

SECUREVIEWER: Enhancing Large Language Models for Secure Code Review through Secure-aware Fine-tuning

Fang Liu¹, Simiao Liu¹, Yinghao Zhu¹, Xiaoli Lian¹, Li Zhang^{1*}

¹State Key Laboratory of Complex & Critical Software Environment, School of Computer Science and Engineering,
Beihang University, China
 {fangliu,buaalsm,zhuyinghao,lianxiaoli,lily}@buaa.edu.cn

Abstract

Identifying and addressing security issues during the early phase of the development lifecycle is critical for mitigating the long-term negative impacts on software systems. Code review serves as an effective practice that enables developers to check their teammates' code before integration into the codebase. To streamline the generation of review comments, various automated code review approaches have been proposed, where Large Language Model (LLM)-based methods have significantly advanced the capabilities of automated review generation. However, existing models primarily focus on general-purpose code review, their effectiveness in identifying and addressing security-related issues remains under-explored. Moreover, adapting existing code review approaches to target security issues faces substantial challenges, including data scarcity and inadequate evaluation metrics. To address these limitations, we propose SECUREVIEWER, a new approach designed for enhancing LLMs' ability to identify and resolve security-related issues during code review. Specifically, we first construct a dataset tailored for training and evaluating secure code review capabilities. Leveraging this dataset, we fine-tune LLMs to generate code review comments that can effectively identify security issues and provide fix suggestions with our proposed secure-aware fine-tuning strategy. To mitigate hallucination in LLMs and enhance the reliability of their outputs, we integrate the Retrieval-Augmented Generation (RAG) technique, which grounds the generated comments in domain-specific security knowledge. Additionally, we introduce SecureBLEU, a new evaluation metric designed to assess the effectiveness of review comments in addressing security issues. Experimental results demonstrate that SECUREVIEWER outperforms state-of-the-art baselines in both security issue detection accuracy and the overall quality and practical utility of generated review comments. Our code and data are available at <https://github.com/SIMIAO515/SecureReviewer>.

CCS Concepts

- Software and its engineering; • Computing methodologies
 → Artificial intelligence;

*Corresponding author.



This work is licensed under a Creative Commons Attribution 4.0 International License.
ICSE '26, Rio de Janeiro, Brazil
 © 2026 Copyright held by the owner/author(s).
 ACM ISBN 979-8-4007-2025-3/26/04
<https://doi.org/10.1145/3744916.3773191>

Keywords

Code Review, Software Security, Large Language Models

ACM Reference Format:

Fang Liu¹, Simiao Liu¹, Yinghao Zhu¹, Xiaoli Lian¹, Li Zhang^{1*}. 2026. SECUREVIEWER: Enhancing Large Language Models for Secure Code Review through Secure-aware Fine-tuning. In *2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE '26), April 12–18, 2026, Rio de Janeiro, Brazil*. ACM, New York, NY, USA, 13 pages. <https://doi.org/10.1145/3744916.3773191>

1 Introduction

As software systems play increasingly critical roles in society, security vulnerabilities can have profound consequences for businesses and individuals [3, 34]. To mitigate these risks, modern software development adopts proactive "shift-left" practices [11, 24] that integrate security testing into earlier development phases. Code review serves as a key preventive measure in this paradigm, where developers submit code changes for systematic evaluation to identify and address issues before codebase integration [15, 47]. For example, Heartbleed (CVE-2014-0160) [37], a famous OpenSSL vulnerability from improper input validation, could have been prevented through effective code review [13]. Furthermore, Bavota and Russo [3] find that unreviewed commits are more than twice as likely to introduce bugs and are less readable than reviewed ones.

There are several empirical studies that explore the role of code review in finding and mitigating security issues [5, 55, 56]. While these studies identify key challenges and limitations in practice and provide valuable recommendations and insights for improving secure code reviews, they do not offer automated solutions to systematically address these issues. To efficiently generate review comments and reduce reliance on manual effort, recent years have seen the emergence of automated code review approaches, leveraging the rapid advancements in deep learning technologies [20, 43, 46].

For example, Gupta and Sundaresan [20] introduce an LSTM-based model designed to analyze the relationships between code changes and review comments, and recommends review comments automatically based on existing code reviews and code changes. Building on advances in Transformer [48] architectures and pre-trained models [16, 40, 51], researchers have developed code review systems through two primary approaches: pre-training models on code review-specific tasks [30] or fine-tuning large language models (LLMs) for code review applications [32, 57]. These approaches, particularly LLM-based methods, have significantly advanced the capabilities of automated review comment generation, pushing the boundaries of what is achievable in this domain and creating new opportunities for secure code review practices. However, existing models focus primarily on general-purpose code review, and their

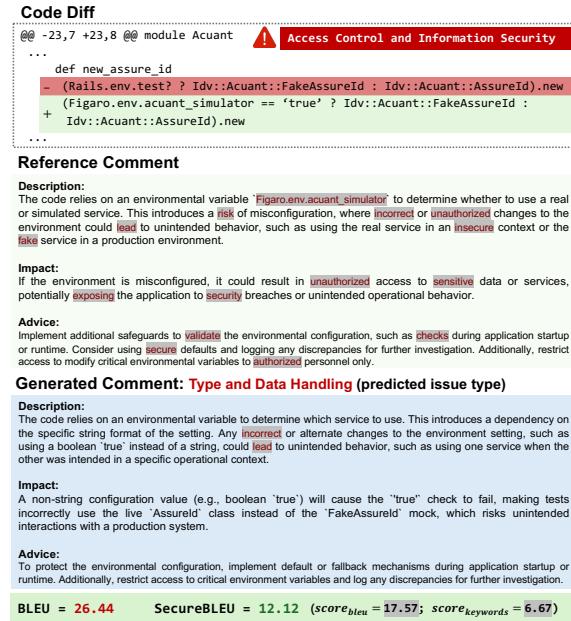


Figure 1: A code review comment with its SecureBLEU score.

effectiveness in identifying and addressing security-related issues remains unexplored [2, 49]. Furthermore, adapting current code review approaches to specifically target security issues faces the following challenges:

① Noisy Code Review Dataset: Existing commonly used code review datasets [30, 46] are primarily collected from generative-purpose review comment in open-source projects, where many of the comments may lack substantive content and are often unrelated to identifying actual issues [57]. For instance, the comments often contain non-informative content such as mentions of authors' names or generic statements like "Looks good to me" or "Why do we need this?", rather than pinpointing specific issues. Moreover, there is a scarcity of high-quality, security-focused datasets specifically constructed for training and evaluating code review models.

② Inadequate Metric: BLEU score [39] is widely adopted in assessing the quality of the review comment by measuring the n-gram overlap between the ground truth and predicted comment [30, 32], which fails to fully assess the effectiveness of the comments in detecting and resolving security issues. As illustrated in Figure 1, even though the generated comment incorrectly classified the security issue from "Access Control and Information Security" to "Type and Data Handling" in the code diff, the BLEU score remains relatively high due to the surface-level linguistic similarity between the generated comment and the ground truth.

We propose SECUREVIEWER to enhance an LLM's security code review capabilities. Our approach first involves an automated data workflow, which integrates LLMs and heuristic rules to build a tailored dataset for training and evaluation. Leveraging this dataset, we devise a security-aware fine-tuning strategy that trains the model to generate precise comments identifying security vulnerabilities and proposing actionable fixes. To further improve comment relevance and mitigate hallucinations, we employ Retrieval-Augmented Generation (RAG) [28], which grounds the generation process by

retrieving relevant examples from a prebuilt datastore of review templates.

To assess comment quality, we introduce SecureBLEU, a novel metric designed to evaluate the effectiveness of comments in identifying and resolving security issues. We evaluate SECUREVIEWER on our constructed security dataset, performing a comprehensive comparison against state-of-the-art (SOTA) code review baselines and leading LLMs. The results demonstrate that SECUREVIEWER surpasses these baselines in both security issue detection accuracy and the overall quality of generated comments. In summary, our contributions are:

- We design an automated data collection and refinement pipeline to construct the dataset specially designed for training and evaluating the model's capabilities of secure code review.
- We propose a secure-aware fine-tuning strategy, enhancing LLM to focus on generating code review comments that can effectively identify security issues and provide fix suggestions.
- We design SecureBLEU, a new evaluation metric for assessing the quality of code review comments by incorporating domain-specific relevance to security.
- We conduct a comprehensive comparison between SECUREVIEWER and state-of-the-art baselines. The evaluation results demonstrate the effectiveness and practicality of our model.

2 Methodology

Figure 2 presents the overview of SECUREVIEWER. Our approach begins with the construction of a high-quality dataset to enable effective training and evaluation of the model's secure code review capabilities. Based on our dataset, we fine-tune LLM to focus on generating code review comments that can precisely identify security issues and provide fix suggestions with our proposed secure-aware fine-tuning strategy. Finally, we integrate the RAG technique to enhance the relevance and reliability of generated review comments.

