LAURA: Enhancing Code Review Generation with Context-Enriched Retrieval-Augmented LLM

Yuxin Zhang
School of Computer Science and Technology
Beijing Institute of Technology
Beijing, China
yuxinzhang@bit.edu.cn

Yuxia Zhang†
School of Computer Science and Technology
Beijing Institute of Technology
Beijing, China
yuxiazh@bit.edu.cn

Zeyu Sun
Institute of Software
Chinese Academy of Sciences
Beijing, China
zeyu.zys@gmail.com

Yanjie Jiang
School of Computer Science
Tianjin University
Tianjin, China
2990094974@qq.com

Hui Liu
School of Computer Science and Technology
Beijing Institute of Technology
Beijing, China
Liuhui08@bit.edu.cn

***Abstract*—Code review is critical for ensuring software quality and maintainability. With the rapid growth in software scale and complexity, code review has become a bottleneck in the development process because of its time-consuming and knowledge-intensive nature and the shortage of experienced developers willing to review code. Several approaches have been proposed for automatically generating code reviews based on retrieval, neural machine translation, pre-trained models, or large language models (LLMs). These approaches mainly leverage historical code changes and review comments. However, a large amount of crucial information for code review, such as the context of code changes and prior review knowledge, has been overlooked. This paper proposes an LLM-based review knowledge-augmented, context-aware framework for code review generation, named LAURA. The framework integrates review exemplar retrieval, context augmentation, and systematic guidance to enhance the performance of ChatGPT-4o and DeepSeek v3 in generating code review comments. Besides, given the extensive low-quality reviews in existing datasets, we also constructed a high-quality dataset. Experimental results show that for both models, LAURA generates review comments that are either completely correct or at least helpful to developers in 42.2% and 40.4% of cases, respectively, significantly outperforming SOTA baselines. Furthermore, our ablation studies demonstrate that all components of LAURA contribute positively to improving comment quality.**

***Index Terms*—Code Review Generation, LLMs, Review Exemplar Retrieval, Context-aware**

I. Introduction

Code review, a process of having peers manually examine source code changes, is critical to software development. Both open source and industrial software projects conduct code review activities to identify defects and ensure software maintainability [1]–[3]. With the increasing growth of software scale and complexity, the drawbacks of modern code reviews, e.g., being time-consuming, cannot be overlooked, and many developers face a substantial review workload [4]. For example, in 2016, a core developer of the Linux kernel raised the argument [5]: "*We are getting more reviewers, but they are coming in slowly and are not anywhere near enough. As a result, the number of unprocessed patches is on the increase.*" To alleviate this burden, a substantial amount of studies has focused on automating the code review process, such as recommending the best reviewers [6]–[13], predicting code locations that need review [14], suggesting relevant code review comments [15], [16], and optimizing code before submission [17], [18].

Driven by the rapid development of artificial intelligence technologies, many studies have shifted towards automatically generating code reviews by leveraging neural machine translation and pre-trained models [18]–[20]. However, achieving satisfactory results remains challenging because of the complexity and knowledge-intensive nature of code review. The rise of large language models (LLMs) brings new opportunities for improvement, as they demonstrate exceptional performance in natural language understanding, generation, and a wide range of SE automation tasks [21]–[23]. However, directly using LLMs for code review generation faces several limitations. On one hand, LLMs often lack access to essential contextual information such as pull request (PR) details, code change specifics, and the domain-specific review experience that human reviewers rely on – all of which are crucial for producing high-quality reviews [24], [25]. On the other hand, LLMs lack systematic guidance when generating code reviews; they typically cannot understand well the review's scope and focus, nor do they operate within a well-defined logical framework to structure their feedback. This absence of context and structured guidance can lead to suboptimal review quality.

To overcome these limitations, we propose **LAURA**, a novel **L**LM **AU**gmentation Framework for Code **R**eview Gener**A**tion. LAURA consists of three components – context augmentation, review exemplar retrieval, and systematic guidance – combining augmentation and prompts to improve LLMs

†Corresponding author.

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00245

in code review generation. We leverage relevant contextual information of PRs and code changes and construct a history code review dataset to provide LLMs with relevant background knowledge and review experience. Specifically, since existing code review datasets do not have the context information we require and tend to have a quarter of low-quality data [26], we construct a new dataset with careful filtering. Our dataset contains 301,256 diff-comment-info series, which includes complete PR data from 1,807 popular GitHub projects. We manually annotate 384 recent entries for evaluation, while the remaining 298,494 entries dated before December 26, 2024, are used as the retrieval-augmented generation (RAG) database to provide valuable reference reviews. This enables the LLM to leverage relevant context and review experience, similar to a human reviewer. To address the second limitation – lack of guidance – we design prompts that provide clear instructions and a logical review structure, helping LLMs better understand review focus and generate coherent, high-quality comments.

