. Section 5 presents results and answers the research questions. Section 6 further examines efficiency and provides qualitative insights. Section 7 outlines threats to validity. Finally, we conclude the paper and highlight directions for future work in Section 8.

2 Related Work

This section surveys related work on automated code review and agent-based AI to situate our research within existing literature.

Automated Code Review. Modern Code Review (MCR) is a collaborative and lightweight practice widely adopted in contemporary and open-source software development, serving as a cornerstone of quality assurance and a vital mechanism for knowledge sharing among developers [8, 43]. The process begins when a developer submits a proposed code change, typically in the form of a pull request or merge request, through a version control platform (e.g., GitHub). Reviewers are then assigned to examine the change, leave inline or general comments, and often engage in iterative discussions with the author to clarify intentions or suggest improvements. The author responds by revising the code, addressing feedback, or justifying design decisions. This cycle continues until the reviewers are satisfied with the change’s correctness, clarity, and overall quality. Once approved and after passing all automated checks, the change is merged into the main codebase.

To alleviate the substantial manual effort involved in code review, automation has garnered increasing attention in recent years. Early efforts, such as CodeReviewer[29] and AUGER[27], applied deep learning techniques based on PLMs to support review activities. With the emergence of LLMs, more recent studies have explored their use in code review, improving both generation quality and evaluation methods. For instance, LLaMA-Reviewer [31] fine-tuned the LLaMA model specifically for review tasks. Building on this direction, subsequent work has investigated dataset augmentation, fine-tuning strategies, prompt engineering, and the design of collaborative AI systems [19, 34, 49, 61]. Specifically, Nashaat et al. [34] and Yu et al. [61] constructed benchmark datasets to support LLM fine-tuning in review settings. CodeAgent [49] introduced a multi-agent framework that simulates collaborative review dynamics through role-specific agents. BitsAI-CR [47] adopted a two-stage architecture combining a RuleChecker and a ReviewFilter to enhance the precision of generated comments. More recently, Wu et al. [60] introduced CodeDoctor that generates multi-category review comments at one shot using one LLM. However, this approach departs from the established paradigm of automated comment generation and may impose additional cognitive burden on practitioners.

Despite these advances, it is noted that most existing methods rely on single-model architectures, limiting their ability to handle diverse issue types and resulting in less precise, context-aware comments. To address this limitation, we follow the established paradigm (for a given code change, generate an issue-specific review comment) and propose an agent-based framework that generates review comments from multiple perspectives and evaluates trade-offs to select the most relevant and informative as the final output.

Agent-Based AI. It refers to artificial intelligence systems designed to achieve shared goals with humans or other AI systems. Previous research has extensively explored the use of multiple LLMs in agent-based or multi-agent settings [39, 48]. These approaches leverage inter-agent interactions to collectively enhance LLM capabilities, leading to improved overall performance across various scenarios [26, 37, 57]. For example, Akata et al. [5] examined LLM cooperation through orchestrated

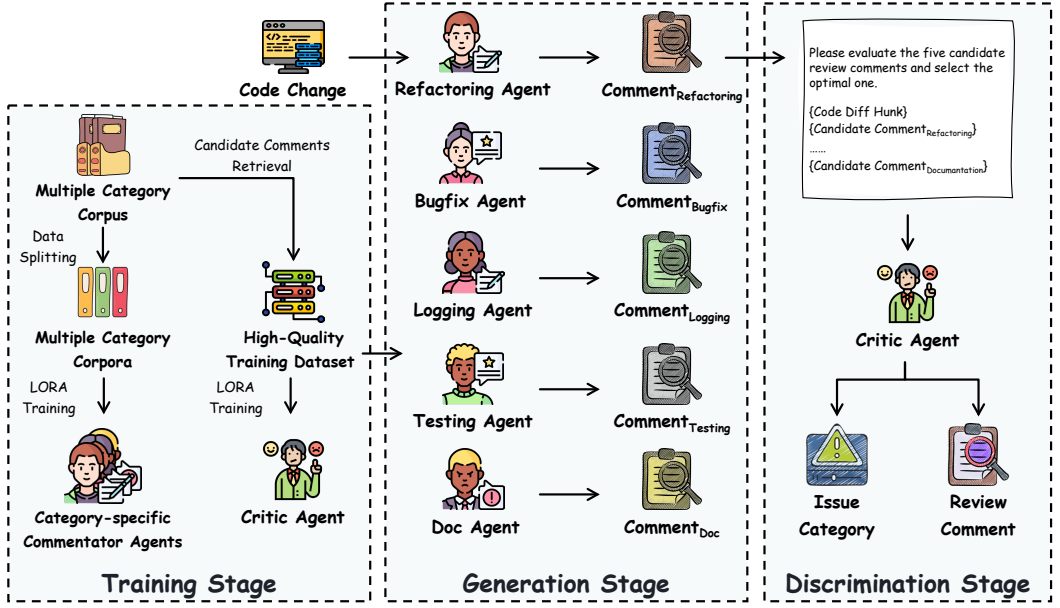


Fig. 1. The overview of RevAgent

repeated games, while Cai et al. [11] proposed an agent-based framework where large models act as tool-makers and small models as tool-users to reduce computational costs.

In the software engineering domain, agent-based AI is increasingly promising for a variety of scenarios [21]. Li et al.[26] and Qian et al.[39] introduced multi-agent frameworks for software development, relying on natural language conversations rather than formalized software engineering artifacts. Elicitation [7] simulates the requirement elicitation phase by assigning distinct personas to different agents for interactive role-play, while MapCoder [25] emulates the code generation cycle through four specialized LLM agents responsible for recalling relevant examples, planning, generating code, and debugging. Wang et al. [55] introduces the multi-model collaborative vulnerability detection approach that leverages the strong capability of analyzing vulnerability semantics from LLMs. Their results show clear gains over single-model detectors.

Despite recent progress, applying agent-based AI to code review poses distinct challenges. Unlike generic generation tasks, reviewers must interpret fine-grained diffs, infer underlying issue type (e.g., refactoring, bug fix, testing), and produce context-aware feedback. A single, monolithic model tends to blur these distinctions and produce generic, low-value comments. Building on these insights, we extend agent-based AI to automated code review with an agent-based framework tailored for issue-oriented comment generation, coupling category-specialist commentator agents with a critic that coordinates and selects the most relevant review comment.

3 RevAgent

This section introduces RevAgent, a novel approach to generating review comments based on an issue-oriented agent-based system. As shown in Figure 1, the framework comprises three stages: *Generation*, *Discrimination*, and *Training*. In the *Generation Stage*, given a code change, RevAgent analyzes it from five distinct issue-category perspectives and generates corresponding candidate review comments. In the *Discrimination Stage*, a critic agent evaluates these candidates based on

Table 1. The taxonomy of issue categories

Category	Definition
Refactoring	Suggestions to improve code structure.
Bugfix	Identifies and suggests fixes for bugs.
Testing	Comments related to test cases.
Logging	Suggestion for logging practices.
Documentation	Recommendations for documentation changes.
Others	Unspecified or ambiguous comments.

factors such as correctness and severity, and selects the most appropriate issue-comment pair as the final output. To enhance the performance of both the category-specific commentator agents and the critic agent, the *Training Stage* involves fine-tuning each agent on task-specific datasets.

3.1 Generation Stage

The goal of this stage is to identify the primary issues in a given code change from diverse issue-category perspectives and generate corresponding candidate review comments. To accomplish this, we deploy five category-specific commentator agents, each functioning as an independent code reviewer specialized in a particular issue type. Each agent analyzes the code change to detect issues relevant to its assigned category and generates a potential suggestion for improvement. These agents operate in a competitive setting, with each striving to produce the most accurate and contextually appropriate review comment from its respective perspective.

The prompt for each category-specific commentator agent contains two main components: a task description and a query code diff hunk. In the task description section, we first define the agent's role and specify the potential issue it should address, as highlighted in red in the example prompt. Following Tufano et al. [50], issues are categorized into six major groups: *Refactoring*, *Bugfix*, *Testing*, *Logging*, *Documentation*, and *Others*, as summarized in Table 1. In line with prior research [15, 33], we treat the *Others* category, defined as containing vague or non-specific comments, as noisy data and exclude it. For each remaining category, we insert a corresponding directive into the prompt to guide the agent's analysis:

- **Refactoring:** "The diff hunk needs to be revised to refactor the code to improve its quality."
- **Bugfix:** "The diff hunk needs to be revised to fix one or more bugs."
- **Testing:** "The diff hunk needs to be revised since tests for this code must be written."
- **Logging:** "The diff hunk needs to be revised to improve the logging of its execution."
- **Documentation:** "The diff hunk needs to be revised to be more compliant with the documentation specification."

In the following prompt, the agent is instructed to assess whether the diff hunk requires revision under the "Bugfix" issue category:

[Role] You are an expert code reviewer.

[Task] Give you a diff hunk, your task is to decide whether **the diff hunk needs to be revised to fix one or more bugs**. If the response to the above point is True, then write a code review.

[Note] Note that code changes may be marked: Code lines that begin with a plus (+) sign indicate new code, while those that begin with a minus (-) sign indicate deleted code.