2.1 Data Collection and Refining

As illustrated in Figure 3, we design an automated data collection and refinement pipeline, integrating both LLMs and heuristic rules, to construct the dataset for secure code review.

2.1.1 Data Collection. We adopt the CodeReviewer dataset [30] as our primary data source, as it is a large-scale dataset curated from pull requests from well-regarded GitHub projects that includes detailed code changes (*code_diff*), commit logs, review comments (*R*), covering nine programming languages. It provides a comprehensive representation of real-world code review practices, making it well-suited for training and evaluating our secure code review model. Specifically, the dataset consists of three sub-datasets corresponding to three downstream tasks: code change quality estimation, review comment generation, and code refinement. Both the code change quality estimation and review comment generation datasets are utilized to construct our dataset.

Due to a large number of code review comments (about 138K comments), it is not feasible for us to manually identify comments related to security issues. To address this, we combine keyword matching and semantic embedding matching [53] methods to capture both explicit mentions of security issues and implicit references to secure coding practices.

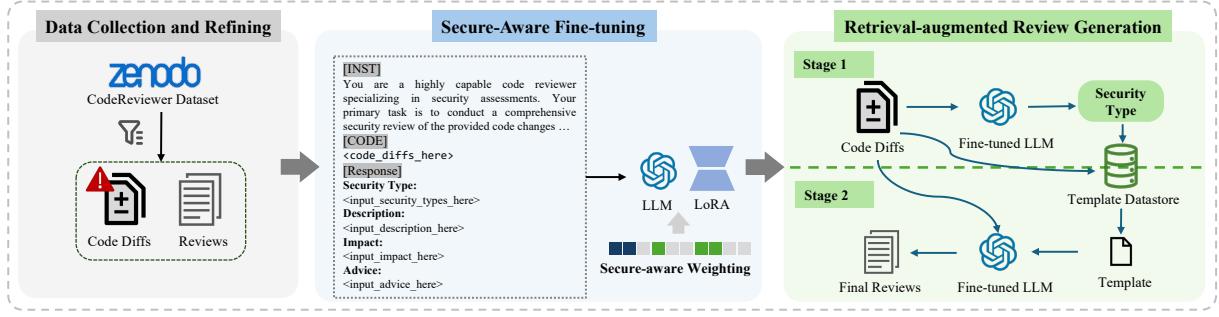


Figure 2: Overview of SECUREREVIEWER.

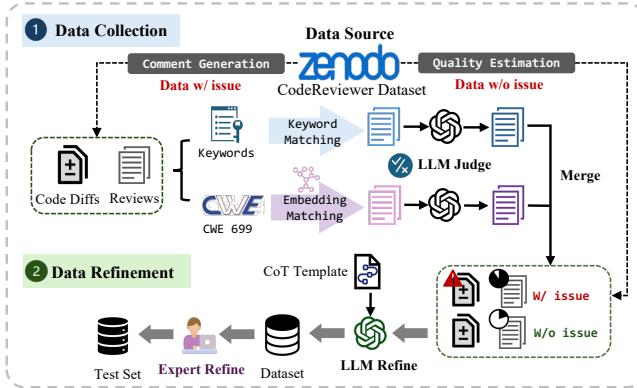


Figure 3: The process of data collection and refining.

Keyword Matching. To extract security weaknesses, we employ keyword matching using a set from Yu et al. [55]. From the original 122 keywords spanning 15 security defect types, we exclude the “common keywords” category to reduce noise. Each defect type is mapped to a Common Weakness Enumeration (CWE)¹. After text normalization (lowercasing, stemming, and punctuation removal), this initial filtering yields 10,840 candidate comments. To ensure high precision, we further refine this set using GPT-4o [1] as an LLM Judge [29, 59]. For each candidate, the LLM Judge receives the code change, the full review comment, and the matched security type. It then performs a binary classification on whether the comment accurately reflects the security issue, resulting in a final, high-quality dataset of 1,995 security-tagged review comments.

Embedding Matching. To identify security-related comments lacking explicit keywords, we employ an embedding-based matching approach. We generate vector representations for review comments and Common Weakness Enumeration (CWE) descriptions using SO_word2vec [14], a model tailored for the software engineering domain. As our semantic anchors, we leverage descriptions from CWE-699 [36], a structured vulnerability classification focused on the software development lifecycle. This classification organizes over 400 individual weaknesses into 40 major categories and notably provides the keyword groups utilized in our preceding keyword matching step. The process involves pre-processing both text sources (e.g., removing stop words and normalizing) and then computing the cosine similarity [44] between comment and CWE vectors. We retain pairs exceeding a 70% similarity threshold, which was empirically chosen over 65% and 75% to best balance match

quality and quantity. This candidate set is then filtered using the identical LLM Judge process from our keyword matching stage, ultimately yielding 2,771 security-tagged review comments.

Data Combination. We integrate the data gathered through keyword and embedding matching by removing duplicates, merging similar security types, and ensuring a balanced distribution across security types. Specifically, we begin by eliminating duplicate entries, resulting in an initial dataset of 4,089 unique data instances from 4,766. Next, we consolidate semantically similar security types to reduce redundancy. Additionally, types with low sample counts are merged with related types to improve the overall balance of the dataset [7]. Finally, we derive seven security types, as detailed in Table 1. To ensure a realistic representation of real-world code scenarios and mitigate class imbalance, we further incorporate “Non-Issue” data from the “code change quality estimation” task in CodeReviewer dataset as the 8-th type, where instances without any code review comments are considered as “Non-Issue” [30], and 585 samples are selected to maintain balance with the other types (approximately 1/8 of the whole dataset). With the inclusion of this additional category, the final dataset comprises 4,674 entries, and the distribution of category proportions is illustrated in Table 1.

2.1.2 Data Refinement. Original review comments frequently contained ambiguous phrasing or lacked critical elements essential for comprehensive security code review. To address these limitations, we perform systematic data refinement that adapts the principles of effective code review [26, 57] to the security context. Specifically, we decomposed the secure code review task into the following four sequential sub-tasks:

- **Identify the Security Type:** Clearly specify the type of security issue that is being addressed.
- **Describe the Issue:** Provide a clear and logical description of the root cause of the identified issue.
- **Explain the Impact:** Analyze the potential impact of the issue, laying the foundation for proposing a solution.
- **Advise an Improvement:** Offer actionable and specific recommendations to resolve the issue.

We argue the security code review comment should encompass the above elements, and we formally define the security code review comment R as: $R = (ST, D, I, A)$, where ST represents the Security Type, D is the issue description, I denotes the impact, and A provides actionable advice for resolving the issue.

Leveraging the advanced capabilities of LLMs in tasks such as code understanding [33], vulnerability detection [12], and bug fixing [54], we employ GPT-4o to automatically refine the collected

¹<https://cwe.mitre.org/>

Table 1: Statistics of our dataset.

Security Type	Keyword	CWE IDs	Count	Prop. (%)
Exception Handling	Crash	CWE-389, CWE-429, CWE-1228	532	11.38
Concurrency	Race Condition, Deadlock	CWE-557, CWE-387	412	8.81
Input Validation	SQL Injection, Format String, Command In-jection	CWE-1215, CWE-133, CWE-137	819	17.52
Access Control and Information Se-curity	Improper Access, Cross Site Scripting (XSS), Cross Site Request Forgery, Encryption	CWE-1211, CWE-1212, CWE-1210, CWE-255, CWE-417, CWE-310, CWE-320, CWE-1216, CWE-275, CWE-265, CWE-355, CWE-1217, CWE-199	795	17.01
Resource Management	Buffer Overflow, Use After Free, Resource Leak	CWE-1218, CWE-411, CWE-465, CWE-452, CWE-1219, CWE-399	292	6.25
State Management	Denial of Service (DoS)	CWE-1006, CWE-438, CWE-840, CWE-1226, CWE-1225, CWE-371	740	15.83
Type and Data Handling	Integer Overflow	CWE-1214, CWE-1227, CWE-569, CWE-1213, CWE-189, CWE-136, CWE-19	499	10.68
Non-Issue	-	-	585	12.52

review comment data using one-shot prompting guided by the aforementioned criteria, transforming raw review comments into structured, comprehensive review comment.

Initial Data Quality Assessment: To validate the quality of the LLM refined data, we randomly sampled 351 data entries from the 4,089 refined pieces (achieving 95% confidence level with 5% confidence interval [4]). Two domain experts, each with over 6 years of software development experience, independently evaluated these samples using aforementioned four criteria. This initial validation required 8-10 minutes per sample for code understanding and security validation, including bidirectional verification with the original comment, totaling 98 person hours. The experts achieved a Cohen's Kappa score of 0.74, indicating substantial inter-rater agreement, with disagreements resolved through discussion. This validation confirmed that 333 entries (95%) met all quality criteria, demonstrating the effectiveness of our automated refinement approach. Following this initial validation, we partition the refined dataset into training, validation, and test sets with sizes of 4,074, 300, and 300 samples, respectively, ensuring proportional representation of each security type across all subsets.