We apply the LAURA framework to two well-performed LLMs, i.e., ChatGPT-4o and DeepSeek v3 (hereinafter, we name them as LAURA-GPT and LAURA-DS, respectively), and conduct experiments to validate the effectiveness of LAURA. To overcome the limitations of syntactic similarity-based metrics in evaluating LLM-generated text [27], we conduct both LLM-based and human evaluations. The results show that LAURA-GPT and LAURA-DS consistently rank at the top across four out of five LLM evaluation metrics for assessing the overall review generation performance – readability, relevance, sufficiency, and operability – only slightly trailing in brevity. Additionally, both models achieve I-Score and IH-Score values of 20.0%, 42.2% and 18.5%, 40.4% respectively, in human evaluation metrics, which measure the usefulness of the generated content, meaning that both models effectively identified issues in about one-fifth of cases and provided at least helpful review comments in over two-fifths of cases. Compared to their respective base models, LAURA-GPT and LAURA-DS improve their I-Score by 30.5% and 36.6%, and their IH-Score by 37.3% and 38.4%, respectively. This indicates a significant improvement over directly using ChatGPT-4o and DeepSeek v3. LAURA also significantly outperforms CodeReviewer [19], the pre-trained SOTA approach in code review generation. Furthermore, we conduct an ablation study to analyze the impact of LAURA's three components on code review quality, confirming the effectiveness of the approach.

In summary, this paper makes the following contributions:

- We design the first composite method, which aims to improve the performance of LLMs in automatic code review generation by context augmentation, review exemplar retrieval, and systematic guidance.
- We construct a high-quality dataset through hybrid filtering methods.
- Our proposed method outperforms the state of the art in automatic code review generation.
- We have made the retrieval-augmented data publicly available to facilitate future improvements in code review generation [28].

II. Approach Design

In this study, we propose an enhanced LLM approach to generate review comments for a given code change. This section provides a detailed description of the three components we designed to enhance LLMs, i.e., context augmentation, review exemplar retrieval, and systematic guidance. Figure 1 shows an overview of our approach.

A. Context Augmentation

In a pull request (PR), the core is code changes, which are the object for code review. However, other information in a pull request can also play an important role in understanding the rationale of these code changes. In practice, a reviewer can read the PR title and body to understand the primary goal of the code change. The reviewer can then use the commit message and file path to identify the general purpose and content of the change, and finally refer to the code context to fully understand the modification in the diff. All of this contextual information played an important role in helping the reviewer thoroughly understand the code changes and provide meaningful review feedback. We believe that providing this information can help LLMs generate more helpful comments.

However, existing studies only utilize code or diffs for code review generation. Meanwhile, since diffs by default only include three lines of context before and after, code context is often missing or insufficient. Therefore, in the context augmentation component, we integrate information that was mainly ignored before but is helpful to understand code changes and evaluate code issues. Specifically, we consider using the following information as a supplement and enhancement to code changes.

Pull request title. The PR title is a concise summary of the changes and often the first thing reviewers read to grasp the basic idea. Providing it to LLMs may help them infer the intent behind the changes and enhance their understanding.

Pull request body. The PR body typically explains the purpose, scope, and rationale of the changes, providing a context for the review. Supplying these details to LLMs can improve their overall understanding of the changes and help them more accurately identify potential issues.

Commit message. Each commit within a PR may cover different aspects of the changes, with messages documenting those changes [29]. Commit messages typically contain less information but are more fine-grained. Including the commit message can provide LLMs with more commit-specific contextual information and help them better understand the changes.

Changed file path. Similarly, the file path provides more fine-grained information compared to the commit message, hinting at the file's role and the nature of the changes. Prior work shows paths can aid in review tasks [16]. It could help LLMs assess change types and locate issues more effectively.

Code Context. Closely tied to the diff, code context is crucial for discovering issues that can only be identified by considering the surrounding code, which is usually the most

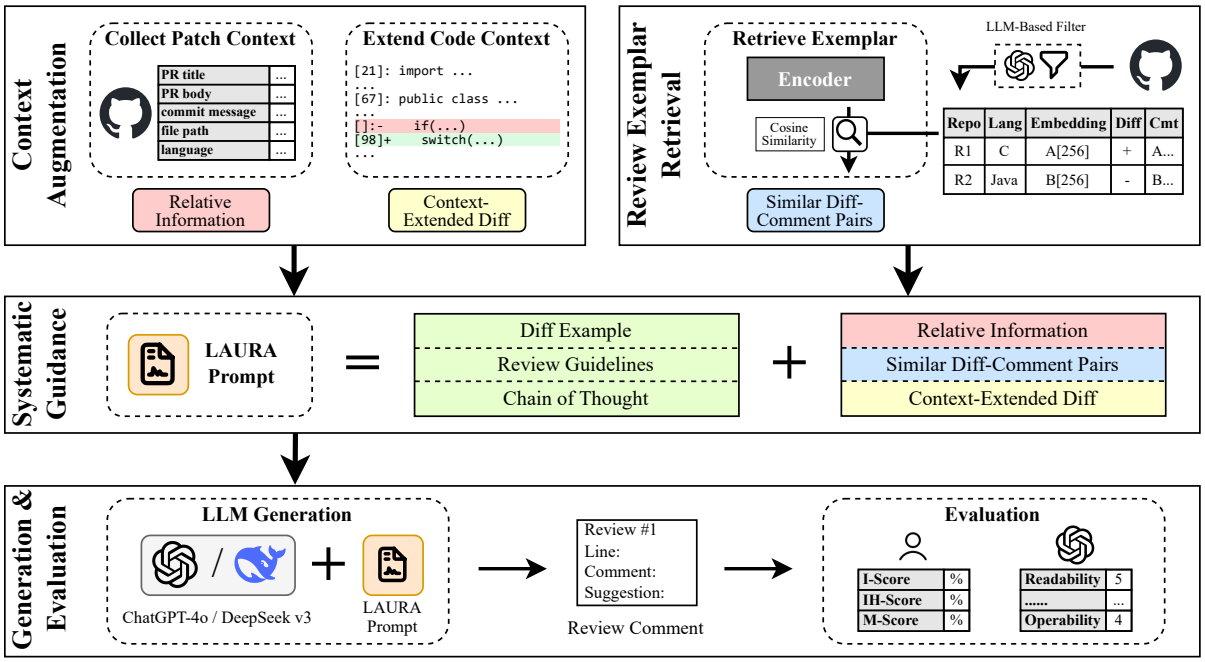

Fig. 1. Overview of the LAURA framework.

crucial information for identifying issues within the diff. Since diffs include limited code context by default, extending the code context is very important.