Finally, the query code diff hunk is provided as input, prompting the agent to analyze the change from the specified issue-category perspective and generate a corresponding review comment in the expected output format.

For the new diff hunk:

< Code Diff Hunk >

Please output the review comment in the following format:

Review Comment:...

3.2 Discrimination Stage

In this stage, the identified issues and their associated review comments are thoroughly scrutinized, evaluated, and selected by the critic agent based on factors such as correctness and severity. The objective of this stage is to simulate the role of an oracle, accurately discerning the correct issue-comment pair and prioritizing it over all other false positives. The relationship between the category-specific commentator agents and the critic agent is inherently cooperative: the critic's ability to make precise judgments depends on the quality and diversity of the candidate comments generated by the commentator agents.

In the prompt for the critic agent, we first provide an overview of the framework's workflow and clearly define its task: to evaluate the candidate review comments based on correctness and severity, and to select the most accurate issue-comment pair as the final output, as shown in the following illustration.

[Role] You are an expert code reviewer.

[Task] There is a code review process where a diff hunk is analyzed from five perspectives: 'refactoring', 'bugfix', 'testing', 'logging', and 'documentation', and then the review comments are generated. After analysis from these different perspectives by category-specific commentator agents, you are required to evaluate the five review comments generated and select the comment that best aligns with the issues present in the target diff hunk.

[Duty] As a meticulous and harsh critic, your duty is to scrutinize these review comments and evaluate the identified issues with scores in terms of correctness.

Next, we provide the critic agent with the query code diff hunk along with the candidate comments generated by the five category-specific commentator agents. This setup allows the critic to directly compare the insights derived from multiple issue-specific perspectives and assess their relative correctness and severity. We then prompt the critic agent to analyze the input and select the most appropriate issue category and corresponding review comment, returning the result in the expected output format. Due to space constraints, the prompt is available in our public repository.

3.3 Training Stage

Through the aforementioned two stages, RevAgent enables the generation of review comments by comprehensively analyzing code changes from multiple issue-specific perspectives. However, due to the inherent complexity of the code review process, it is challenging for LLM-based agents to fully internalize and execute the framework using prompt engineering alone. Recent studies [24, 28, 61] have shown that general-purpose LLMs, while equipped with broad linguistic and factual knowledge through large-scale pre-training, require fine-tuning to perform effectively on specialized downstream tasks. This fine-tuning is essential for aligning model behavior with task-specific requirements and enabling precise reasoning in structured, role-based workflows. Accordingly, we fine-tune each agent in RevAgent for its designated role, equipping the commentator agents with the ability to generate category-specific review comments and the critic agent with the capability to accurately identify issue categories.

Tuning Category-specific Commentator Agents. We begin by partitioning the training dataset, referred to as the Multiple Category Corpus, into five separate Category-specific Corpora

based on the issue category of each data instance. Each corpus is then used to fine-tune a corresponding commentator agent specialized for that category. To support this process, we adopt the LlamaFactory framework [64], a widely used tool in the open-source community for fine-tuning LLMs. To minimize computational and storage overhead, we apply low-parameter fine-tuning (i.e., LoRA [23]), a parameter-efficient fine-tuning method. LoRA assumes that the parameter updates required during fine-tuning lie in a low-dimensional subspace, allowing the update ΔW to be decomposed as:

$$W' = W_0 + \Delta W = W_0 + BA \quad (1)$$

where, W' represents the fine-tuned parameters, W_0 denotes the pre-trained parameters, and ΔW signifies the change in the model parameters during fine-tuning. Matrices $B \in \mathbb{R}^{d \times r}$ and $A \in \mathbb{R}^{r \times k}$ are the low-rank decomposition components, with d and k being the dimensions of the model parameters and $r \ll \min(d, k)$. During training, only B and A are updated while W_0 remains fixed, significantly reducing the number of trainable parameters. This approach enables efficient fine-tuning of LLMs with minimal resource consumption while preserving performance.

Tuning Critic Agent. Unlike the category-specific commentator agents, training the critic agent requires constructing candidate comments from the four non-ground-truth categories for each training instance. This step is essential to align with the structure expected by the instruction fine-tuning template. Formally, an instruction tuning template is defined as:

$$P = \{\mathcal{NL} + x + y + R_x\} \quad (2)$$

where $\mathcal{NL}$ is the prompt template described in Section 3.2, x and y represent the input code diff hunk and the desired review comment, respectively, and R_x is a set of candidate comments for the other four categories (excluding c_y). R_x is defined as:

$$R_x = \{r_{x,c} | c \in C \setminus \{c_y\}\} \quad (3)$$

where C denotes the set of categories, c_y is the category of y and $r_{x,c}$ represents a candidate comment for x in the issue category c . We conjecture that the stronger the relevance between the candidate comment $r_{x,c}$ and the code changes x , the more effectively the fine-tuned critic agent can learn the fine-grained differences across various issue categories. To support this, we introduce a *Candidate Comment Retrieval* approach, which constructs highly relevant candidate comments for each code change. This ensures that the critic agent is trained on contextually meaningful and diverse examples.

Candidate Comment Retrieval. Although many studies [14, 56] have explored data augmentation using LLMs, existing research [12, 13] suggests that its effectiveness diminishes as dataset size increases. Furthermore, LLM-based augmentation is computationally expensive and time-consuming, making it less practical for large-scale datasets. To address these limitations, we adopt a simpler and more efficient retrieval-based approach for constructing the training dataset of the critic agent. Prior work [22] has shown that similar code changes often receive similar review comments, even when their surrounding contexts differ. Inspired by this, we propose a retrieval-based approach for constructing candidate comments. To ensure stronger relevance between the retrieved $r_{x,c}$ and the query diff hunk x —for example, ensuring $r_{x,c}$ includes variable names from x —we follow prior studies [4, 35] and employ the BM25 IR algorithm [44]. Specifically, given a code diff hunk x from the Multiple Category Corpus, RevAgent inputs x into four category-specific commentator agents to obtain the candidate comments R_x . Formally, this is defined as:

$$R_x = \{CCR(x, c) | c \in C \setminus \{c_y\}\} \quad (4)$$

where $CCR(x, c)$ denotes the retrieval process, using BM25 to retrieve the most relevant example from the corpora of category c based on the code similarity, and its associated comment is used as

Table 2. Overall statistic of Curev datasets

Category	Train			Test		
	Count	Avg. Code Line	Avg. Comment Token	Count	Avg. Code Line	Avg. Comment Token
Refactoring	10,396	20.33	22.20	3,465	19.72	22.19
Bugfix	2,061	18.82	25.24	687	19.74	25.42
Testing	361	31.97	23.42	120	29.85	23.30
Logging	146	16.16	22.01	48	11.54	22.33
Documentation	702	14.15	24.34	234	16.28	23.43
Total	13,666	20.05	22.80	4554	19.73	22.77

* “Avg.” denotes average values of all instances.

the candidate comment $r_{x,c}$. As the five category-specific corpora are mutually disjoint, there is no risk of candidate comment duplication across categories. To build a training instance for the critic agent, we combine the instruction prompt $\mathcal{NL}$, the code diff hunk x , its ground-truth comment y , and the set of generated candidate comments R_x . Finally, we adopt LoRA to fine-tune the critic agent, enhancing its capability on issue-category identification.

4 Experimental Setup

We formulate the following three research questions (RQs):

- **RQ1:** How does RevAgent perform compared to the state-of-the-art baselines?
- **RQ2:** How much does each component contribute to the overall performance of RevAgent?
 - **RQ2.1:** What is the effect of *Training Stage*?
 - **RQ2.2:** What are the effects of the *Generation Stage* and *Discrimination Stage*?
- **RQ3:** How does RevAgent perform in human evaluation?

Dataset. In this work, we use the multi-category review comment datasets, Curev, introduced by Sghaier et al. [46] for evaluation. Curev is derived from the dataset originally proposed by CodeReviewer [29], which remains the largest publicly available dataset for code reviews. The raw dataset contains 176,613 multilingual samples across nine programming languages, including PHP, Ruby, C#, C, Java, Python, C++, Go, and JavaScript. It has been widely used in prior works [29, 31, 34, 45, 51, 52, 61]. Sghaier et al. [46] used Llama-3.1-70B to evaluate and classify the raw dataset based on the definition of issue category in [50]. Then they filtered out the low-quality samples and selected a subset of 20,000 samples to construct the curated datasets. In this work, following the original configuration, we split the Curev dataset into 75% for training and 25% for evaluation, as shown in Table 2, ensuring that each sample is a unique code change–comment pair from a single pull request, with no train–test overlap. Beyond its large scale and multi-language coverage, Curev also exhibits marked class imbalance, with refactoring accounting for 69.3% of the entire dataset and the remaining categories jointly comprising 30.7%. This mirrors real-world issue distributions, where some categories are far more common than others. As a result, models trained on such data tend to overfit dominant classes (e.g., refactoring) and underperform on rarer but important ones (e.g., testing). In this work, we address this methodological challenge directly rather than relying on simple data rebalancing.