Test Set Quality Control: To establish a reliable evaluation benchmark, the same two experts conducted additional quality control specifically on the test samples (262 samples with security issues). Through meticulous examination, they identified 83 samples requiring content clarification or enhancement, which were then collaboratively refined to ensure strict adherence to our secure code review criteria. This collaborative refinement process required experts to clarify technical descriptions, enhance impact analyses, and optimize remediation advice specificity. The whole quality control process took approximately 87.3 person hours.

2.2 Secure-aware Fine-tuning

While standard end-to-end instruction-based fine-tuning for review comment generation enables LLMs to produce feedback, this approach fails to effectively identify security issues or provide context-specific actionable suggestions, often resulting in inaccurate or overly generic comments. To this end, we propose a new secure-aware fine-tuning strategy, which fine-tunes LLM to focus

on generating code review comments capable of accurately identifying security issues and providing actionable fix suggestions, leveraging our curated dataset. Specifically, we refine the training objective by modifying the loss function to prioritize two criteria: precise categorization of security issue types and heightened attention to security-critical code elements in code diffs. This approach enhances the model's capacity to produce context-sensitive, security-focused feedback, ultimately strengthening the efficacy of automated secure code reviews.

To achieve this, we introduce specific token sets that highlight security-critical elements within the code by adjusting the weighting scheme. These sets are defined as follows:

- \mathcal{I}_V : The set of tokens corresponding to identifiers in code changes referenced in review comments R receive additional weighting, as these elements are critical for pinpointing security issues (e.g., insecure function usage, improper array indexing, etc.).
- \mathcal{I}_{ST} : The set of tokens representing the specific security type (e.g., Input Validation) also receive additional weighting

Building upon this foundation, we design our secure-aware (SA) loss function $-\mathcal{L}_{SA}$ as follows:

$$-\mathcal{L}_{SA} = \sum_{t \in R} \log P(x_t | x_{<t}) + \alpha \sum_{t \in \mathcal{I}_V} \log P(x_t | x_{<t}) + \beta \sum_{t \in \mathcal{I}_{ST}} \log P(x_t | x_{<t}) \quad (1)$$

where x_i denotes the token at position i , and $P(x_i | x_{<i})$ is the probability of generating x_i based on the proceeding tokens $x_{<i}$. The first part of the equation calculates the standard cross-entropy loss for all tokens in the review comment R . The second and third terms introduce a targeted upweighting for security-critical elements, i.e., tokens in \mathcal{I}_V and \mathcal{I}_{ST} , modulated by coefficients α and β , respectively. This approach sharpens the model's focus on key security indicators.

We adopted Low-Rank Adaptation (LoRA) [22] to optimize our training process in a cost-effective manner.

2.3 Retrieval-augmented Review Generation

To further improve the quality of generated review comments and mitigate the hallucination issues commonly encountered in LLMs,

we leverage the RAG technique to incorporate specialized security domain knowledge. RAG is a widely adopted paradigm that improves LLMs by integrating relevant information retrieved from external databases into the input [17], and has been widely used in various code-related tasks [42, 52, 58]. We first construct a template datastore consisting of high-quality code review comment templates, and then retrieve the most similar comment from the datastore based on the code under review and incorporate it into the generation process.

Template datastore construction. Following established RAG practices [50] that build retrieval datastores from training data, we use our fine-tuning dataset to create templates. Given the current landscape of limited high quality secure code review data, this approach maximizes resource utilization. We manually crafted 261 high-quality code review comment templates from the training set adhering to our previously defined structure for security code review comments ($R = (ST, D, I, A)$), encompassing all security types presented in Table 1, with a distribution that closely approximates the proportional representation of each security type in the training dataset. These templates serve as a knowledge base for generating high-quality review comments.

Retrieval-Augmented Review Generation (RARG). In this process, we employ a two-stage strategy. SECUREVIEWER first generates an initial review comment based on the code change, from which we extract the corresponding security issue type. In the second stage, we utilize the BM25 algorithm to retrieve the most relevant review comment template. This retrieval process uses the code change as the query and the set of code changes linked to the predicted security issue type within the template library as the document corpus.

The retrieved template serves as an auxiliary context of the prompt to guide the generation of the final review comment, ensuring it is more accurate and normative. It is important to note that *incorporating RAG does not affect the model’s performance on issue detection since the retrieval is based on the predicted issue and does not alter the issue type within the review comment; instead, it solely updates other aspects of the comment’s content*. The prompt templates used in this process are illustrated in Figure 4.

3 Experimental Setup

3.1 Metrics

3.1.1 Issue Detection. For this task, following existing work [30, 57], we employ conventional metrics, *Precision*, *Recall*, *F1-score*, and *Accuracy*, to quantitatively assess the model’s capability to accurately identify specific security issue within the framework of an 8-category classification task (7 security types + 1 non-issue type). The security type is extracted from the generated comment (*ST*).

3.1.2 Review Comment Generation. For this task, *samples with a reference security type of “Non-Issue” are excluded from this evaluation, as no review comments are expected for these cases*, resulting in 262 samples from the test set being evaluated (38 out of the original 300 were excluded). To evaluate the quality of generated secure review comments, we use both *BLEU-4* score and a new metric we designed, *SecureBLEU*. Given that *BLEU* score struggles to fully

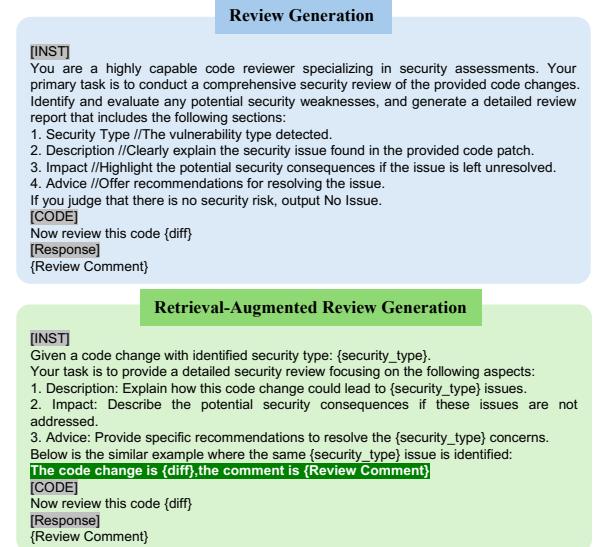


Figure 4: Prompt templates used for review generation.

assess the effectiveness of review comments in detecting and resolving security issues, we design SecureBLEU to capture both the general linguistic similarity between generated and reference texts and the critical inclusion of security-specific content.

As shown in Algorithm 1, the metric computes a score by combining two components. The first component is a modified BLEU score ($\text{score}_{\text{bleu}}$) evaluated across multiple fields of the review comment: security type, description, impact, and advice. The security type field is assessed through direct comparison—yielding a score of 100 for an exact match and 0 otherwise—while the remaining fields (description, impact, and advice) are evaluated using BLEU-4. The second component ($\text{score}_{\text{keywords}}$) evaluates the overlap of security-specific keywords (associated with the detected security type *ST*) within the description, impact, and advice. These keywords are identified using a predefined dictionary $K[ST]$, where K is a keyword dictionary organized by security type. The final score for each instance is calculated by equally weighting these two components, which was empirically validated in Section 5.2, ensuring a balanced assessment of both linguistic quality and security relevance while maintaining alignment with human judgment. $\text{Score}_{\text{bleu}}$ assesses overall linguistic similarity, treating security-critical keywords no differently than ordinary words. In contrast, $\text{score}_{\text{keywords}}$ specifically targets these security-critical terms as independent indicators of technical accuracy and domain expertise.

3.1.3 Rationale behind SecureBLEU. To justify the rationale behind SecureBLEU, we provide a detailed breakdown of how its two components work complementarily in Figure 1, where the model **incorrectly classified a security issue from “Access Control and Information Security” to “Type and Data Handling”**. Traditional BLEU-4 scores this comment at 26.44, focusing primarily on surface-level linguistic similarity. However, SecureBLEU’s two-component analysis reveals critical deficiencies: (1) $\text{score}_{\text{bleu}} = 17.57$, where the incorrect security type classification (*ST* field = 0) penalized the overall linguistic assessment. This penalty mechanism is implemented through our modified BLEU computation in

Algorithm 1, which incorporates the security type accuracy multiplier to ensure that misclassified security types receive substantially reduced scores regardless of surface-level text similarity. (2) $\text{score}_{\text{keywords}} = 6.67$, indicating that the model fails to include critical security keywords like “unauthorized access”, “misconfiguration”, and “environmental security” in its generated review comment. This omission occurred primarily due to misclassification of the security issue type, significantly impairing the comment’s ability to properly address the “Access Control and Information Security” concern. The final SecureBLEU score of 12.12 through weighted averaging provides a more justifiable quality assessment that BLEU’s surface-level matching failed to capture, demonstrating how the two components together expose both linguistic and technical inadequacies in security-focused code review.