For each code change, we gather the relevant information mentioned earlier and provide all of it, except for code context, directly to the LLMs. We integrate the code context into the diff as follows: (1) We use Tree-sitter [30] to build an Abstract Syntax Tree (AST) of the source file corresponding to the diff. If possible, we extend the diff to the boundaries of the enclosing function/method, limited to three times the original diff length; otherwise, we extend it up to that length. (2) We extract all header/library/package imports appearing before the diff and include any not already in the expanded diff. (3) We annotate each line of the extended diff with line numbers: added lines as "[line number]+", unchanged lines as "[line number] ", and deleted lines as "[]-". In rare cases (under 3%) where the source file retrieved via the GitHub GraphQL API doesn't match the diff exactly, we only apply step (3). This approach enables us to create context-extended diffs, giving LLMs richer and more useful code context information.

Ultimately, we provide the LLMs with both the non-code context information and the enriched diff, enabling them to better review the code changes and generate more reliable review comments by leveraging these enhanced details.

B. Review Exemplar Retrieval

Existing studies indicate that the types of issues identified during code reviews and the level of usefulness of the reviews are related to the experience and knowledge of reviewers [24], [25]. Simply relying on LLMs can hardly grasp the pertinent knowledge from the vast amount of pre-training data. Thus, we hypothesize that providing LLMs with diffs similar to the code diff under review, along with the review comments made by other reviewers for those diffs, can serve as a reference for generating review comments. Prior studies [31], [32] applied retrieval-augmented approaches to automatically generate commit messages and achieved promising effectiveness. In this study, we first introduce the retrieval-augmented generation (RAG) method to retrieve diffs similar to the code diff under review and their corresponding review comments, using this retrieved information as part of the input to LLMs to achieve better quality in the generated review comments.

We use cosine similarity scoring [33], based on transformer encoders and vector embeddings, to assess code diff similarity and retrieve the most relevant code diffs and their review comments. For embedding, we use CodeT5+ [34], an enhanced version of CodeT5 [35], known for its Transformer encoder-decoder architecture and rich code representations from pre-training. We specifically use the "codet5p-110m-embedding" model to convert code diffs into 256-dimensional dense vectors, which encapsulate semantic information. Kartal et al. [36] show that transformers, when pre-trained and fine-tuned, outperform traditional methods like Word2Vec and Code2Vec in encoding embeddings, making them well-suited for retrieval-augmented generation (RAG). After embedding the code diffs, we compute cosine similarity scores to identify the most relevant code diff and its review comments, which will be used to enhance the performance of LLMs.

Given that the diff to be reviewed may be extremely large, we introduce a token count threshold t to balance cost and

performance. We provide review exemplars for the diff under review as follows: (1) For diffs with token count no more than t, we retrieve the most similar diffs (based on highest cosine similarity) from two sources – exemplars from the same repository and exemplars in the same programming language. If the two retrieved diffs are not the same, we provide both sets of diff-comment pairs; otherwise, we only provide the pair from the same programming language. (2) For diffs with token count more than t, we provide only the pair from the same programming language. We empirically set t = 2,048 in our experiments. This approach offers more references for small diffs while avoiding overly long input contexts for large diffs.

C. Systematic Guidance

We provide the augmented information, i.e., PR context and retrieved review exemplars, to LLMs along with the code diff under review through prompts. However, we still need to address the challenge that LLMs lack scientific guidance. Research shows that a well-designed prompt can guide LLMs in understanding the task and focusing on key points, leading to more structured, detailed, and accurate outputs, with its importance potentially surpassing that of the data itself [37]. Many automated tasks in software engineering have proved this, such as [38]–[40]. Thus, we also explore systematic guidance to enhance the performance of LLMs in the code review generation task. Inspired by relevant studies [37], [41], we design three prompt components – diff example, review guidelines, and chain of thought – to improve the generation quality of LLMs, focusing on input comprehension, task-critical point emphasis, and structured logical guidance, respectively.

Diff example. We provide an example of the extended code diff, along with a brief explanation of its components and format. Since our extension modifies the original diff structure by introducing longer code context and line numbers, we include this example and explanation to help LLMs better understand our input format and minimize potential misunderstandings.

Review guidelines. We design two sets of guidelines: review guidelines and code suggestion guidelines, to provide clearer instructions for LLMs in review generation. The review guidelines define the review's scope and focus, while the code suggestion guidelines ensure LLMs provide compliant, useful suggestions, as human reviewers often do. We instruct LLMs to assess the code diff from multiple perspectives, emphasize key aspects based on the change type, and avoid redundant or meaningless suggestions. These guidelines aim to guide LLMs towards more effective reviews and prevent unproductive output.