Studied LLMs. For practicality (e.g., the high cost of fine-tuning proprietary models like GPT-4o), we excluded proprietary models and evaluated three leading families of open-source LLMs for broader coverage: Qwen2.5-Coder [40], DeepSeek-Coder [18], and Llama3 [2]. Qwen2.5-Coder

is a code-specialized model developed by Alibaba as part of the Qwen 2.5 series. It features enhanced performance in programming tasks, achieving state-of-the-art results among open models. DeepSeek-Coder is a family of code-focused LLMs developed by DeepSeek, comprising both base and instruction-tuned variants. Trained on a diverse dataset including source code and related documentation, it supports long-context inputs up to 16K tokens. Llama3 is the latest iteration in the Llama series, developed by Meta. It offers enhanced processing power, versatility, and accessibility, with improvements in multilingual support, coding, and reasoning capabilities.

Baselines. Our baseline selection follows established practices in code review research [61], including the most representative and state-of-the-art approaches from relevant categories. The baselines used for comparison in this study include CodeReviewer [29], LLaMA-Reviewer [31], TufanoLLM [50], and CodeAgent [49]:

- **CodeReviewer** is a pre-trained model to automate code review activities and possesses four specialized tasks to improve its understanding of the review domain. To further investigate CodeReviewer’s performance on issue-category identification, we extend it into a variant named **CodeReviewer-c**, which is retrained to jointly predict both the review category and the corresponding review comment.
- **LLaMA-Reviewer** is an LLM-based approach for automatic code review and uses Parameter-Efficient Fine-Tuning (PEFT) to fine-tune LLaMA for the code review task. In line with CodeReviewer-c, we construct a variant **LLaMA-Reviewer-c**, which prompts the LLM to generate both the review category and the review comment simultaneously.
- **TufanoLLM** employs a prompt engineering technique that first prompts the LLMs to identify the issue category, followed by generating corresponding review comments.
- **CodeAgent** utilizes a multi-agent framework to detect functionally related defects in pull requests, incorporating inter-agent communication and debate to produce more comprehensive review reports. Although originally operating at the commit level, we adapt it to our function-level setting for a fair comparison.

We exclude BitsAI-CR [47] and CodeMentor [34] due to their lack of open-source availability, and Carllm [61] is omitted as it primarily concerns on fine-tuning LLMs on benchmarks with specific requirements, without proposing generic methodologies on review comment generation. Additionally, we eliminate CodeDoctor [60] that generates multi-category review comments at one shot for a given code change, a setting that diverges from the mainstream paradigm of automated code review. A related line of work is automatic commit-message generation, which summarizes the what and why of code changes, essentially a summarization task. In contrast, code review seeks to detect potential issues (e.g., hidden defects), making it a detection task. Due to this task mismatch, we do not treat commit-message generation methods [59, 63] as direct baselines. To ensure a fair comparison and isolate the impact of methodological differences, all LLM-based approaches were implemented using the same underlying LLM.

Evaluation Metrics. We use the following commonly used evaluation metrics to assess the comments generated by RevAgent:

- **BLEU** (BiLingual Evaluation Understudy) [36] is a precision-based metric computing the weighted geometric mean of modified 1- to 4-gram precisions, with length penalty. In this study, we use BLEU-4, which is widely adopted in code comment generation tasks.
- **ROUGE-L** [30] measures the longest common subsequence (LCS) between the generated and reference comments, focusing on recall and content coverage.
- **METEOR** [9] combines precision and recall using their harmonic mean, offering a more balanced similarity evaluation than BLEU.

Table 3. Performances of RevAgent and baselines on average

Model	Method	BLEU	ROUGE-L	METEOR	SBERT	Pred. Acc.
CodeT5-base	CodeReviewer	7.36	16.45	10.11	30.85	-
	CodeReviewer-c	6.46	14.88	7.86	19.08	76.35%
Llama-3-8B	LLaMA-Reviewer	7.64	17.70	12.32	44.42	-
	LLaMA-Reviewer-c	7.63	17.77	11.01	40.88	51.19%
	TufanoLLM	2.83	8.56	8.37	34.52	34.89%
	CodeAgent	3.80	9.68	6.50	18.49	-
	RevAgent	8.51	19.51	12.73	47.96	56.52%
Deepseek-Coder-6.7B	LLaMA-Reviewer	7.60	17.84	12.07	43.78	-
	LLaMA-Reviewer-c	6.22	13.93	9.68	33.71	1.84%
	TufanoLLM	4.40	11.09	9.05	37.05	72.84%
	CodeAgent	4.39	6.95	3.78	4.68	-
	RevAgent	8.61	19.73	12.97	48.35	67.13%
Qwen2.5-Coder-7B	LLaMA-Reviewer	7.75	17.97	12.29	44.88	-
	LLaMA-Reviewer-c	6.98	16.10	10.40	37.72	44.73%
	TufanoLLM	2.93	11.10	11.12	42.16	41.77%
	CodeAgent	5.05	8.93	3.92	7.16	-
	RevAgent	8.69	19.92	13.14	48.58	63.02%
Qwen2.5-Coder-14B	LLaMA-Reviewer	7.91	18.30	12.55	45.50	-
	LLaMA-Reviewer-c	8.33	19.29	11.90	44.00	52.48%
	TufanoLLM	2.70	10.24	10.67	39.91	40.21%
	CodeAgent	4.16	6.16	3.93	7.38	-
	RevAgent	9.08	20.46	13.50	48.97	54.13%

- **SBERT** [20] assesses semantic similarity by computing cosine similarity between Sentence-BERT [42] embeddings of generated and reference comments.
- **Pred. Acc.** measures whether RevAgent correctly predicts the issue category of a code diff. Formally, let $C_{correct}$ be the number of correct predictions, and C_{total} the total predictions. We define prediction accuracy as:

$$Pred. Acc. = \frac{C_{correct}}{C_{total}} \times 100\% \quad (5)$$

Implementation and Environment. We implemented our approach using the LlamaFactory [64] framework. All experiments were conducted on a server with four NVIDIA A800 GPUs and 512 GB of memory, running Ubuntu 20.04.2. Following the prior studies [31, 61], during training, the model is loaded with float16 precision and trained for 5 epochs with a batch size of 64. In Low-rank Adaptation (LoRA), we set the learning rate to 0.0003, weight decay to 0.01, LoRA rank to 8, the LoRA scaling factor to 16, LORA dropout to 0.05, and warmup ratio to 0.1. Following prior work [4], we set the temperatures to 0 for well-defined answers from the LLM. More details on the specific hyperparameters are available in our materials.

5 Result

5.1 RQ1: Comparison with Baselines

Analysis. We evaluate RevAgent and four state-of-the-art baselines, TufanoLLM, CodeAgent, CodeReviewer, and LLaMA-Reviewer, as well as two variants (described in Section 4) on the code review comment generation task using four widely adopted metrics: BLEU, ROUGE-L, METEOR, and SBERT. To ensure model diversity, we select one representative from each major LLM family: LLaMA-3-8B, Deepseek-Coder-6.7B, and Qwen2.5-Coder-7B. To further explore the impact of model

Table 4. Wilcoxon Signed-Rank Test P-Values for Metric Comparisons Between RevAgent and LLaMA-Reviewer

LLM	BLEU	ROUGE-L	METEOR	SBERT
Llama-3-8B	6.86×10^{-35}	1.44×10^{-29}	7.53×10^{-5}	1.40×10^{-27}
Deepseek-Coder-6.7B	2.05×10^{-33}	5.07×10^{-28}	5.71×10^{-15}	4.00×10^{-43}
Qwen2.5-Coder-7B	1.46×10^{-27}	8.04×10^{-30}	4.78×10^{-12}	4.09×10^{-28}
Qwen2.5-Coder-14B	8.07×10^{-39}	7.84×10^{-37}	4.98×10^{-16}	2.50×10^{-26}

scale on RevAgent’s performance, we also include Qwen2.5-Coder-14B in our experiments. Due to resource constraints for fine-tuning, we limit our evaluation to models with up to 14B parameters. To validate the statistical significance of our results, we conducted the Wilcoxon signed-rank test [58] at a 0.05 confidence level between RevAgent and the best-performing baseline. Additionally, to assess the effectiveness of review comment generation across different issue categories, we partition the test dataset into five groups based on issue categories. For each category, we report the average prediction accuracy (Pred. Acc.).