3.2 Baselines

We select a diverse set of baselines, including both specialized code review models and general-purpose LLMs, to ensure a comprehensive comparison with our proposed method.

- **CodeReviewer** [30]: A pre-trained model specifically designed for code review. We fine-tuned the model on our dataset using the official code scripts and recommended hyperparameters.
- **LlamaReviewer** [32]: A fine-tuned LLaMA model for code review tasks. We fine-tuned the model on our dataset, maintaining the same LoRA configurations as in our experiments.
- **General LLMs**: We evaluated a wide range of general-purpose LLMs that have shown strong performance in code-related tasks. These models span diverse architectures and parameter scales, including **GPT-4o** [1], **Claude-3.5-sonnet** [9], **DeepSeek-V3** [31], **DeepSeek-R1** [18], **DeepSeek-Coder-6.7B-Instruct** [19], **CodeLlama-7B-Instruct** [41], and **Qwen2.5-Coder-7B** [23].

3.3 Implementation Details

Given their strong performance in code-related tasks and our $\sim 4K$ fine-tuning samples, we selected CodeLlama-7B [41], DeepSeek-Coder-6.7B [19], and Qwen2.5-Coder-7B [23] as our backbone models. Their 6-7B parameter size offers an optimal balance of capacity and efficiency, mitigating overfitting risks. We configured LoRA with parameters of $r = 8$, $\text{lora_alpha}=16$, and $\text{lora_dropout}=0.05$. Training was conducted with a maximum token length of 2048, a batch size of 4, gradient accumulation of 8, and a learning rate of $3e-4$. To ensure reproducible outputs, inference utilized deterministic generation with greedy decoding. For baseline reproduction, we ensured fair comparisons by adhering to original specifications. CodeReviewer [30] was fine-tuned using its official repository and recommended hyper-parameters. To isolate architectural differences, LlamaReviewer [32] was adapted using identical LoRA configurations as our method. The hyper-parameters for the baseline versions of CodeLlama-7B, DeepSeek-Coder-6.7B, and Qwen2.5-Coder-7B were also kept consistent with our model’s setup.

For the remaining baseline models—*i.e.*, GPT-4o, Claude-3.5-Sonnet, and DeepSeek-V3/R1—we used API with consistent parameters: $\text{temperature}=0.7$, $\text{top_p}=0.7$, and $\text{frequency_penalty}=0.5$ to ensure fair comparison. To account for the stochastic nature of these models with $\text{temperature}=0.7$, we conducted three independent runs for each API-based model and report the mean and

Algorithm 1 SecureBLEU

```

1: Input:  $R^P$ : predicted review,  $R^r$ : reference review,  $K$ : security-specific keywords dict,  $W$ : weight dict for fields
2: Output: SecureBLEU score
3: if  $R^P[\text{ST}]$  = “Non-Issue” then return 0
4: end if
5:  $\text{score}_{\text{bleu}} \leftarrow 0$ 
6: for each field in {ST, D, I, A} do // First Term
7:   if field = ST then
8:      $\text{score} \leftarrow 100$  if  $R^P[\text{field}] = R^r[\text{field}]$ , 0 otherwise
9:   else
10:     $\text{score} \leftarrow \text{BLEU-4}(R^P[\text{field}], R^r[\text{field}])$ 
11:   end if
12:    $\text{score}_{\text{bleu}} \leftarrow \text{score}_{\text{bleu}} + \text{score} * W[\text{field}]$ 
13: end for
14:  $\text{score}_{\text{keywords}} \leftarrow 0$ 
15: for each field in {D, I, A} do // Second Term
16:    $\text{keywords}^r \leftarrow \text{extract\_keywords}(R^r[\text{field}], K[\text{ST}])$ 
17:    $\text{keywords}^P \leftarrow \text{extract\_keywords}(R^P[\text{field}], \text{keywords}^r)$ 
18:   if  $|\text{keywords}^r| > 0$  then
19:      $\text{ratio} \leftarrow |\text{keywords}^P| / |\text{keywords}^r|$ 
20:   else
21:      $\text{ratio} \leftarrow 0$ 
22:   end if
23:    $\text{score}_{\text{keywords}} \leftarrow \text{score}_{\text{keywords}} + \text{ratio} * W[\text{field}]$ 
24: end for
25: return  $0.5 * \text{score}_{\text{bleu}} + 0.5 * \text{score}_{\text{keywords}}$ 

```

standard deviation results (in Table 2). For our SA loss function, after extensive experiments to balance the trade-off between generating fluent review comments and focusing on security-critical elements, we set the coefficients to $\alpha = 2$ and $\beta = 5$. To ensure a fair evaluation and to eliminate potential output format bias, all models were trained/prompted to generate reviews in standardized format according to our definition in Section 2.1.2.

3.4 Dataset Construction Cost

Our dataset construction relies on both GPT-4o and expert validation, incurring financial and human costs. For the LLM judge process in data collection, the LLM processed 13,611 candidate samples (10,840 from keyword matching and 2,771 from embedding matching), requiring an average of 230.43 input tokens per judgment. For data refinement procedure, LLM handled 4,089 samples, consuming an average of 692.14 input and 193.73 output tokens per sample. The total dataset construction cost was approximately \$46 based on GPT-4o pricing (\$5/1M input, \$20/1M output tokens). For the expert validation, each review took an average of 8–10 minutes per sample, totaling 98 person hours for the initial quality assessment (351 samples) and 87.3 person hours for test set refinement (262 samples, including collaborative enhancements).

4 Experimental Results and Analysis

To assess the effectiveness of SECUREVIEWER, we conduct experiments to address the following research questions:

- **RQ1: Overall Performance** - How does SECUREVIEWER perform compared to state-of-the-art code review models in terms of (1) accuracy in security issue detection, and (2) overall quality of generated review comments?
- **RQ2: Ablation Study** - What is the contribution of each component in SECUREVIEWER to its overall performance?
- **RQ3: Quality Analysis** - How effectively does SECUREVIEWER address different types of security issues?

Table 2: Results on issue detection and review comment generation. For general LLMs, we conducted three independent runs and report the mean and standard deviation of the results.

Model	Issue Detection				Comment Generation	
	Precision	Recall	F1	Accuracy	BLEU	SecureBLEU
CodeReviewer	65.88	57.44	59.03	58.53	8.66	21.31
LlamaReviewer	<u>66.06</u>	<u>60.29</u>	<u>61.46</u>	<u>61.20</u>	9.20	<u>24.56</u>
DeepSeek-R1	54.73 (± 1.77)	46.54 (± 1.45)	46.24 (± 1.31)	46.27 (± 2.05)	5.81 (± 0.38)	15.84 (± 1.27)
DeepSeek-V3	62.83 (± 0.29)	51.89 (± 0.38)	53.31 (± 0.43)	53.56 (± 0.51)	<u>10.80</u> (± 0.34)	21.84 (± 0.92)
DeepSeek-Coder-6.7B	36.30	18.55	15.58	23.23	6.85	16.00
CodeLlama-7B	14.69	12.35	6.22	17.39	4.26	11.68
Qwen2.5-Coder-7B	46.31	39.61	38.57	45.00	7.04	20.63
GPT-4o	58.74 (± 0.52)	52.66 (± 0.43)	53.18 (± 0.35)	54.50 (± 0.44)	7.60 (± 0.26)	19.33 (± 0.52)
Claude-3.5-sonnet	60.74 (± 0.46)	53.56 (± 0.51)	52.81 (± 0.65)	54.27 (± 0.51)	8.83 (± 0.24)	19.54 (± 0.83)
SECUREVIEWER _{CL}	73.28	71.48	71.98	71.91	11.34	29.31
SECUREVIEWER _{DS}	72.25	71.23	71.62	72.24	11.01	29.23
SECUREVIEWER _{QW}	73.56	70.64	71.60	71.33	9.35	28.76

4.1 RQ1: Overall Performance

4.1.1 RQ1-1: Performance of Issue Detection. The left section of Table 2 presents the performance comparison on the issue detection task. Among all the baselines, LlamaReviewer and CodeReviewer, both fine-tuned on our constructed dataset, demonstrate superior performance compared to general-purpose LLMs. This highlights the effectiveness and importance of fine-tuning on domain-specific and high-quality datasets, enabling these models to outperform their general-purpose counterparts by leveraging domain-specific knowledge. SECUREVIEWER, implemented in three variants (SECUREVIEWER_{CL} based on CodeLlama, SECUREVIEWER_{DS} based on DeepSeek-Coder, and SECUREVIEWER_{QW} based on Qwen2.5-Coder), consistently outperforms all baseline models across all evaluation metrics. This highlights the efficacy of our secure-aware fine-tuning strategy, which enhances the model’s sensitivity to security-related classification tokens, thus achieving better results in identifying security issues. It is also worth noting that while CodeLlama initially struggles to detect security issues, its fine-tuned version, SECUREVIEWER_{CL}, achieves significant performance improvements. This further underscores the effectiveness of our fine-tuning strategy in enhancing the model’s capabilities. Moreover, we observe that general LLMs such as Claude, DeepSeek-V3, Qwen2.5-Coder, and GPT-4o, while not fine-tuned, still achieve considerable performance, demonstrating their promising potential and performance in detecting security issues even without task-specific optimization. The consistent results across multiple runs further validate the reliability of these comparisons.