Chain of thought. LLMs also need guidance on reasoning and information use. Prior work shows the chain of thought (CoT) approach can improve review outcomes [26]. Thus, we adopt CoT to provide LLMs with a clear logical framework, guiding them to logically process the augmented information: first, understanding the intent and content of changes, then identifying and explaining issues, and finally offering suggestions. We also integrate contextual information into the CoT to help LLMs better leverage relevant details during review.

Additionally, we provide LLMs with requirements for the generation format, as having a consistent generation format helps ensure that the outputs of LLMs are more organized and easier to read. The Prompt we used in our experiments is shown in Figure 2. LAURA uses the full prompt, while the direct generation method uses the main prompt with all additional parts removed.

III. Experimental Design

In this section, we present our experimental design, outlining the research questions we focus on, the process of constructing the dataset, the evaluation metrics and methods used, and the implementation details.

A. Research Questions

To evaluate the impact of our proposed method on the performance of automatic code review generation, we design experiments to address the following two research questions.

RQ1: How does LAURA perform on code review generation compared to the baselines? In this RQ, we evaluate our method using ChatGPT-4o and DeepSeek v3 to generate review comments on our evaluation dataset, as they represent closed-source and open-source LLMs, respectively. We compare two setups: (1) LAURA guidance, and (2) direct generation, and benchmark against CodeReviewer [19]. Results are assessed via LLM and human evaluation.

RQ2: How much does each of the three components of LAURA contribute to its overall effectiveness? In this RQ, we perform an ablation study, comparing three setups: (1) LAURA without context augmentation, (2) LAURA without review exemplar retrieval, and (3) LAURA without systematic guidance. Evaluation follows the same methods.

B. Dataset Construction

Only a few code review generation datasets are available, notably the T5CR dataset [18] and the CodeReviewer dataset [19]. Many studies rely on the CodeReviewer dataset, but existing datasets suffer from quality issues. Tufano et al. found that around 25% of these datasets are of low quality, with the CodeReviewer dataset [26] reaching about 32%. This significantly limits dataset reuse. Furthermore, these datasets lack contextual information beyond the code itself and remove the source links of PRs, further constraining their usefulness. Therefore, we construct our own code review dataset with careful filtering to ensure high quality.

1) Project Selection: We selected high-quality open-source projects hosted on GitHub as case studies for our research. An overview of the project selection process is shown in Table I. Specifically, we focused on projects primarily written in one of four programming languages: C, C++, Java, and Python. This selection covers a broad range of programming paradigms and includes languages that are widely used. We first sorted these projects in descending order by the number of stars and retained only those with at least 2,500 stars. As a result, we

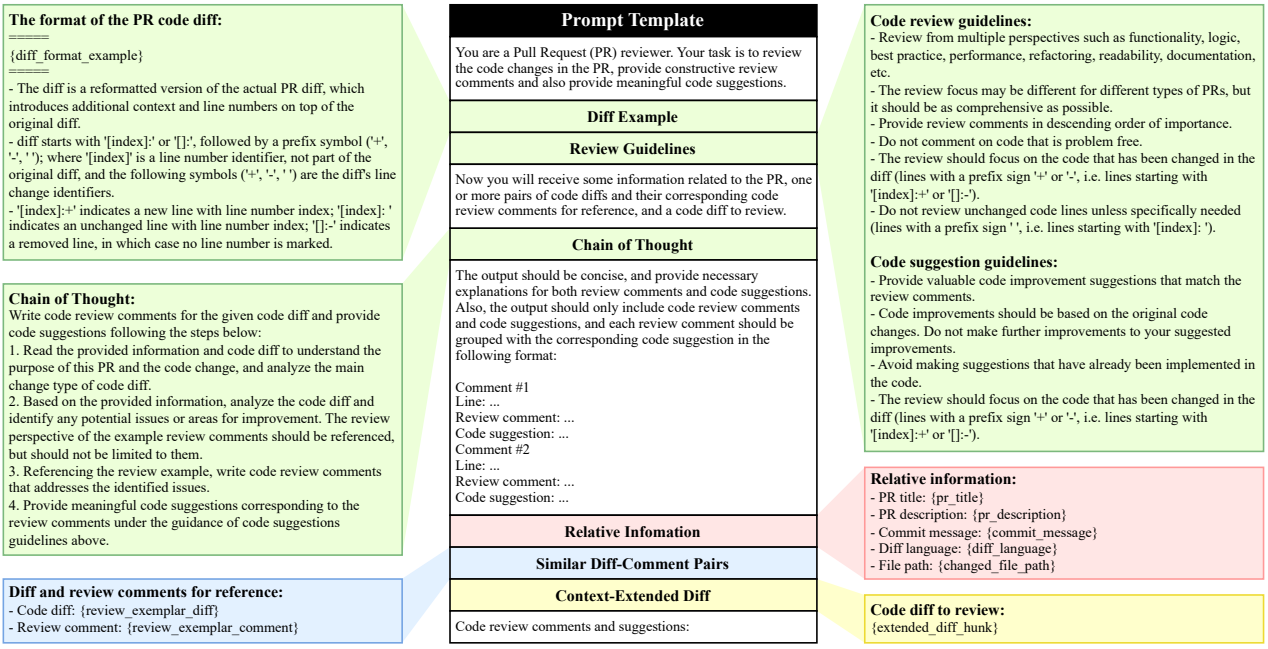

Fig. 2. Prompts used for LAURA and direct generation.