Results. Table 3 reports the average performance of RevAgent versus the baselines, and Table 4 presents the statistical test results. First, in terms of textual metrics, we observe that RevAgent outperforms across all evaluation metrics. Specifically, RevAgent improves BLEU, ROUGE-L, METEOR, and SBERT by 12.90%, 10.87%, 6.32%, and 8.57% on average across four different LLMs, respectively, compared to the best-performing baseline, i.e., LLaMA-Reviewer. As shown in Table 4, statistical testing via the two-sided Wilcoxon signed-rank test confirms significance for all evaluation metrics, with all p -values $< 7.53\text{e-}05$. Furthermore, we find that RevAgent slightly outperforms LLaMA-Reviewer in METEOR across these four LLMs. This difference may be attributed to the nature of the METEOR metric, which is sensitive to synonyms and stem variations but also penalizes non-contiguous word order. Consequently, RevAgent may generate more flexible or restructured phrasing, which makes its scores slightly lower on METEOR. This is further supported by the SBERT results, where RevAgent consistently achieves the highest scores among all baselines, indicating its superior ability to generate semantically accurate review comments. In contrast, TufanoLLM shows weak performance. This may be due to its lack of access to sufficient in-context examples, which makes it difficult to accurately identify the issue categories in the code diffs, resulting in divergent and imprecise review comments. Similarly, CodeAgent achieves poor performance due to its focus on security analysis, making it hard to detect other categories of issues in the code diffs. Moreover, we observe that CodeReviewer-c and LLaMA-Reviewer-c, which incorporate issue-category predictions, generally exhibit a slight performance decline compared to their original counterparts. This degradation likely results from the increased complexity of combining classification with generation, which transforms the task from a single-objective generation problem into a more challenging classification-plus-generation setting.

In terms of the model diversity and scale, when comparing models of similar sizes from different families (6.7B, 7B, and 8B), we observe that the general-purpose model LLaMA-3, which is not specialized for the code task, performs the worst, while Qwen2.5-Coder and DeepSeek-Coder show complementary strengths, each excelling in different aspects. These findings suggest that when model sizes are comparable, performance under RevAgent is potentially influenced more by model specialization. Conversely, when comparing models of different sizes within the same family (i.e., Qwen2.5-Coder 7B vs. 14B), the larger 14B variant outperforms the 7B model by 0.39, 0.54, 0.36, and 0.39 points on BLEU, ROUGE-L, METEOR, and SBERT, respectively. This indicates that as model size increases, RevAgent likely further amplifies its performance.

Table 5. Performances of RevAgent and baselines on each category on Deepseek-Coder-6.7B

Category	Method	BLEU	ROUGE-L	METEOR	SBERT	Pred. Acc.
Refactoring	CodeReviewer	7.65	17.02	10.85	35.42	-
	CodeReviewer-c	6.70	15.47	8.26	20.00	97.72%
	LLaMA-Reviewer	7.58	17.84	12.03	43.44	-
	LLaMA-Reviewer-c	6.31	14.00	9.78	33.60	2.11%
	TufanoLLM	4.36	10.87	8.97	36.70	95.53%
	CodeAgent	4.35	6.95	3.88	3.02	-
	RevAgent	8.67	19.98	13.08	48.23	82.48%
Bugfix	CodeReviewer	6.03	13.29	7.15	15.35	-
	CodeReviewer-c	5.33	11.74	6.22	15.59	4.95%
	LLaMA-Reviewer	7.49	17.13	11.91	44.83	-
	LLaMA-Reviewer-c	5.59	12.87	8.91	32.99	0.58%
	TufanoLLM	4.54	12.00	9.34	38.91	0.87%
	CodeAgent	4.31	6.87	3.78	5.20	-
	RevAgent	8.14	17.66	12.28	49.29	21.69%
Testing	CodeReviewer	7.43	16.17	8.56	19.49	-
	CodeReviewer-c	5.75	11.70	6.14	18.52	0.00%
	LLaMA-Reviewer	7.60	17.13	11.91	41.17	-
	LLaMA-Reviewer-c	5.77	12.98	8.38	28.30	0.83%
	TufanoLLM	4.60	10.98	8.39	30.87	0.00%
	CodeAgent	4.45	6.38	3.49	6.53	-
	RevAgent	8.15	19.44	11.99	43.73	14.17%
Logging	CodeReviewer	6.66	15.96	9.42	31.91	-
	CodeReviewer-c	5.86	13.84	7.54	27.93	0.00%
	LLaMA-Reviewer	7.55	19.56	12.54	49.94	-
	LLaMA-Reviewer-c	6.09	15.06	10.26	42.74	0.02%
	TufanoLLM	4.39	11.67	9.61	41.91	2.08%
	CodeAgent	4.44	6.41	3.30	6.68	-
	RevAgent	8.26	19.95	12.52	49.37	8.33%
Documentation	CodeReviewer	7.12	17.59	8.81	14.37	-
	CodeReviewer-c	6.62	17.16	7.71	13.97	24.36%
	LLaMA-Reviewer	8.20	19.96	13.43	45.83	-
	LLaMA-Reviewer-c	7.11	16.25	11.05	38.48	0.02%
	TufanoLLM	4.63	11.67	9.56	38.94	0.00%
	CodeAgent	4.77	7.52	3.53	9.80	-
	RevAgent	9.36	22.22	13.99	49.61	12.39%
Average	CodeReviewer	7.36	16.45	10.11	30.85	-
	CodeReviewer-c	6.46	14.88	7.86	19.08	76.35%
	LLaMA-Reviewer	7.60	17.84	12.07	43.78	-
	LLaMA-Reviewer-c	6.22	13.93	9.68	33.71	1.84%
	TufanoLLM	4.40	11.09	9.05	37.05	72.84%
	CodeAgent	4.39	6.95	3.78	4.68	-
	RevAgent	8.61	19.73	12.97	48.35	67.13%

We further analyzed the average performance across the five issue categories, as shown in Table 5. Given the importance of accurate issue-category identification for fine-grained analysis and downstream code refinement, we selected Deepseek-Coder-6.7B for detailed examination. This model achieved the highest prediction accuracy (Pred. Acc.), demonstrating a strong balance between review comment quality and classification performance. In terms of per-category performance, RevAgent outperforms all baselines across five metrics in the majority of issue categories. However, it underperforms LLaMA-Reviewer on METEOR and SBERT within the Logging category, likely due to the underrepresentation of Logging samples in the critic agent’s training set, which affects

Table 6. Performances of RevAgent and variants on average

Model	Method	BLEU	ROUGH-L	METEOR	SBERT	Pred. Acc.
Llama-3-8B	RevAgent	8.51	19.51	12.73	47.96	56.52%
	w/o SFT	4.43	9.70	5.79	20.90	42.89%
	w/o SFT-MCCA	6.32	13.54	8.31	34.23	31.42%
	w/o SFT-CA	4.97	11.29	7.14	26.10	44.69%
	w/o CCR	8.41	19.10	11.83	44.95	47.85%
Deepseek-Coder-6.7B	RevAgent	8.61	19.73	12.97	48.35	67.13%
	w/o SFT	5.09	11.24	6.36	16.82	60.14%
	w/o SFT-MCCA	5.84	13.08	7.08	23.59	49.60%
	w/o SFT-CA	7.03	15.97	10.23	38.47	63.57%
	w/o CCR	8.36	19.21	12.32	46.30	65.48%
Qwen2.5-Coder-7B	RevAgent	8.69	19.92	13.14	48.58	63.02%
	w/o SFT	7.07	15.45	10.53	41.63	52.83%
	w/o SFT-MCCA	7.49	16.92	10.96	43.69	56.08%
	w/o SFT-CA	8.27	18.28	12.12	45.68	47.43%
	w/o CCR	8.62	19.54	12.59	46.48	54.61%
Qwen2.5-Coder-14B	RevAgent	9.08	20.46	13.50	48.97	54.13%
	w/o SFT	7.67	17.61	10.78	42.24	58.21%
	w/o SFT-MCCA	7.76	17.73	11.43	45.65	38.45%
	w/o SFT-CA	8.94	20.18	12.92	46.84	64.16%
	w/o CCR	8.97	20.11	12.76	46.36	61.35%

its ability to accurately identify logging-related issues. A similar trend appears in the Refactoring category, where RevAgent falls behind CodeReviewer-c and TufanoLLM in issue-category prediction accuracy. Further analysis shows that both models exhibit a strong bias toward labeling most changes as Refactoring. Since Refactoring is the most frequent category in the test set, this bias artificially inflates their prediction accuracy. This behavior is rooted in the dominance of Refactoring scenarios in real-world datasets, which biases the LLMs during pre-training. In contrast, RevAgent mitigates this bias through targeted fine-tuning. By leveraging category-specific features from both code and comments, it achieves more balanced and fair classification across all issue categories. Notably, RevAgent demonstrates superior performance on challenging issue types, achieving 21.69% accuracy in the Bugfix category, where existing approaches typically achieve less than 5%. By contrast, CodeAgent performs the worst across all categories. Despite its multi-agent architecture, it lacks category-specific training and targeted knowledge alignment. As a result, agents likely fail to capture issue-dependent characteristics, yielding weak overall performance.

Answering RQ1: Compared to the best-performing baselines, RevAgent significantly improves the average performance of BLEU, ROUGE-L, METEOR, and SBERT by 12.90%, 10.87%, 6.32%, and 8.57% across four LLMs. Moreover, RevAgent demonstrates superior performance on tackling challenging issue types (i.e., bug-fix, testing, and logging).

5.2 RQ2: Ablation Experiment

This section evaluates the contribution of each component in the *Generation*, *Discrimination*, and *Training* stages to the overall performance of RevAgent. Since the *Generation* and *Discrimination Stages* focus on producing review comments, while the *Training Stage* equips them with the necessary capabilities, we first analyze the *Training Stage* in RQ2.1, followed by the analysis of the *Generation* and *Discrimination Stages* in RQ2.2.