4.1.2 RQ1-2: Performance of Review Comment Generation. For the evaluation of review comment generation, we utilize both the BLEU-4 score and the SecureBLEU metric.

While BLEU-4 measures general linguistic similarity, SecureBLEU provides a more nuanced assessment by focusing on security-specific content. The results are shown in the right portion of Table 2. Regarding BLEU-4 score, SECUREVIEWER_{CL} achieves a score of 11.34, outperforming the best-performing baseline (DeepSeek-V3) by 5%. Among all the evaluated models, BLEU-4 scores show relatively modest variation, with most baselines falling between

7 and 10. However, DeepSeek-R1 and CodeLlama diverge significantly from this range, scoring 5.81 and 4.26, respectively. After analyzing the results, we observe that the lower performance of DeepSeek-R1 is primarily due to its overly divergent and unstructured reasoning processes during code analysis. Specifically, the model tends to engage in excessive and repetitive thinking patterns, frequently shifting between analytical approaches without fully developing any single line of reasoning. This leads to shallow analyses that overlook critical issues while emphasizing irrelevant aspects of the code. As for CodeLlama, it often fails to adhere to instruction guidelines, frequently repeating input code verbatim rather than providing meaningful feedback.

Unlike BLEU-4, the SecureBLEU metric, which measures the effectiveness of the review comment in detecting and resolving the security issues, reveals more pronounced differences across models. This metric effectively captures variations in the quality of security-focused content within the generated comments, providing a more nuanced assessment of their relevance and utility in addressing security concerns.

Among the baseline models, LlamaReviewer achieves the highest SecureBLEU score of 24.56, aligning with its strong performance in issue detection. This correlation indicates that the quality of generated review comments is closely tied to the model’s ability to detect security issues. In other words, if a model can accurately identify security vulnerabilities, it is more likely to produce clear issue descriptions, thorough impact analyses, and actionable remediation recommendations. Among general-purpose LLMs, DeepSeek-V3 and Claude achieve promising performance, with SecureBLEU scores of 21.84 and 19.54, respectively, even exceeding that of the fine-tuned CodeReviewer (21.31), demonstrating the adaptability of these LLMs to security-related tasks despite their lack of domain-specific fine-tuning. Nevertheless, SECUREVIEWER outperforms all baselines substantially, achieving a 19% relative improvement over the best baseline, underscoring the effectiveness of our fine-tuning strategy combined with retrieval-augmented generation, which improves both the linguistic quality and security relevance of the generated comments.

Answer to RQ1: SECUREREVIEWER achieves state-of-the-art performance in secure code review. For issue detection, it achieves 17% higher F1 score and 18% better accuracy than the best-performing baseline. Regarding the quality of generated review comments, it exceeds the best baseline of 11% in BLEU-4 and demonstrates approximately 19% improvement in SecureBLEU.

4.2 RQ2: Ablation Study

We conduct an ablation study to evaluate each component’s contribution in SECUREREVIEWER, including: (1) domain-specific fine-tuning to establish baseline security expertise, (2) security-aware loss optimization to enhance focus on critical security elements, and (3) retrieval-augmented generation to ground reviews in established security best practices. We incrementally incorporate these components using CodeLlama-7B, DeepSeek-Coder-6.7B, and Qwen2.5-Coder-7B as backbone models. The results are presented in Table 3.

Table 3: Ablation study results of SECUREREVIEWER.

Model	Precision	Issue Detection			Comment Generation	
		Recall	F1	Accuracy	BLEU	SecureBLEU
DeepSeek-Coder-6.7B	36.30	18.55	15.58	23.23	6.85	16.00
+ Fine-tuning	70.70	68.76	68.90	68.23	11.08	26.27
+ SA-Loss	72.25	71.23	71.62	72.24	11.27	28.79
+ RARG (our model)	72.25	71.23	71.62	72.24	11.01	29.23
CodeLlama-7B	14.69	12.35	6.22	17.39	4.26	11.68
+ Fine-tuning	71.64	69.95	71.09	70.23	11.91	27.88
+ SA-Loss	73.28	71.48	71.98	71.91	12.46	29.69
+ RARG (our model)	73.28	71.48	71.98	71.91	11.34	29.31
Qwen2.5-Coder-7B	46.31	39.61	38.57	45.00	7.04	20.63
+ Fine-tuning	71.12	68.42	68.84	68.67	9.41	27.61
+ SA-Loss	73.56	70.64	71.60	71.33	9.47	29.21
+ RARG (our model)	73.56	70.64	71.60	71.33	9.35	28.76

4.2.1 Impact of Fine-tuning. Domain-specific fine-tuning represents a fundamental adaptation strategy for LLMs to specialize in security-focused code review. We evaluate its contribution to enhancing our model’s performance.

As seen from the results, the performance of vanilla LLMs reveals limited capability in secure code review. This is particularly pronounced in CodeLlama-7B, where poor instruction-following behavior significantly impairs its effectiveness. It frequently generates irrelevant identifiers, repetitive code snippets, and meaningless outputs, which may due to its insufficient exposure to security review tasks and code-diff patterns during pre-training.

Applying domain-specific fine-tuning yields substantial improvements, aligning the models with intricate code patterns and security vulnerabilities. For instance, fine-tuning CodeLlama-7B achieves an absolute improvement of 64.87 F1-score in issue detection, while Qwen2.5-Coder-7B improves by 30.27 F1-score, enhancing review comment quality with BLEU-4 increasing by 7.65 and SecureBLEU improving by 16.2 for CodeLlama-7B. These gains highlight fine-tuning’s critical role in adapting general-purpose LLMs to the nuanced requirements of secure code review.

4.2.2 Impact of SA-Loss. After employing our proposed secure-aware loss optimization, which enhances the model’s focus on security-critical tokens through a re-weighted loss function, the performance of both issue detection and review comment generation is further improved. Although the overall performance improvements

are less pronounced compared to those achieved through domain-specific fine-tuning, this approach sharpens the model’s sensitivity to security-related features, thus striking a better balance between precision and recall in issue detection and further enhancing the overall quality of the generated review comments.

4.2.3 Impact of RARG. As mentioned in Section 2.3, applying the RARG does not affect model’s issue detection performance. As a result, the results of issue detection remain consistent with those achieved through fine-tuning and SA loss optimization. Regarding the review comment generation, we observe that applying the RARG does not yield consistent or significant improvements.

This primarily stems from the following two aspects. First, domain-specific capabilities instilled during fine-tuning render retrieved templates largely redundant with the model’s internal knowledge base. The fine-tuned model already encodes specialized patterns for security issue detection and resolution, reducing the added value of external templates. Second, fine-tuning may diminish the model’s general instruction-following capacity, constraining its ability to leverage RAG’s external context effectively. This is compounded by an inconsistency in instruction formats between training and inference phases. During fine-tuning, the model learns to generate comments without example-based instructions, whereas during RAG inference, a retrieved template is injected into the input prompt. This structural mismatch disrupts the model’s ability to generalize under the altered input format, leading to suboptimal adaptation and diminishing returns.

Building on the aforementioned analysis, we argue that our RARG framework may prove particularly advantageous for general-purpose LLMs. To validate this hypothesis, we apply RARG to GPT-4o, Claude-3.5-Sonnet, and DeepSeek-V3, with results summarized in Table 4. As demonstrated in the results, RARG brings substantially improvements in SecureBLEU scores for these models. This notable improvement stems from a key distinction: general-purpose LLMs are inherently trained on broad, diverse datasets without task-specific specialization, rendering them deficient in domain-specific security knowledge compared to fine-tuned counterparts. The RARG framework effectively bridges this critical gap by retrieving and integrating security-relevant contextual patterns that these models would otherwise fail to prioritize. This supplementation enables them to produce more security-relevant and actionable review comments.

Table 4: Performance of RARG on general LLMs.