TABLE I
STATISTICS OF DATASET PREPARATION.

Procedure	Count
Project selection	# Projects
Preliminary selection of projects	6,467
Filtered (PR count <500)	-4.607
Filtered (manual check)	-53
Remaining projects	1,807
Data collection and cleaning	# Diff-comment-info series
Preliminary collection of data	1,020,537
Filtered (rule-based)	-202,704
Filtered (LLM-based)	-446,002
Merged (comments on the same diff)	-70,575
Remaining data	301,256

obtained 6,467 repositories, with the number of repositories for C, C++, Java, and Python being 782, 1,089, 1,384, and 3,212, respectively. We then filtered for repositories with at least 500 pull requests, narrowing it down to 1,860, as those with fewer PRs may lack activity or proper code reviews. Finally, we conducted a manual review and removed the following types of projects: (1) tutorial-oriented projects; and (2) projects that were forked from other repositories. Ultimately, we selected 1,807 repositories, including 228 C, 420 C++, 346 Java, and 813 Python repositories. In the next step, we collected pull request information from these repositories.

2) Data Collection: Table I summarizes our data collection and cleaning process. We primarily used the GitHub GraphQL API [42] to retrieve pull request data, as its flexible query system allows precise JSON data retrieval via HTTP POST requests. We collected PR titles, bodies, numbers, states, authors, reviewers, review comments (with timestamps), commit messages, diffs, and file paths – collectively referred to as a "diff-comment-info series." We filtered out cases where the PR author and reviewer were the same. Since comment-associated diffs were truncated, we fetched complete diffs via the GitHub commit-comparing function [43].

To improve data quality, we applied the 10-line rule [44], which assumes that code changes within 10 lines of a review location likely address the review and thus indicate valuable feedback. Following Rong et al. [45], we sorted commits and reviews chronologically within PRs and checked whether later commits modified code near earlier review locations. We kept only data conforming to the 10-line rule. After initial processing, we obtained 1,020,537 diff-comment-info series, but further filtering was necessary as the 10-line rule alone does not guarantee high quality.

3) Data Filtering: After data collection, we cleaned the collected data. First, we applied rule-based filtering: (1) We removed bot-generated comments by filtering reviewer names with "bot", "-cr", or "gpt" suffixes, and by using Golzadeh et al.'s bot list [46]. (2) We filtered invalid and irrelevant comments by (a) removing comments with two or fewer words, as they typically offer little value [47], [48], and (b) applying Tufano et al.'s heuristic algorithm based on manually designed patterns [18]. (3) We removed comments containing non-ASCII characters. After these steps, we retained 817,833 diff-comment-info series.

Next, we applied an LLM-based filter to improve review quality. As Tufano et al. [26] note, 25% of existing review comments remain low-quality despite basic filtering. We used

GPT-4o mini to evaluate each comment, its corresponding diff, and a task prompt: "Determine whether the review comments provided are valuable for improving the code diff. If the review comments point out problems in the code diff or give useful improvement suggestions, and are easy to understand, answer 'yes'; if the review comments are irrelevant to the content of the code diff (such as only requiring it to be done in a separate PR or commit), or are difficult to understand or not clearly described, answer 'no'. The answer should be one of 'yes' or 'no', without other content."

To assess the feasibility and effectiveness of this filtering method, we manually reviewed 240 random samples (60 per language), identifying 65 low-quality comments (27.1%). We treated high-quality comments as positive samples, and the LLM filter achieved an accuracy of 0.914 and a recall of 0.606. This indicates that although some high-quality samples may be filtered out, the proportion of low-quality samples in the final constructed dataset is significantly reduced. While not ideal for precise quality evaluation, the method is effective for filtering, as retaining low-quality comments is more harmful than discarding a few valuable ones. After this step, we had 371,831 diff-comment-info series.

Finally, we merged all review comments for the same code diff within the same pull request to provide richer reference comments for retrieval and evaluation. This yielded 301,256 diff-comment-info series. After data collection and cleaning, we finalized the context-enhancement component for our hybrid method.

4) Retrieval System Construction: We split the constructed dataset by time and selected 298,494 diff-comment-info samples prior to December 26, 2024, as the retrieval-augmented generation database. To improve retrieval efficiency, we pre-embedded the code diffs for similarity comparison using the "codet5p-110m-embedding" model [34], generating 256-dimensional vectors. For test diffs, we embedded them with the same model and calculated cosine similarity scores against the RAG database. The most similar diffs and their review comments were merged into the test diff-comment-info series as review references for the LLM. With data partitioning and post-processing complete, we finalized the review exemplar retrieval component for our composite method.