RQ2.1: Effect of Training Stage

Analysis. To assess the contributions of each core component of *Training Stage* in RevAgent, we conducted an ablation study comparing RevAgent with four variants:

- *w/o SFT*: Replacing Supervised Fine-Tuning (SFT) with few-shot learning for both the category-specific commentator agents and the critic agent, aiming to assess the necessity of fine-tuning.
- *w/o SFT-MCCA*: Replacing the Supervised Fine-Tuned Multiple Category-specific Commentator Agents (SFT-MCCA) with their few-shot learning counterparts, isolating the impact of fine-tuning specifically for the commentator agents.
- *w/o SFT-CA*: Replacing the Supervised Fine-Tuned Critic Agent (SFT-CA) with a few-shot learning version, to assess the importance of fine-tuning specifically for the critic agent.
- *w/o CCR*: Removing the Candidate Comment Retrieval (CCR) during the construction of the critic agent’s training dataset. Instead, it uses candidate comments generated by the fine-tuned category-specific commentator agents to evaluate the role of CCR in improving issue-category identification.

Following prior works [3, 4, 38], our experiments adopt a 3-shot setting and utilize BM25 similarity [44] to retrieve relevant code changes for demonstration purposes.

Results. Table 6 presents the related results. Overall, all variants exhibit varying degrees of performance degradation compared to RevAgent. Specifically, the performance degradation of RevAgent *w/o SFT* highlights the necessity of fine-tuning both the category-specific commentator agents and the critic agent. This step is critical for enhancing these agents to generate accurate category-specific review comments and to perform effective issue-category identification. At the same time, the drop in performance observed in RevAgent *w/o SFT-MCCA* and RevAgent *w/o SFT-CA* suggests that while those agents can acquire a basic level of task capability through few-shot learning, they still underperform compared to the fine-tuned versions. Interestingly, on Qwen2.5-Coder-14B, RevAgent *w/o SFT-CA* achieves a higher prediction accuracy and close performance comparable to RevAgent across the other three metrics. This can be attributed to Qwen2.5-Coder-14B’s strong code understanding capabilities, which enable it to achieve good performance even with limited supervision. Nevertheless, fine-tuning remains essential for equipping smaller models with capabilities that exceed those of much larger models.

Furthermore, we observe that RevAgent *w/o CCR* outperforms RevAgent in prediction accuracy on Qwen2.5-Coder-14B. This may be because, compared to smaller LLMs, the category-specific commentator agents based on Qwen2.5-Coder-14B generate higher-quality candidate comments used to train the critic agent with stylistic consistency closer to the ground truth. Such consistency enables the critic agent to better focus on semantic distinctions—such as reasoning about variables and logic structures—rather than being distracted by stylistic variations. As a result, even without CCR, the critic agent trained on Qwen2.5-Coder-14B demonstrates strong issue-category identification capabilities. In contrast, with smaller LLMs, the quality of generated candidate comments is generally lower than that of retrieved comments, leading to poorer performance without CCR. Therefore, RevAgent *w/o CCR* tends to perform better on more capable models, such as Qwen2.5-Coder-14B.

Answering RQ2.1: Both Supervised Fine-Tuning and Comment Candidate Retrieval significantly enhance the *Training Stage* of RevAgent, enabling agents to achieve more accurate issue-category identification and generate higher-quality review comments.

RQ2.2: Effect of Generation and Discrimination Stages

Analysis. To further evaluate the effectiveness of the current configuration of the category-specific commentator agents in the *Generation Stage* and the critic agent in the *Discrimination Stage*, we designed alternative configurations as comparative variants:

Table 7. Performances of different configurations of two types of agent in RevAgent on average

Model	Method	BLEU	ROUGE-L	METEOR	SBERT	Pred. Acc.
Llama-3-8B	RevAgent	8.51	19.51	12.73	47.96	56.52%
	w/ SFA	7.82	18.03	12.97	47.94	42.64%
	w/ MSC	3.83	13.36	12.51	43.92	-
Deepseek-Coder-6.7B	RevAgent	8.61	19.73	12.97	48.35	67.13%
	w/ SFA	8.27	18.94	12.99	47.93	60.32%
	w/ MSC	8.13	18.35	11.62	44.39	-
Qwen2.5-Coder-7B	RevAgent	8.69	19.92	13.14	48.58	63.02%
	w/ SFA	8.34	19.01	13.17	48.52	59.95%
	w/ MSC	4.38	16.02	15.63	50.02	-
Qwen2.5-Coder-14B	RevAgent	9.08	20.46	13.50	48.97	54.13%
	w/ SFA	8.05	18.65	13.37	48.49	47.83%
	w/ MSC	4.47	16.27	15.40	49.73	-

- *with SFA*: This variant replaces the Multiple Category-specific Commentator Agents (MCCA) with a Single Fusion Agent (SFA), which is fine-tuned on the entire multi-category corpus and is responsible for generating review comments across all issue categories. This setup allows us to assess the importance of training separate agents for each specific issue category.
- *with MSC*: Instead of selecting a single best review comment from the candidate comments, this variant allows the critic to select Multiple Suitable Comments (MSC) and merge them. Specifically, the critic agent's prompt is modified as "Please read these review comments and merge the suitable review comments." This variant enables us to investigate whether merging multiple review comments can lead to better performance.

Results. The results under different configurations of the two types of agents in RevAgent are shown in Table 7. Overall, RevAgent outperforms both RevAgent *with SFA* and RevAgent *with MSC* across the majority of evaluation metrics. Interestingly, RevAgent *with SFA* shows slightly higher METEOR scores. This is primarily due to the fusion agent, trained on the entire dataset, tending to generate comments with more common tokens. Consequently, as the critic agent misclassifies the issue category, the generated comments may still exhibit greater token-level overlap with the ground truth, leading to inflated METEOR scores despite being semantically inaccurate. Similarly, we observe that RevAgent *with MSC* consistently yields lower BLEU and ROUGE-L scores across all four LLMs. However, for both versions of Qwen2.5-Coder, this variant achieves higher scores on METEOR and SBERT. This discrepancy can be attributed to the merging of multiple review comments, which decreases token-level similarity with the ground truth, leading to lower BLEU and ROUGE-L scores, while enhancing semantic richness, thereby improving performance on metrics that emphasize semantic similarity, such as METEOR and SBERT. These findings indicate that both the commentator agents and the critic agent are essential to RevAgent's effectiveness. Notably, when comparing the performance differences across the two variants, RevAgent *with MSC* exhibits more pronounced changes, showing significant declines in BLEU and ROUGE-L, alongside noticeable improvements in METEOR and SBERT. This suggests that the critic agent plays a particularly critical role in the overall system.

Answering RQ2.2: The category-specific commentator agents and the critic agent both contribute to RevAgent's effectiveness in terms of textual similarity and prediction accuracy, with the critic agent playing the most crucial role.

Table 8. Results of the human evaluation on 1,920 generated comments from RevAgent and four baselines

Metric	Method	Avg.	Std.
Readability	CodeReviewer	2.3	1.2
	LLaMA-Reviewer	3.1	1.0
	TufanoLLM	2.6	1.3
	CodeAgent	1.3	0.8
	RevAgent	3.8	0.7
Accuracy	CodeReviewer	2.3	1.3
	LLaMA-Reviewer	2.8	1.3
	TufanoLLM	2.3	1.3
	CodeAgent	1.3	0.8
	RevAgent	3.6	1.0
Category-Matching	CodeReviewer	2.2	1.1
	LLaMA-Reviewer	2.8	0.8
	TufanoLLM	2.5	1.3
	CodeAgent	1.2	0.6
	RevAgent	3.5	0.9

5.3 RQ3: Human Evaluation

Analysis. Although the four existing evaluation metrics measure lexical and semantic differences between generated comments and ground truth, they often fail to capture real semantic disparities. To comprehensively evaluate the quality of review comments generated by various approaches, we rigorously conducted a human evaluation. Following prior works [33, 46, 61], we randomly selected 384 samples (achieving a 95% confidence level with a confidence interval of less than 3%) while maintaining proportional representation across issue categories in the test set (ensuring at least one sample from each category). This process resulted in 1,920 generated comments for evaluation from five approaches (i.e., CodeReviewer, LLaMA-Reviewer, TufanoLLM, CodeAgent, and RevAgent) on Deepseek-Coder-6.7B. Consistent with prior studies [46], we adopted a reference-free evaluation, which allows evaluators to exercise subjective judgment, recognize reasonable comments beyond reference matches, assess additional dimensions (e.g., readability), and reduce bias from poor-quality references [6, 16]. We recruited two participants with over five years of Java development experience, who are not co-authors of this paper. Among the 384 randomly selected samples, the first evaluator reviewed the first 242 samples, and the second evaluator reviewed the last 242, with 100 overlapping samples in the middle used to measure inter-rater agreement via Cohen’s kappa. We also ensured that each evaluator assessed an equal number of cases from each issue category to provide balanced evaluation coverage. To ensure fairness, evaluators were not informed of the source of the comments. Each comment was rated across three dimensions: (1) **Readability**, which reflects the fluency, clarity, specificity, and ease of understanding of generated review comments; (2) **Accuracy**, which reflects whether they clearly point out code issues rather than being vague, and whether there are any incorrect suggestions; (3) **Category-Matching**, which reflects whether the review comments are relevant to the target issue categories. All ratings were given on a 5-point Likert scale: 1 for poor, 2 for marginal, 3 for acceptable, 4 for good, and 5 for excellent. The Cohen’s Kappa coefficient between the two evaluators is 0.74, indicating substantial agreement and confirming the reliability of their assessments.