Model	BLEU	SecureBLEU
GPT-4o	7.60	19.33
+ RARG	7.47	23.93
Claude-3.5-sonnet	8.83	19.54
+ RARG	7.75	29.34
DeepSeek-V3	10.80	21.84
+ RARG	10.19	25.64

Answer to RQ2: Each component contributes to SECUREREVIEWER’s performance gains, with domain-specific fine-tuning delivering

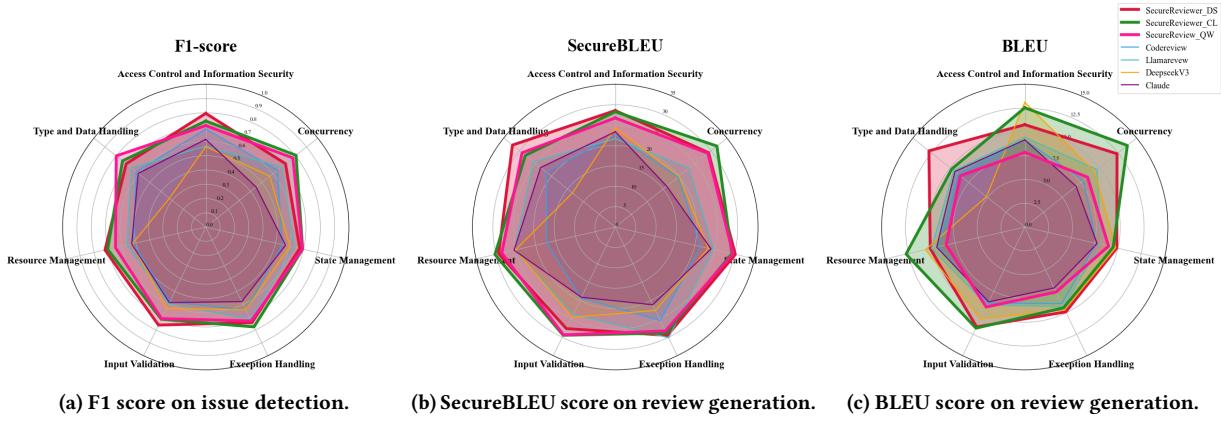


Figure 5: Performance across various security types.

the most substantial improvements. While RARG provides limited benefits for fine-tuned models, it substantially enhances general-purpose LLMs by augmenting their security knowledge.

4.3 RQ3: Quality Analysis

We analyze SECUREVIEWER’s performance in issue detection and review generation across the seven security types detailed in Table 1. We compare our model against four top-performing baselines—CodeReviewer, LlamaReviewer, DeepSeek-V3, and Claude-3.5-sonnet—with the results presented in Figures 5.

4.3.1 Issue Detection. Figure 5a illustrates the issue detection performance across various security types. We can observe that all three variants of SECUREVIEWER achieve balanced performance in issue detection, outperforming baseline models across multiple categories. The results reveal that baseline performance degrades with higher vulnerability complexity. Specifically, *Concurrency* issues—which require complex semantic reasoning about thread synchronization—show large performance gaps between baselines and SECUREVIEWER. CodeReviewer and LlamaReviewer, fine-tuned with domain-specific data, display more balanced performance across all types. These findings underscore the importance of domain-specific fine-tuning for achieving robust and generalizable performance on security-focused tasks. Despite SECUREVIEWER’s substantial improvements over baselines, its performance varies across different security types. The approach is less effective on *State Management* and *Resource Management* issues. Notably, while demonstrating significant gains for *Concurrency* issues, they remain a particularly challenging category across all model variants. This variation is attributable to the fundamental differences in how these distinct issue types manifest.

Specifically, *Concurrency*, *State Management*, and *Resource Management* issues require deeper semantic reasoning about thread synchronization, state transitions, and resource lifecycles that extend beyond isolated code diff contexts. These limitations highlight the tension between pattern recognition and comprehensive semantic reasoning in our approach, as further analyzed in Section 5.1.

Figures 5b and 5c present the review comment performance across different security types, evaluated using SecureBLEU and BLEU metrics, respectively. For SecureBLEU, all variants of SECUREVIEWER deliver balanced and superior performance across

all categories, substantially outperforming the baselines. Notably, the score distribution of all models aligns closely with the F1-score trends observed in issue detection (Figure 5a). This consistency highlights a strong correlation between issue detection performance and the quality of generated review comments, as captured by SecureBLEU.

Conversely, categories with lower F1 performance in issue detection like *State Management* and *Resource Management* show correspondingly modest SecureBLEU scores. On one hand, lower F1 scores indicate fewer successful predictions, resulting in reduced overlap of category-specific security keywords and consequently lower weighted SecureBLEU scores. On the other hand, human-crafted reference comments for *State Management* and *Resource Management* issues extensively incorporate contextual code identifiers and causal explanations (e.g., “variable X not released leads to resource leak”), while our model still struggles to capture these contextual references.

Regarding BLEU scores, as shown in Figure 5c, SECUREVIEWER achieves comparable or slightly higher scores than baselines, though performance improvements are less pronounced compared to SecureBLEU and F1 gains. This discrepancy primarily arises from our proposed secure-aware fine-tuning strategy, which is specifically optimized to prioritize security-critical tokens over general linguistic fluency. By focusing on accurately detecting security issues and providing precise explanations and advice, SECUREVIEWER generates comments that, while highly relevant to security, may differ from reference review comments in phrasing or structure, resulting in relatively lower BLEU scores despite enhanced practical utility for security review purposes.

Answer to RQ3: SECUREVIEWER shows balanced and superior performance across various security types in identifying and addressing security issues. An obvious correlation exists between the issue detection accuracy and the quality of the generated review comments. However, issues requiring deep semantic understanding remain challenging due to limited context incorporation and the inherent capabilities of LLMs.

5 Discussion

5.1 Human Evaluation

Since automatic metrics do not always agree with the practical utility of the review, we conduct human evaluation to further assess the quality of review comments generated by SECUREVIEWER.

Procedure. We recruit two software engineers in the evaluation, each with over 6 years of experience in Java and Python development, code review practices, and expertise in CWEs. The core security concepts in CWE—such as injection attacks, access control flaws, and cryptographic issues—share common security principles across different languages, enabling our evaluators to assess review comments based on security concepts and impact analysis rather than language-specific syntax details. These experts independently evaluated the 262 generated comments from the test set (cases labeled “Non-Issue” were excluded). For each data point, evaluators were presented with the code diff, its corresponding reference comment, and the generated comment. Each generated comment was rated against four criteria drawn from academic research on code review effectiveness [8] and industry standards for security-focused reviews [35, 38]: ① **Clarity:** Whether the review comment clearly explain the root cause of the issue, and references specific code snippets or patterns. ② **Relevance:** Whether the review comment relevant to the code contexts and issues, and avoid irrelevant or overly generic content. ③ **Comprehensiveness:** Whether the impact analysis thoroughly explain potential consequences. ④ **Actionability:** Whether the improvement advice specific, feasible, and aligned with best practices. All ratings are integers on a scale of 1 to 5, with higher scores indicating better performance.

Results. Figure 6 presents the results of the human evaluation. Each score represents the average rating from two evaluators for the 262 test samples. A Cohen’s Kappa coefficient [10] of 0.66 confirms a substantial agreement between the raters. The generated comments received consistently high ratings across all four criteria, with average scores of 3.93 for Clarity, 4.06 for Relevance, 3.98 for Comprehensiveness, and 3.90 for Actionability. These scores indicate that the reviews generated by SECUREVIEWER demonstrate strong practical utility, effectively combining the proficiency to identify security issues with the ability to provide actionable guidance. This highlights the model’s effectiveness in supporting real-world security review workflows.

Correlation with SecureBLEU&BLEU. We further calculate the Pearson’s correlation between the human evaluation score with the two metrics used in our evaluation (BLEU and SecureBLEU). The values are $r = 0.7533$ and $r = 0.4026$ for SecureBLEU and BLEU, respectively, which validate the strong alignment of SecureBLEU with human judgment.

Figure 7 shows the distribution of human evaluation scores alongside the corresponding BLEU and SecureBLEU scores. Notably, SecureBLEU exhibits a stronger correlation with human judgment compared to BLEU. The BLEU plot shows numerous points in the top-left region, where comments received low BLEU scores but high human ratings, indicating BLEU undervalues comments that human experts consider high quality. In contrast, the SecureBLEU plot shows a more desirable distribution with fewer inconsistently evaluated points, broader score range, and stronger clustering of high human ratings with high SecureBLEU scores. These findings

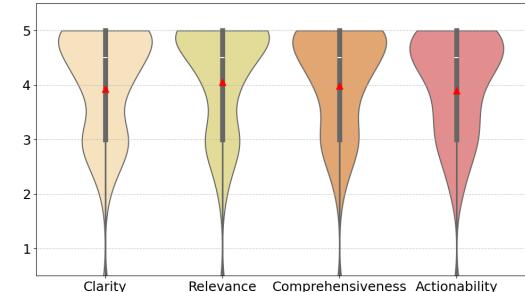


Figure 6: Human evaluation results of SECUREVIEWERDS.