5) Evaluation Dataset Construction: Since LAURA involves LLM augmentation and human evaluation, large-scale experiments aren't feasible. We intended to sample 384 instances for evaluation (95% confidence level, 5% margin of error). Given the possibility of residual low-quality data in the LLM-filtered set and the risk of bias from discarding valuable comments, we opted to manually annotate high-quality data from the dataset without LLM filtering. We manually annotated high-quality code review comments based on the following criteria: (1) Readability and comprehensibility (this criterion excluded comments like "s/Check less/Check if less/" that were difficult to understand); (2) Clear relevance to issues in the code under review (this criterion excluded comments with low relevance to the code itself, such as "I don't think this should be part of this PR?"); (3) Containing at least an effective issue explanation or code suggestion (this criterion excluded comments with low informational value, such as "When does this happen?" or "Thanks"); (4) Self-contained understanding (this criterion excluded comments like "Ditto, also change this" that lacked essential contextual information). The annotation continues until we obtain 384 valid samples. To minimize data leakage risk (i.e., the chance that LLMs had seen the data during training), we only selected the diff-comment-info series dated after December 26, 2024 (the release date of DeepSeek v3, the latest model used in our experiments). We merged and shuffled the data. Then, the first two authors independently annotated each sample. A Cohen's kappa [49] of 0.883 between their results, indicating a high level of inter-rater agreement. Subsequently, any disagreements were resolved by conducting several meetings and involving a third author as arbiter. In total, we annotated 546 samples, yielding 384 high-quality samples for evaluation.

C. Baselines

Tufano et al. [26] compared ChatGPT-3 with T5CR [18], CodeReviewer [19], and CommentFinder [15] in code review generation/recommendation, concluding that SOTA methods outperform ChatGPT-3. Among these methods, **CodeReviewer** is also designed for reviewing code diffs. Therefore, we choose CodeReviewer as a baseline to evaluate our method against traditional pre-trained models. Additionally, since LAURA is based on **ChatGPT-4o** and **DeepSeek v3**[1], we also select ChatGPT-4o and DeepSeek v3 as additional baselines to evaluate the performance improvements introduced by our method.

D. Evaluation

Given the nature of LLMs and the diversity of natural language [27], it is impractical to expect LLM outputs to closely match human reviewer comments. Thus, traditional metrics like BLEU and ROUGE are not suitable [50]. To efficiently evaluate the overall performance of the review comments generated by LAURA, we adopt an LLM-based evaluation approach. However, LLM evaluation has limitations when it comes to assessing the practical usefulness of review comments in real-world scenarios. Therefore, we also introduce a human evaluation approach.

1) LLM Evaluation: We follow the LLM-based Deep-CRCEval method by Lu et al. [51], with some modifications: (1) merging some metrics to improve clarity; (2) instructing the LLM to rate each comment on a 1–5 scale according to five metrics (higher scores indicating better quality), instead of the original 10-point scale, since prior study [52] shows that a 5-point scale is easier to understand and apply accurately, avoids the coarseness of a 3-point scale, and provides greater stability and consistency than a 10-point scale; (3) adding simple scoring guidelines for each metric. For example, for the "Readability" metric, we provided the LLM with the following explanation: "In the 1-5 scoring system, a score of 5 means the

[1]We exclude DeepSeek R1 due to insufficient data after its release date.

text is completely clear and readable; a score of 3 means the expressed meaning can be basically understood; and a score of 1 means the text is nearly unreadable or incomprehensible." We believe these scoring guidelines help calibrate scores across different evaluation samples and positively impact the consistency of the LLM. Table II shows the final metrics.

For RQ1, we provided ChatGPT-4o (specifically, GPT-4o-2024-11-20) with five anonymized sets of review comments per data sample – generated by LAURA-GPT, LAURA-DS, ChatGPT-4o only, DeepSeek v3 only, and CodeReviewer – and asked it to score them using the five metrics. For RQ2, due to LLM input length limits and the relative nature of our LLM-based approach, we split the evaluation. Each data sample was paired with two anonymized sets of five review comments: one from LAURA-GPT, its three ablations, and ChatGPT-4o only; the other from LAURA-DS, its three ablations, and DeepSeek v3 only. ChatGPT-4o scored each set using the same metrics.

2) Human Evaluation: Sometimes, LLM-based evaluation may not effectively assess the usefulness of the generated review comments in real-world scenarios. Therefore, we also conduct a human evaluation. Following Mahajan et al. [53], we adjust their method for LLM-generated results (Table III): we remove the "Unavailable" category, redefine the other three to suit our goals, and add an "Uncertain" category. This is because, in our evaluation, all test data have corresponding generated review comments, so the "Unavailable" case does not apply. However, given the limited knowledge of the repository and PR context, these review comments could be difficult to evaluate, leading to "Uncertain" cases. Specifically, since the data comes from 1,807 open-source projects, we cannot fully understand every project's code requirements, organizational structure, or other contextual details. Therefore, even if we determine that the LLM-generated code review comments are generally useful, without factual grounding, we cannot be certain whether they would be accepted within the specific project context of the reviewed code. We categorize each generated review comment as "Instrumental", "Helpful", "Uncertain", or "Misleading", and use three metrics to evaluate quality.

I-Score ($\frac{I}{I+H+U+M} \times 100\%$): Proportion of comments rated as "Instrumental", reflecting the ability to correctly identify and address issues in the code diff. Higher is better.

IH-Score ($\frac{I+H}{I+H+U+M} \times 100\%$): Proportion of comments rated as "Instrumental" or "Helpful", indicating the overall usefulness of comments. Higher is better.

M-Score ($\frac{M}{I+H+U+M} \times 100\%$): Proportion of "Misleading" comments that contain factual errors, incorrect suggestions, or fail to address the code diff. Lower is better.