Results. Table 8 summarizes the related human evaluation results. Overall, RevAgent outperforms all baselines across the three evaluation dimensions: Readability, Accuracy, and Category-Matching.

Case 1 (Refactoring):

```
public boolean isFocusTraversable() {
-   Component comp = getTextComponent();
-   if (comp instanceof JTextComponent)
-       if (((JTextComponent)comp).isEditable())
+   JTextComponent comp = getTextComponent();
+   if (comp != null)
+       if (comp.isEditable())
        return true;
}
```

Ground Truth: Consider merging the two conditional statements into a single line, e.g., `\`return comp != null && comp.isEditable();\``.

CodeReviewer: Consider adding type hint for the `'scale'` and `'scaling'` parameters in the `'draw'` method to improve code readability and maintainability.

LLaMA-Reviewer: Consider simplifying the `'isFocusTraversable'` method by directly returning `'getTextComponent()' instanceof JTextComponent'`.

TufanoLLM: The code is refactored to use a more specific type for the variable `'comp'` which is more accurate and helps to avoid potential null pointer exceptions.

CodeAgent: Based on the above analysis, we suggest no revisions to the code.

RevAgent: Consider simplifying the code by directly checking the result of `'getTextComponent()'` and `'isEditable()'` in a single line, like this: `'return getTextComponent() != null && getTextComponent().isEditable();'`.

Case 2 (Bugfix):

```
func getDefaultEndpoint(ip, port string) string {
    if ip == "" {
        var err error
        if ip, err = utils.GetIPAddress(); err != nil {
            panic(err)
+           //fmt.Fprintf(os.Stderr, "ERROR:Unable to get default endpoint:%s\n", err)
+           return ""
        }
    }
}
```

Ground Truth: Consider handling the error more robustly instead of returning an empty string. Should we log the error or panic here as well?

CodeReviewer: Consider removing the `'OnStart'` method as it appears unnecessary. Can you confirm if it's still required or if it can be removed?

LLaMA-Reviewer: Consider using `'os.Stderr'` instead of `'fmt.Fprintf'` for error logging to maintain consistency with the original panic approach.

TufanoLLM: The code is throwing a panic when it fails to get the IP address. This is not a good practice as it can cause the application to crash.

CodeAgent: I'm sorry, but as a CodeAgent, I'm unable to write a manual.md file.

RevAgent: Consider returning an error instead of returning an empty string when the default endpoint cannot be retrieved.

Fig. 2. Generated comments by RevAgent and baselines

For example, RevAgent outperforms LLaMA-Reviewer by 0.7, 0.8, and 0.7 points, respectively. This demonstrates that RevAgent is capable of generating review comments that align well with human judgment. In particular, RevAgent significantly surpasses the baselines in both generating accurate review comments and correctly identifying the issue categories present in code diffs. Moreover, its lower standard deviation in Readability and Accuracy indicates more consistent performance across different cases. We also observe that TufanoLLM, due to its open-ended generation style, achieves better Readability and Category-Matching scores than CodeReviewer. In contrast, LLaMA-Reviewer, which benefits from supervised fine-tuning, consistently outperforms the CodeT5-based CodeReviewer across all three evaluation metrics. Finally, CodeAgent performs the worst among all methods, primarily because its excessive openness, where all steps are autonomously executed by

Table 9. The average inference consumption of RevAgent and baselines.

LLM	Time Consumption				Token Consumption			
	LLaMA-Reviewer	TufanoLLM	CodeAgent	RevAgent	LLaMA-Reviewer	TufanoLLM	CodeAgent	RevAgent
LLaMA-3-8B	0.016	0.055	212.603	0.049	232	420	39,143	2,338
Deepseek-Coder-6.7B	0.019	0.202	238.670	0.056	294	532	99,713	2,796
Qwen2.5-Coder-7B	0.015	0.027	183.868	0.044	247	427	6,887	2,371
Qwen2.5-Coder-14B	0.026	0.047	163.263	0.077	247	427	7,096	2,366
Average	0.019	0.083	240.417	0.057	255	452	39,984	2,468

agents without explicit guidance, leads to unstable and inaccurate results when handling complex code review tasks. Figure 2 presents a refactoring case and a bugfix case. In the refactoring case, the code diff contains multiple nested conditional statements that should be merged. While the other baselines failed to identify the core issue, focusing instead on aspects such as variable declarations and function signatures, RevAgent correctly detects the problem of deeply nested conditions and provides a feasible refactoring suggestion. In the bug-fix case, RevAgent successfully identifies the issue with the abnormal return value in the code diff, further highlighting its superiority in precise issue detection.

Answering RQ3: In human evaluation, compared to the baselines, RevAgent achieves the highest scores on Readability, Accuracy, and Category-Matching, demonstrating its effectiveness in generating high-quality comments that help developers identify and understand code issues.

6 Discussion

Inference Consumption Analysis. Although the preceding RQ results demonstrate the effectiveness of our approach, the efficiency of the agent-based framework, particularly for LLM-based agents, should also be considered for practical applicability. To assess the efficiency of RevAgent, we measured both average token consumption and inference time across four LLMs. As shown in Table 9, RevAgent increases token usage by 8.68 $\times$ and 4.47 $\times$ compared to LLaMA-Reviewer and TufanoLLM, respectively, but only 0.06 $\times$ that of CodeAgent. While the average token count is relatively high (2,468 tokens on average), it remains acceptable in practice, as we utilize open-source LLMs that do not incur additional token-based costs. Moreover, RevAgent’s token consumption is significantly lower than that of the multi-agent-based CodeAgent. In terms of inference time, RevAgent introduces a 1.97 $\times$ latency increase compared to LLaMA-Reviewer, but operates 0.46 $\times$ faster than TufanoLLM. Despite employing an agent-based architecture, the additional 0.038 seconds per prediction compared to LLaMA-Reviewer falls within a reasonable trade-off. Furthermore, compared with CodeAgent’s multi-agent design, our framework is considerably lighter, achieving substantially lower inference time due to its streamlined and well-coordinated agent interactions. In summary, considering the substantial performance improvements achieved by RevAgent, the framework maintains a favorable balance between effectiveness and efficiency, making it suitable for practical large-scale code review applications.

Role of Issue Prediction in Review Comment Generation. To further investigate the influence of issue-category prediction on comment generation, we analyzed the quality of review comments generated by RevAgent under both correct and incorrect issue predictions using Deepseek-Coder-6.7B. Specifically, when the issue category is correctly predicted, RevAgent achieves BLEU, ROUGE-L, METEOR, and SBERT scores of 9.08, 20.68, 13.50, and 49.25, respectively. In cases of misprediction, these scores decrease to 7.65, 17.79, 11.90, and 46.51, indicating a consistent decline in performance across all metrics. This degradation can be attributed to the role of issue categories in choosing the

correct commentator agents to focus on the most relevant issues of the code change. Accurate prediction ensures that the corresponding category-specific agent is better aligned with the underlying issue in the code change, thereby facilitating the generation of precise and contextually appropriate comments. Conversely, incorrect predictions may lead to an ill-suited agent, resulting in comments that are either overly generic or fail to address the core issue. These findings underscore the critical role of issue-category prediction in enhancing the quality and relevance of generated comments in RevAgent.

Root Causes Behind Low-Quality Generated Comments. Upon our human evaluation, we further conducted a manual analysis of low-quality generated comments (i.e., 69 with average scores below 2) to identify their root causes and provide insights to inform future improvements. Three root causes are summarized with their frequency (an instance could contain more than one cause): (1) *Lack of business logic* (22%). Many real-world code changes are driven by business requirements rather than functional correctness. Without access to business-specific rationale, current methods often misinterpret the intent of changes or flag them as erroneous. (2) *Confusion over project-specific coding standards* (48%). Code edits typically need to follow project-specific coding standards. However, models trained on datasets aggregated from diverse repositories may fail to capture these localized standards (e.g., import ordering), resulting in incorrect or inconsistent comments. (3) *Limited contextual information* (43%). Omitting full function or class definitions restricts the model’s ability to capture semantic dependencies beyond the changed lines, causing it to miss context-dependent issues. These findings highlight the need for repository-level code review approaches that incorporate business logic, enforce project-specific standards, and utilize broader contextual information to improve the accuracy and relevance of generated comments.