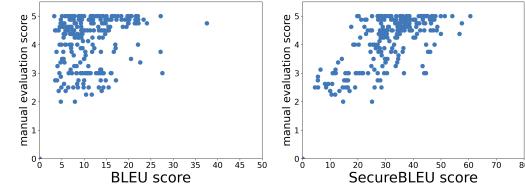


Figure 7: Correlation between human evaluation scores and SecureBLEU&BLEU scores.

further confirm SecureBLEU’s superior alignment with human preferences, establishing it as a more reliable metric for the automated evaluation of secure code review.

Error Analysis. To gain deeper insight into the limitations of SECUREVIEWER, we perform a thorough error analysis of review comments that received low human evaluation scores and identify the following two error patterns. ① **Superficial Pattern Matching:** SECUREVIEWER occasionally prioritizes surface-level pattern recognition—such as security-related keywords (e.g., map) or syntactic structures (e.g., mutex operations)—over in-depth semantic reasoning. For instance, when analyzing code with map operations, our model may focus on superficial indicators like the presence of map keywords or deletion operations. Thus, it erroneously flags concurrency issues (e.g., suggesting mutex protection for maps) but fails to diagnose the underlying root cause of the actual vulnerability, such as missing input validation or array bounds checking.

② **Limited Contextual Awareness:** In some cases, our model struggles to accurately interpret code semantics due to its reliance on isolated code diff, which lack the broader context of the full codebase, execution flows, and inter-procedural dependencies, resulting in failures on identifying possible issues. For example, when reviewing array operations, SECUREVIEWER may fail to detect out-of-bounds vulnerabilities because it cannot infer how the array is initialized or modified in other parts of the codebase. These findings highlight the need for enhanced semantic understanding and broader contextual integration in automated code review, which will be the focus of our future work to further improve the effectiveness of our model.

5.2 Impact of Weight Setting for SecureBLEU

When computing SecureBLEU, we employ an equal weighting scheme for $\text{score}_{\text{bleu}}$ and $\text{score}_{\text{keywords}}$ to balance linguistic quality and security relevance. To validate this choice, we empirically compared different weighting schemes by measuring Pearson correlation between human evaluation scores and SecureBLEU (following Section 5.1). We systematically evaluated weight ratios for $(\text{score}_{\text{bleu}}, \text{score}_{\text{keywords}})$ ranging from 0.2/0.8 to 0.8/0.2. The results

demonstrate that the 0.5/0.5 setting achieves the highest correlation coefficient ($r = 0.7533$) with human preferences, outperforming all alternative configurations. This confirms that an equal weighting aligns best with human judgment, thereby justifying our choice for computing SecureBLEU.

5.3 Threats to Validity

Threats to internal validity relate to the hyper-parameters setting during the fine-tuning. For API-based models, we used temperature=0.7 and conducted three runs to ensure statistical reliability. We conduct a small-range grid search on learning rate, batch size, LoRA parameters, the coefficients α and β in SA loss, and the final setting was selected based on the best performance observed on the validation set. It is expected that more hyper-parameter tuning would bring more improvements. Our study was also constrained by several factors. The limited size of our dataset restricted training to 7B backbone LLMs, and it is possible that newer models not included in our evaluation may offer superior performance. Furthermore, our use of the fine-tuning dataset to construct RAG templates, a decision necessitated by the scarcity of high-quality secure code review data, may limit the technique's effectiveness. Future work will explore additional LLMs and alternative datastores for RAG template construction.

Threats to external validity arise from potential errors in the LLM-based data refinement. We mitigated this by confirming that 95% of the data met our quality standards via manual sampling (Section 2.1.2) and by having two domain experts manually review and refine the entire test set. Moreover, our dataset construction relies on both LLM (GPT-4o) and expert annotation, incurring financial and human costs. Future work could leverage emerging cost-effective models like DeepSeek-V3 to reduce this expense while maintaining quality. While manual validation of the test set remains essential for ensuring reliable evaluation, this process can be optimized. Strategic automation, such as using CodeQL for initial security checks and leveraging cost-efficient LLMs for preliminary assessments and formatting, could further streamline this workflow. **Threats to construct validity** relate to the rationality of evaluation metrics. Following existing code review research [30, 32, 57], we employ Precision, Recall, F1, and Accuracy to assess the model's capability to correctly identify security issues, and use BLEU-4 score and our proposed SecureBLEU to evaluate the quality of generated secure review comments. We further conduct human evaluation studies to assess the practical utility and quality of review comments generated by SECUREVIEWER.

6 Related Work

6.1 Code Review Automation

Code review is a key practice in software development that involves a systematic review of the source code to find defects as well as improve quality. Recent advances in deep learning and LLMs have enabled significant progress in automating code review. Tufano et al. [46] fine-tune T5 [40] model for generating review comments. Li et al. [30] introduce the CodeReviewer, a transformer-based model pre-trained with four pre-training tasks for code review. Recent advances using LLMs have shown promising results in both accuracy and interpretability. Lu et al. [32] propose the LLaMA-Reviewer

using parametric efficient fine-tuning techniques to fine-tune the LLaMA model. Yu et al. [57] fine-tune open-source LLMs with chain-of-thought-guided data to generate review comment that not only pinpoint code issues in detail but also provide logical explanations and actionable repair suggestions. However, these general-purpose code review approaches often produce inaccurate or irrelevant comments, are impacted by dataset noise [30, 45], and lack security specialization. Moreover, current widely-adopted evaluation metrics, such as BLEU-4 score, also fail to address security-specific needs, underscoring the need for tailored automated techniques and evaluation frameworks focused on security issue detection.

6.2 Code Review for Security Issues

Existing research on security-related code review predominantly focus on empirical studies, which investigate the role of code review in finding and mitigating security issues [5, 55, 56].

Charoenwet [5] find that conventional reviews struggle with language-specific security issues, such as C++ memory management [27] or cross-site scripting attacks [21]. Similarly, Yu et al. [55] analyze 430,000 comments from open-source communities and reveal that security defects accounted for less than 1% of discussions, with race conditions and resource leaks dominating the conversation. Charoenwet et al. [6] show that static analysis tools can detect certain vulnerabilities but falter when faced with context-dependent issues. Developers also resist these tools due to usability and integration difficulties, as noted by Johnson et al. [25]. More recently, Yu et al. [56] demonstrate that LLMs outperform static analysis tools in detecting security defects but still face limitations in accuracy and contextual understanding.

While these studies identify challenges and limitations in practice and provide valuable insights for improving secure code reviews, they fall short of providing automated solutions to systematically address these issues. These studies directly inspire our work in several key ways. For example, the security defect categories identified by Yu et al. [55] guide our selection of security-relevant keywords for the SecureBLEU metric and serve as filtering criteria during dataset construction.

7 Conclusion

In this paper, we propose SECUREVIEWER, a framework designed for secure code review. We begin by constructing a dataset for training and evaluating the model's secure code review capabilities. Building on this foundation, we introduce a security-aware fine-tuning strategy to enhance the LLM's ability to generate code review comments that effectively identify security issues and provide actionable fix recommendations. Additionally, we integrate the RAG technique to mitigate LLM hallucinations and improve the relevance and reliability of generated comments. Experimental results demonstrate that SECUREVIEWER outperforms state-of-the-art models, validating its effectiveness and practical applicability.

Acknowledgments

This research is supported by the National Natural Science Foundation of China Grants Nos. 62302021, 62332001, and the Fundamental Research Funds for the Central Universities (Grant No. JK2024-28).