For RQ1, we recruited four volunteers, each with over five years of software development experience and programming proficiency. To manually assess the 384 results per method through a rigorous multi-stage annotation process: initial calibration with shared examples, a discussion to align scoring, and then independent ratings of the remaining dataset. First, each evaluator reviewed the initial 32 results per method together, learning from each other, gaining different perspectives, and aligning on scoring criteria. Then, the evaluators independently evaluated the next 96 results per method, achieving a Krippendorff's Alpha of 0.804 with $p < 0.001$, indicating good agreement. Based on this consensus and considering evaluation efficiency, the remaining 256 diff-comment-info series were evenly distributed among the four evaluators. To minimize potential bias, we randomly shuffled the evaluation samples, which means the evaluators were blinded to comment sources. For RQ2, the same four evaluators evaluated the ablation experiment results on the 384 diff-comment-info series. Based on their evaluation experience in RQ1, the evaluators evenly divided the 384 evaluations, each handling 96 results. We report the ablation evaluation results for these 384 samples.

TABLE II
ADJUSTED LLM EVALUATION METRICS.

Metric	Description
Readability	Clear, easily understandable language.
Relevance	Directly related to the specific code.
Brevity	Clearly explain the issues identified, and at the same time, be concise and not lengthy.
Sufficiency	Cover all issues as much as possible, and point out the exact location of issues for comprehensive review.
Operability	Practical advice for addressing identified issues.

TABLE III
ADJUSTED HUMAN EVALUATION CATEGORIES.

Category	Description
Instrumental (I)	The review fully matches the ground truth, that is, identifying all issues or providing equivalent solutions, with no obvious errors.
Helpful (H)	The review partially matches the ground truth, that is, identifying some or all issues, or at least providing related suggestions, but may lack completeness or accuracy.
Uncertain (U)	The review does not overlap with the ground truth but offers suggestions on other aspects, with no errors or direct conflicts. Its usefulness is unclear based on the ground truth alone.
Misleading (M)	The review directly contradicts the ground truth, provides irrelevant or incomprehensible feedback, or offers no meaningful input.

E. Experimental Details

The dataset was prepared on a platform with two NVIDIA GeForce RTX 3090 GPUs. During the data split step, we used the "codet5p-110m-embedding" model to embed code diffs, setting the "max_length" to 2048 to balance input size and efficiency – this setting ensured high efficiency while only 5.3% of diffs were truncated. All other parameters remained at default values.

Experiments were conducted via the APIs of ChatGPT-4o (i.e., GPT-4o-2024-11-20) and DeepSeek v3, ensuring each data entry was tested independently with no external context beyond the designed prompts and inputs. We set the API temperature to 0.5, as this produced better outputs in preliminary tests. For CodeReviewer, another baseline model in RQ1, we performed additional fine-tuning using our custom

dataset on the same platform. Since CodeReviewer had already been fine-tuned on its original dataset and demonstrated a strong understanding of the code review generation task, we further fine-tuned it on our dataset for 10 epochs, keeping all other parameters the same as those used in the original fine-tuning process.

IV. Experimental Results

In this section, we present the results of the effectiveness of LAURA and the impact of each of its three components.

A. RQ1: Effectiveness of LAURA

In this RQ, we ask ChatGPT-4o and DeepSeek v3 to review code diffs using the LAURA framework and also directly without it. Code diffs are also submitted to CodeReviewer for evaluation. Table IV shows the results, reporting five LLM evaluation metrics, along with I-Score, IH-Score, and M-Score from human evaluation. The results show that LAURA-GPT achieves the highest I-Score of 20.0% and IH-Score of 42.2%, representing 30.5% and 37.3% improvements over directly using ChatGPT-4o, while also maintaining the lowest M-Score (0.8%). It excels in four out of five LLM-evaluated metrics, with only a slight weakness in brevity. LAURA-DS ranks second, with an I-Score of 18.5% and an IH-Score of 40.4%, achieving improvements of 36.6% and 38.4% over directly using DeepSeek v3, respectively. Both methods perform well across most metrics, indicating that our method helps LLMs better identify areas for improvement in code changes. In contrast, CodeReviewer performs worst, with an I-Score of 3.4%, IH-Score of 10.2%, and the lowest scores in all LLM metrics, showing LLMs' superiority over traditional pre-trained models in realistic code review scenarios.

We provide a human evaluation example in Figure 3. In this case, the diff under review modifies a string construction method, changing *"std::string(ptr, buf.ndim == 0 ? buf.itemsize : buf.strides[0])"* to *"std::string(ptr, strnlen(ptr, buf.itemsize))"*. The ground truth highlights a string truncation risk and suggests adding tests. LAURA-GPT identifies both issues, advises reverting the change, and recommends tests, earning an "Instrumental" rating. LAURA-DS detects the string truncation risk but omits the test suggestion, earning a "Helpful" rating. CodeReviewer simply approves the change without further suggestions, conflicting with the ground truth and receiving a "Misleading" rating.

The improvements are expected, as adding code change-related information equips LLMs with rich background knowledge, helping align their understanding with that of human reviewers. Retrieving similar code diffs and corresponding comments offers valuable reference experience, much like how human reviewers draw on past knowledge. As shown in Figure 4, in case 1, the additional code context provided by LAURA highlights code duplication related to the diff, enabling GPT-4o to generate a review comment consistent with the ground truth. In case 2, the review exemplar provided by LAURA addresses the same issue of redundant "displayName" as the ground truth, thereby assisting DeepSeek v3 in generating a valuable review comment. These two real-world cases highlight the significant assistance provided by LAURA's helpful information in enabling LLMs to generate review comments.