7 Threats to Validity

Internal Validity. A potential threat is the data leakage risk in LLMs, as they are trained on open-source projects and may have encountered some test cases during training. However, our analysis reveals that the LLMs employed in this work exhibit suboptimal performance in zero-shot settings, indicating that their outputs are not solely derived from memorization. Moreover, given the complexity of the code review task, it is inherently unlikely that LLMs can generate results solely through memorization. This concern has been similarly acknowledged in other LLM studies [15, 35]. Another potential threat lies in the potential positional bias in the fine-tuned critic agent. Prior studies [10, 41] show that while discriminative LLMs can exhibit permutation invariance, fine-tuning does not always eliminate such effects. To mitigate this, we adopt a structured input format to enforce consistent element mapping, though minor bias may remain. Finally, to mitigate the threat of the inherent randomness in the LLM inference process, we set the temperature to 0 to stable answers from the LLMs, facilitating reproducibility in future research.

External Validity. The primary threat to external validity is that the dataset was annotated using LLaMA-3.1-70B rather than fully human-labeled, which may introduce annotation errors [46]. To mitigate this, all experiments were conducted on the same dataset, ensuring consistent comparisons and minimizing the impact of potential noise. Another limitation lies in the class imbalance of the test set, where Refactoring accounts for a disproportionately large share. This skews overall performance metrics and may introduce bias. We address this by reporting detailed, category-level results, which highlight RevAgent’s effectiveness across diverse issue types. Finally, our study fine-tuned only a few representative LLMs with parameter sizes under 14B and did not include proprietary models such as GPT-4o due to resource constraints. We regard our work as a foundation and remain open to incorporating more advanced and larger models in future studies.

Construct Validity. The primary threat lies in the human evaluation of comment quality, which may be biased due to inherent subjectivity. To mitigate this, we recruited two participants with substantial experience in Java development, who are independent and not involved in the authorship of this paper. Furthermore, following established practices, we systematically assessed their consistency and comprehension using the inter-rater agreement metric (i.e., FCohen’s kappa) during the manual evaluation.

8 Conclusion

In this paper, we propose RevAgent, a novel agent-based framework for code review comment generation, addressing the limitations of existing approaches that rely on a single model and overlook the diverse nature of issues in code changes. Our evaluation on large-scale datasets shows that RevAgent outperforms state-of-the-art baselines in both textual quality metrics and issue category prediction accuracy, underscoring its overall effectiveness. Human evaluation further confirms the quality and usefulness of the generated comments. Moreover, RevAgent strikes a favorable balance between performance and efficiency. Meanwhile, our work also opens up several promising future directions, including constructing the fine-grained issue typing beyond the current taxonomy, elevating the target of code review from individual code changes to the function or class level, and leveraging external sources (e.g., repository-level documentation or history information) to support more accurate and context-aware code review.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant Nos. 62472310, 62322208.

References

- [1] [n. d.]. <https://anonymous.4open.science/r/RevAgent-3775/>.
- [2] 2024. Introducing Meta Llama 3: The most capable openly available LLM to date. <https://ai.meta.com/blog/meta-llama-3/>.
- [3] Toufique Ahmed and Premkumar T. Devanbu. 2022. Few-shot training LLMs for project-specific code-summarization. In *ASE. ACM*, 177:1–177:5.
- [4] Toufique Ahmed, Kunal Suresh Pai, Premkumar T. Devanbu, and Earl T. Barr. 2024. Automatic Semantic Augmentation of Language Model Prompts (for Code Summarization). In *ICSE. ACM*, 220:1–220:13.
- [5] Elif Akata, Lion Schulz, Julian Coda-Forno, Seong Joon Oh, Matthias Bethge, and Eric Schulz. 2023. Playing repeated games with Large Language Models. *CoRR* abs/2305.16867 (2023).
- [6] Abbas Akkasi, Kathleen C. Fraser, and Majid Komeili. 2023. Reference-Free Summarization Evaluation with Large Language Models. In *Eval4NLP. Association for Computational Linguistics*, 193–201.
- [7] Mohammadmehdi Ataei, Hyunmin Cheong, Daniele Grandi, Ye Wang, Nigel Morris, and Alexander Tessier. 2024. Elicitron: An LLM Agent-Based Simulation Framework for Design Requirements Elicitation. *CoRR* abs/2404.16045 (2024).
- [8] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 712–721.
- [9] Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments. In *IEEevaluation@ACL. Association for Computational Linguistics*, 65–72.
- [10] Qiming Bao, Gaël Gendron, Alex Yuxuan Peng, Wanjun Zhong, Neset Tan, Yang Chen, Michael Witbrock, and Jiamou Liu. 2024. Assessing and Enhancing the Robustness of Large Language Models with Task Structure Variations for Logical Reasoning. In *ICONIP (10) (Lecture Notes in Computer Science, Vol. 15295)*. Springer, 313–327.
- [11] Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. 2024. Large Language Models as Tool Makers. In *ICLR. OpenReview.net*.
- [12] Ján Cegin, Jakub Simko, and Peter Brusilovsky. 2024. LLMs vs Established Text Augmentation Techniques for Classification: When do the Benefits Outweight the Costs? *CoRR* abs/2408.16502 (2024).
- [13] Yung-Chieh Chan, George Pu, Apaar Shanker, Parth Suresh, Penn Jenks, John Heyer, and Sam Denton. 2024. Balancing Cost and Effectiveness of Synthetic Data Generation Strategies for LLMs. *arXiv:2409.19759 [cs.CL]* <https://arxiv.org/>

- abs/2409.19759
- [14] Seyed Shayan Daneshvar, Yu Nong, Xu Yang, Shaowei Wang, and Haipeng Cai. 2024. Exploring RAG-based Vulnerability Augmentation with LLMs. *CoRR* abs/2408.04125 (2024).
 - [15] Mingyang Geng, Shangwen Wang, Dezun Dong, Haotian Wang, Ge Li, Zhi Jin, Xiaoguang Mao, and Xiangke Liao. 2024. Large Language Models are Few-Shot Summarizers: Multi-Intent Comment Generation via In-Context Learning. In *ICSE*. ACM, 39:1–39:13.
 - [16] Théo Gigant, Camille Guinaudeau, Marc Decombas, and Frédéric Dufaux. 2024. Mitigating the Impact of Reference Quality on Evaluation of Summarization Systems with Reference-Free Metrics. In *EMNLP*. Association for Computational Linguistics, 19355–19368.
 - [17] Saul Goldman, Hong Yi Lin, Jirat Pasuksmit, Patanamon Thongtanunam, Kla Tantithamthavorn, Zhe Wang, Ray Zhang, Ali Behnaz, Fan Jiang, Michael Siers, Ryan Jiang, Mike Buller, Minwoo Jeong, and Ming Wu. 2025. What Types of Code Review Comments Do Developers Most Frequently Resolve? arXiv:2510.05450 [cs.SE] <https://arxiv.org/abs/2510.05450>
 - [18] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y.K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. DeepSeek-Coder: When the Large Language Model Meets Programming – The Rise of Code Intelligence. <https://arxiv.org/abs/2401.14196>
 - [19] Qi Guo, Junming Cao, Xiaofei Xie, Shangqing Liu, Xiaohong Li, Bihuan Chen, and Xin Peng. 2024. Exploring the Potential of ChatGPT in Automated Code Refinement: An Empirical Study. In *ICSE*. ACM, 34:1–34:13.
 - [20] Sakib Haque, Zachary Eberhart, Aakash Bansal, and Collin McMillan. 2022. Semantic similarity metrics for evaluating source code summarization. In *ICPC*. ACM, 36–47.
 - [21] Junda He, Christoph Treude, and David Lo. 2025. LLM-Based Multi-Agent Systems for Software Engineering: Literature Review, Vision, and the Road Ahead. *ACM Transactions on Software Engineering and Methodology* 34, 5 (2025), 1–30.
 - [22] Yang Hong, Chakkrit Tantithamthavorn, Patanamon Thongtanunam, and Aldeida Aleti. 2022. CommentFinder: a simpler, faster, more accurate code review comments recommendation. In *ESEC/SIGSOFT FSE*. ACM, 507–519.
 - [23] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. 2021. LoRA: Low-Rank Adaptation of Large Language Models. *CoRR* abs/2106.09685 (2021).
 - [24] Quzhe Huang, Mingxu Tao, Chen Zhang, Zhenwei An, Cong Jiang, Zhibin Chen, Zirui Wu, and Yansong Feng. 2023. Lawyer LLaMA Technical Report. arXiv:2305.15062 [cs.CL] <https://arxiv.org/abs/2305.15062>
 - [25] Md. Ashraful Islam, Mohammed Eunus Ali, and Md. Rizwan Parvez. 2024. MapCoder: Multi-Agent Code Generation for Competitive Problem Solving. In *ACL (1)*. Association for Computational Linguistics, 4912–4944.
 - [26] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. CAMEL: Communicative Agents for "Mind" Exploration of Large Scale Language Model Society. *CoRR* abs/2303.17760 (2023).
 - [27] Lingwei Li, Li Yang, Huaxi Jiang, Jun Yan, Tiejian Luo, Zihan Hua, Geng Liang, and Chun Zuo. 2022. AUGER: automatically generating review comments with pre-training models. In *ESEC/SIGSOFT FSE*. ACM, 1009–1021.
 - [28] Yunxiang Li, Zihan Li, Kai Zhang, Ruilong Dan, and You Zhang. 2023. ChatDoctor: A Medical Chat Model Fine-tuned on LLaMA Model using Medical Domain Knowledge. *CoRR* abs/2303.14070 (2023).
 - [29] Zhiyu Li, Shuai Lu, Daya Guo, Nan Duan, Shailesh Jannu, Grant Jenks, Deep Majumder, Jared Green, Alexey Svyatkovskiy, Shengyu Fu, and Neel Sundaresan. 2022. Automating code review activities by large-scale pre-training. In *ESEC/SIGSOFT FSE*. ACM, 1035–1047.
 - [30] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. *Meeting of the Association for Computational Linguistics, Meeting of the Association for Computational Linguistics* (Jul 2004).
 - [31] Junyi Lu, Lei Yu, Xiaojia Li, Li Yang, and Chun Zuo. 2023. LLaMA-Reviewer: Advancing Code Review Automation with Large Language Models through Parameter-Efficient Fine-Tuning. In *ISSRE*. IEEE, 647–658.
 - [32] Laura MacLeod, Michaela Greiler, Margaret-Anne Storey, Christian Bird, and Jacek Czerwinka. 2017. Code reviewing in the trenches: Challenges and best practices. *IEEE Software* 35, 4 (2017), 34–42.
 - [33] Fangwen Mu, Xiao Chen, Lin Shi, Song Wang, and Qing Wang. 2023. Developer-Intent Driven Code Comment Generation. In *ICSE*. IEEE, 768–780.
 - [34] Mona Nashaat and James Miller. 2024. Towards Efficient Fine-Tuning of Language Models With Organizational Data for Automated Software Review. *IEEE Trans. Software Eng.* 50, 9 (2024), 2240–2253.
 - [35] Noor Nashid, Mifta Sintaha, and Ali Mesbah. 2023. Retrieval-Based Prompt Selection for Code-Related Few-Shot Learning. In *ICSE*. IEEE, 2450–2462.
 - [36] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *ACL*. ACL, 311–318.
 - [37] Joon Sung Park, Joseph C. O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative Agents: Interactive Simulacra of Human Behavior. In *UIST*. ACM, 2:1–2:22.
 - [38] Ajay Patel, Bryan Li, Mohammad Sadeh Rasooli, Noah Constant, Colin Raffel, and Chris Callison-Burch. 2023. Bidirectional Language Models Are Also Few-shot Learners. In *ICLR*. OpenReview.net.