References

- [1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).
- [2] Enna Basic and Alberto Giaretta. 2024. Large Language Models and Code Security: A Systematic Literature Review. *arXiv preprint arXiv:2412.15004* (2024).
- [3] Gabriela Bavota and Barbara Russo. 2015. Four eyes are better than two: On the impact of code reviews on software quality. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 81–90.
- [4] CJ Bulpitt. 1987. Confidence intervals. *The Lancet* 329, 8531 (1987), 494–497.
- [5] Wachiraphan Charoenwet. 2023. Complementing Secure Code Review with Automated Program Analysis. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 189–191.
- [6] Wachiraphan Charoenwet, Patanamon Thongtanunam, Van-Thuan Pham, and Christoph Treude. 2024. An empirical study of static analysis tools for secure code review. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 691–703.
- [7] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.
- [8] Junkai Chen, Zhenhai Li, Qiheng Mao, Xing Hu, Kui Liu, and Xin Xia. 2025. Understanding Practitioners' Expectations on Clear Code Review Comments. *Proceedings of the ACM on Software Engineering* 2, ISSTA (2025), 1257–1279.
- [9] claude. 2023. Claude. <https://claude.ai/>
- [10] Jacob Cohen. 1968. Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. *Psychological bulletin* 70, 4 (1968), 213.
- [11] Abdallah Dawoud, Soeren Finster, Nicolas Coppik, and Virendra Ashiwal. 2024. Better Left Shift Security! Framework for Secure Software Development. In *2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 642–649.
- [12] Xueying Du, Geng Zheng, Kaixin Wang, Jiayi Feng, Wentai Deng, Mingwei Liu, Bihuan Chen, Xin Peng, Tao Ma, and Yiling Lou. 2024. Vul-rag: Enhancing llm-based vulnerability detection via knowledge-level rag. *arXiv preprint arXiv:2406.11147* (2024).
- [13] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, Michael Bailey, et al. 2014. The matter of heartbleed. In *Proceedings of the 2014 conference on internet measurement conference*. 475–488.
- [14] Vasiliki Efstathiou, Christos Chatzilena, and Diomidis Spinellis. 2018. Word embeddings for the software engineering domain. In *Proceedings of the 15th international conference on mining software repositories*. 38–41.
- [15] Michael Fagan. 2002. A history of software inspections. *Software pioneers: contributions to software engineering* (2002), 562–573.
- [16] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020).
- [17] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* (2023).
- [18] Daya Guo, Dejian Yang, Haowei Zhang, Junxia Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948* (2025).
- [19] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. 2024. DeepSeek-Coder: When the Large Language Model Meets Programming—The Rise of Code Intelligence. *arXiv preprint arXiv:2401.14196* (2024).
- [20] Anshul Gupta and Neel Sundaresan. 2018. Intelligent code reviews using deep learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'18) Deep Learning Day*.
- [21] Abdelhakim Hannousse, Salima Yahiouche, and Mohamed Cherif Nait-Hamoud. 2024. Twenty-two years since revealing cross-site scripting attacks: A systematic mapping and a comprehensive survey. *Computer Science Review* 52 (2024), 100634.
- [22] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685* (2021).
- [23] Bin yuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. 2024. Qwen2. 5-Coder Technical Report. *arXiv preprint arXiv:2409.12186* (2024).
- [24] Emmanuel Ichu and Rao Nemani. 2011. The role of quality assurance in software development projects: Project failures and business performance. *Int. J. Comp. Tech. Appl.* 2, 4 (2011), 716–725.
- [25] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. 2013. Why don't software developers use static analysis tools to find bugs?. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 672–681.
- [26] Oleksii Kononenko, Olga Baysal, and Michael W Godfrey. 2016. Code review quality: How developers see it. In *Proceedings of the 38th international conference on software engineering*. 1028–1038.
- [27] Woo Hyong Lee and Morris Chang. 2002. A study of dynamic memory management in C++ programs. *Computer Languages, Systems & Structures* 28, 3 (2002), 237–272.
- [28] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems* 33 (2020), 9459–9474.
- [29] Bowen Li, Wenhan Wu, Ziwei Tang, Lin Shi, John Yang, Jinyang Li, Shunyu Yao, Chen Qian, Bin yuan Hui, Qicheng Zhang, Zhiyin Yu, He Du, Ping Yang, Dahu Lin, Chao Peng, and Kai Chen. 2024. DevBench: A Comprehensive Benchmark for Software Development. *CoRR* abs/2403.08604 (2024).
- [30] Zhiyu Li, Shuai Lu, Daya Guo, Nan Duan, Shailesh Janmu, Grant Jenks, Deep Majumder, Jared Green, Alexey Svyatkovskiy, Shengyu Fu, et al. 2022. 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*. 1035–1047.
- [31] Aixin Liu, Bei Feng, Bing Xue, Binxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437* (2024).
- [32] 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 *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 647–658.
- [33] Yingwei Ma, Qingping Yang, Rongyu Cao, Binhua Li, Fei Huang, and Yongbin Li. 2024. How to understand whole software repository? *arXiv preprint arXiv:2406.01422* (2024).
- [34] Gary McGraw. 2004. Software security. *IEEE Security & Privacy* 2, 2 (2004), 80–83.
- [35] Metrudev. 2023. Code Review Guidelines: Best Strategies. <https://www.metrudev.com/metrics/code-review-guidelines-best-strategies/>.
- [36] MITRE. n.d. CWE VIEW: Software Development (View ID: 699). <https://cwe.mitre.org/data/definitions/699.html>
- [37] MITRE Corporation. 2014. CVE-2014-0160 Detail. <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160>. [Accessed: 2023-10-01].
- [38] OWASP Foundation. 2023. OWASP Risk Rating Methodology. https://owasp.org/www-community/OWASP_Risk_Rating_Methodology.
- [39] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 311–318.
- [40] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research* 21, 140 (2020), 1–67.
- [41] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaojing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950* (2023).
- [42] Ensheng Shi, Yanlin Wang, Wei Tao, Lun Du, Hongyu Zhang, Shi Han, Dongmei Zhang, and Hongbin Sun. 2022. RACE: Retrieval-augmented commit message generation. *arXiv preprint arXiv:2203.02700* (2022).
- [43] Shu-Ting Shi, Ming Li, David Lo, Ferdinand Thung, and Xuan Huo. 2019. Automatic code review by learning the revision of source code. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4910–4917.
- [44] Amit Singhal et al. 2001. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.* 24, 4 (2001), 35–43.
- [45] Rosalia Tufano, Ozren Dabić, Antonio Mastropaoletti, Matteo Ciniselli, and Gabriele Bavota. 2024. Code review automation: strengths and weaknesses of the state of the art. *IEEE Transactions on Software Engineering* (2024).
- [46] Rosalia Tufano, Simone Masiero, Antonio Mastropaoletti, Luca Pasquarella, Denys Poshyvanyk, and Gabriele Bavota. 2022. Using pre-trained models to boost code review automation. In *Proceedings of the 44th international conference on software engineering*. 2291–2302.
- [47] Rosalia Tufano, Luca Pasquarella, Michele Tufano, Denys Poshyvanyk, and Gabriele Bavota. 2021. Towards Automating Code Review Activities. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 163–174.
- [48] A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* (2017).
- [49] Jiejin Wang, Xitong Luo, Liwen Cao, Hongkui He, Hailin Huang, Jiayuan Xie, Adam Jatowt, and Yi Cai. 2024. Is your ai-generated code really safe? evaluating large language models on secure code generation with codesceval. *arXiv preprint arXiv:2407.02395* (2024).
- [50] Shuohang Wang, Yichong Xu, Yuwei Fang, Yang Liu, Siqi Sun, Ruochen Xu, Chenguang Zhu, and Michael Zeng. 2022. Training Data is More Valuable than You Think: A Simple and Effective Method by Retrieving from Training Data.

- In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 3170–3179.
- [51] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859* (2021).
 - [52] Di Wu, Wasi Uddin Ahmad, Dejiao Zhang, Murali Krishna Ramanathan, and Xiaofei Ma. 2024. Repoformer: Selective retrieval for repository-level code completion. *arXiv preprint arXiv:2403.10059* (2024).
 - [53] Lingfei Wu, Ian EH Yen, Kun Xu, Fangli Xu, Avinash Balakrishnan, Pin-Yu Chen, Pradeep Ravikumar, and Michael J Witbrock. 2018. Word mover’s embedding: From word2vec to document embedding. *arXiv preprint arXiv:1811.01713* (2018).
 - [54] Chunqiu Steven Xia and Lingming Zhang. 2024. Automated program repair via conversation: Fixing 162 out of 337 bugs for \$0.42 each using ChatGPT. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 819–831.
 - [55] Jiaxin Yu, Liming Fu, Peng Liang, Amjad Tahir, and Mojtaba Shahin. 2023. Security Defect Detection via Code Review: A Study of the OpenStack and Qt Communities. In *2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 1–12.
 - [56] Jiaxin Yu, Peng Liang, Yujia Fu, Amjad Tahir, Mojtaba Shahin, Chong Wang, and Yangxiao Cai. 2024. An Insight into Security Code Review with LLMs: Capabilities, Obstacles and Influential Factors. *arXiv preprint arXiv:2401.16310* (2024).
 - [57] Yongda Yu, Guoping Rong, Haifeng Shen, He Zhang, Dong Shao, Min Wang, Zhao Wei, Yong Xu, and Juhong Wang. 2024. Fine-tuning large language models to improve accuracy and comprehensibility of automated code review. *ACM transactions on software engineering and methodology* 34, 1 (2024), 1–26.
 - [58] Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023. Repocoder: Repository-level code completion through iterative retrieval and generation. *arXiv preprint arXiv:2303.12570* (2023).
 - [59] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems* 36 (2023), 46595–46623.