- [39] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ChatDev: Communicative Agents for Software Development. In *ACL (1)*. Association for Computational Linguistics, 15174–15186.
- [40] Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiayi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 Technical Report. arXiv:2412.15115 [cs.CL] <https://arxiv.org/abs/2412.15115>
- [41] Bianca Raimondi and Maurizio Gabbriellini. 2025. Exploiting Primacy Effect To Improve Large Language Models. *CoRR* abs/2507.13949 (2025).
- [42] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *EMNLP/IJCNLP (1)*. Association for Computational Linguistics, 3980–3990.
- [43] Peter C Rigby and Christian Bird. 2013. Convergent contemporary software peer review practices. In *Proceedings of the 2013 9th joint meeting on foundations of software engineering*. 202–212.
- [44] Stephen E. Robertson and Hugo Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends Inf. Retr.* 3, 4 (2009), 333–389.
- [45] Oussama Ben Sghaier and Houari A. Sahraoui. 2024. Improving the Learning of Code Review Successive Tasks with Cross-Task Knowledge Distillation. *Proc. ACM Softw. Eng.* 1, FSE (2024), 1086–1106.
- [46] Oussama Ben Sghaier, Martin Weyssow, and Houari A. Sahraoui. 2025. Harnessing Large Language Models for Curated Code Reviews. *CoRR* abs/2502.03425 (2025).
- [47] Tao Sun, Jian Xu, Yuanpeng Li, Zhao Yan, Ge Zhang, Lintao Xie, Lu Geng, Zheng Wang, Yueyan Chen, Qin Lin, Wenbo Duan, and Kaixin Sui. 2025. BitsAI-CR: Automated Code Review via LLM in Practice. *CoRR* abs/2501.15134 (2025).
- [48] Yashar Talebirad and Amirhossein Nadiri. 2023. Multi-Agent Collaboration: Harnessing the Power of Intelligent LLM Agents. *CoRR* abs/2306.03314 (2023).
- [49] Xunzhu Tang, Kisub Kim, Yewei Song, Cedric Lothritz, Bei Li, Saad Ezzini, Haoye Tian, Jacques Klein, and Tegawendé F. Bissyandé. 2024. CodeAgent: Autonomous Communicative Agents for Code Review. In *EMNLP*. Association for Computational Linguistics, 11279–11313.
- [50] Rosalia Tufano, Ozren Dabic, Antonio Mastropaolo, Matteo Ciniselli, and Gabriele Bavota. 2024. Code Review Automation: Strengths and Weaknesses of the State of the Art. *IEEE Trans. Software Eng.* 50, 2 (2024), 338–353.
- [51] Rosalia Tufano, Simone Masiero, Antonio Mastropaolo, Luca Pascarella, Denys Poshyvanyk, and Gabriele Bavota. 2022. Using Pre-Trained Models to Boost Code Review Automation. In *ICSE*. ACM, 2291–2302.
- [52] Rosalia Tufano, Luca Pascarella, Michele Tufano, Denys Poshyvanyk, and Gabriele Bavota. 2021. Towards Automating Code Review Activities. In *ICSE*. IEEE, 163–174.
- [53] Dong Wang, Yuki Ueda, Raula Gaikovina Kula, Takashi Ishio, and Kenichi Matsumoto. 2021. Can we benchmark code review studies? a systematic mapping study of methodology, dataset, and metric. *Journal of Systems and Software* 180 (2021), 111009.
- [54] Dong Wang, Tao Xiao, Patanamon Thongtanunam, Raula Gaikovina Kula, and Kenichi Matsumoto. 2021. Understanding shared links and their intentions to meet information needs in modern code review: A case study of the OpenStack and Qt projects. *Empirical Software Engineering* 26, 5 (2021), 96.
- [55] Ziliang Wang, Ge Li, Jia Li, Jia Li, Meng Yan, Yingfei Xiong, and Zhi Jin. 2025. M2CVD: Enhancing Vulnerability Understanding through Multi-Model Collaboration for Code Vulnerability Detection. *ACM Transactions on Software Engineering and Methodology* (2025).
- [56] Zaitian Wang, Jinghan Zhang, Xinhao Zhang, Kunpeng Liu, Pengfei Wang, and Yuanchun Zhou. 2025. Diversity-Oriented Data Augmentation with Large Language Models. *CoRR* abs/2502.11671 (2025).
- [57] Jimmy Wei, Kurt Shuster, Arthur Szlam, Jason Weston, Jack Urbanek, and Mojtaba Komeili. 2023. Multi-Party Chat: Conversational Agents in Group Settings with Humans and Models. arXiv:2304.13835 [cs.CL] <https://arxiv.org/abs/2304.13835>
- [58] Robert F Woolson. 2007. Wilcoxon signed-rank test. *Wiley encyclopedia of clinical trials* (2007), 1–3.
- [59] Yifan Wu, Yunpeng Wang, Ying Li, Wei Tao, Siyu Yu, Haowen Yang, Wei Jiang, and Jianguo Li. 2025. An Empirical Study on Commit Message Generation Using LLMs via In-Context Learning. In *ICSE*. IEEE, 553–565.
- [60] Yuhan Wu, Ziao Wang, Lei Huang, Junjie Wang, Jianping Li, and Mingyong Huang. 2025. CodeDoctor: multi-category code review comment generation. *Autom. Softw. Eng.* 32, 1 (2025), 25.
- [61] Yongda Yu, Guoping Rong, Haifeng Shen, He Zhang, Dong Shao, Min Wang, Zhao Wei, Yong Xu, and Juhong Wang. 2025. Fine-Tuning Large Language Models to Improve Accuracy and Comprehensibility of Automated Code Review. *ACM Trans. Softw. Eng. Methodol.* 34, 1 (2025), 14:1–14:26.

- [62] Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. 2024. CodeAgent: Enhancing Code Generation with Tool-Integrated Agent Systems for Real-World Repo-level Coding Challenges. In *ACL (1)*. Association for Computational Linguistics, 13643–13658.
- [63] Yuxia Zhang, Zhiqing Qiu, Klaas-Jan Stol, Wenhui Zhu, Jiaxin Zhu, Yingchen Tian, and Hui Liu. 2024. Automatic Commit Message Generation: A Critical Review and Directions for Future Work. *IEEE Trans. Software Eng.* 50, 4 (2024), 816–835.
- [64] Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyang Luo, Zhangchi Feng, and Yongqiang Ma. 2024. LlamaFactory: Unified Efficient Fine-Tuning of 100+ Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*. Association for Computational Linguistics, Bangkok, Thailand. <http://arxiv.org/abs/2403.13372>