In addition, in our experiments, the performance of directly using ChatGPT-4o for code review generation even surpasses that of CodeReviewer, which contrasts with the results of Tufano et al. [26]. We attribute this to advances in LLMs since ChatGPT-3, dataset differences, and varied evaluation methods. Our dataset, possibly containing more long code diffs, may have overwhelmed pretrained models. Additionally, Tufano et al. evaluated the three traditional methods as a whole, which also leads to differences in results.

> **RQ1 Summary:** LAURA significantly outperforms the traditional pre-trained model, CodeReviewer. Moreover, both LAURA-GPT and LAURA-DS achieve better performance compared to directly using the base model, indicating that this composite approach – based on data augmentation and rapid fine-tuning – effectively enhances the performance of LLMs in code review generation tasks.

B. RQ2: Ablation Study of the Three Key Components of LAURA

In this RQ, we instruct ChatGPT-4o and DeepSeek v3 to review code changes under conditions where one of the three components of LAURA is removed. Table V presents the experimental results, where we report the scores of comments generated by different methods on five metrics as evaluated by LLM, along with the I-Score, IH-Score, and M-Score of these comments as assessed by human evaluation. To make the comparison clearer, we also show the performance of LAURA-GPT and LAURA-DS with all three components in Table V shaded in light grey.

The ablation study results indicate that all three components of LAURA play important roles. Among them, the method that removes only the review exemplar retrieval component performs slightly better overall – achieving the best LLM and human evaluation results within its respective ablation experiment group, whether based on ChatGPT-4o or DeepSeek v3. However, removing the review exemplar retrieval component also leads to a decrease in the I-Score of LAURA-GPT and LAURA-DS by 9.1% and 8.4%, respectively, and a decrease in the IH-Score by 10.5% and 8.4%, respectively. The methods that remove only the context augmentation component or only the systematic guidance component each have their own strengths and weaknesses, with no significant difference between them. However, all these methods show a decrease in LLM evaluation metrics compared to the complete method, while the I-Score in human evaluation dropped by 10.4% - 11.7%, and the IH-Score dropped by 11.0% - 12.2%.

These results indicate that all three components have a significant positive impact on improving the quality of LLM-generated outputs. This suggests that rich relevant information

TABLE IV
EVALUATION RESULTS OF CODE REVIEW GENERATION.

Method	LLM Evaluation					Human Evaluation		
	Readability	Relevance	Brevity	Sufficiency	Operability	I-Score	IH-Score	M-Score
LAURA-GPT	**4.956**	**4.956**	4.185	**4.859**	**4.953**	**20.0%**	**42.2%**	**0.8%**
ChatGPT-4o	4.162	4.040	**4.235**	3.566	3.855	15.4%	30.7%	1.3%
LAURA-DS	4.761	4.786	4.151	4.556	4.686	18.5%	40.4%	1.0%
DeepSeek v3	4.031	3.861	4.183	3.255	3.635	13.5%	29.2%	1.0%
CodeReviewer	2.182	1.909	2.859	1.188	1.404	3.4%	10.2%	9.1%

The highest and second-highest results are in bold and underlined, respectively.

Review Comments:

LAURA-GPT: (I)

Comment #1
Line: [180]
Review comment: Using strnlen(ptr, buf.itemsize) improves safety by avoiding overreads, but it can **lead to incorrect truncation of strings containing valid null characters**, which is a potential issue. This may not match expected behavior if such characters are valid input. Consider clarifying the expected input semantics or handling this explicitly.
.....
Comment #3
Line: [182]
Review comment: This function handles a sensitive memory operation. Since the behavior has changed regarding null-padding, consider **adding tests that cover edge cases such as strings with embedded null characters and maximum-length strings**. This will ensure the change behaves as intended and is robust across all Python string representations.

LAURA-DS (H)

Comment #1
Line: 180
Review comment: The change from using `buf.strides[0]` to `strnlen` is good for removing padding, but we **should consider cases where the string might contain null bytes intentionally** (binary data). The current implementation would truncate at the first null byte.....

CodeReviewer: (M)

[Line #43]: strnlen(ptr, buf.itemsize) as we have already checked that buf.ndim equals at this point

Ground Truth:

[Line #180]: Can tests be added e.g. using data (or more cases) from PR description?
What about case when input buffer has special characters inside the string e.g. `Tes\u0000t`? The `strnlen` can stop on first null character and string will be truncated to `Tes`

Fig. 3. A real example of human evaluation of code review comments generated by LAURA-GPT, LAURA-DS, and CodeReviewer.

TABLE V
EVALUATION RESULTS OF THE ABLATION STUDY.

Method	LLM Evaluation					Human Evaluation		
	Readability	Relevance	Brevity	Sufficiency	Operability	I-Score	IH-Score	M-Score
LAURA-GPT	**4.956**	**4.956**	**4.185**	**4.859**	**4.953**	**20.0%**	**42.2%**	**0.8%**
w/o context augmentation	4.393	4.341	4.140	3.838	4.101	17.7%	37.5%	**0.8%**
w/o review exemplar retrieval	4.561	4.522	4.166	4.163	4.314	18.2%	37.8%	**0.8%**
w/o systematic guidance	4.281	4.205	3.888	3.942	4.088	18.0%	37.2%	**0.8%**
LAURA-DS	**4.761**	**4.786**	**4.151**	**4.556**	**4.686**	**18.5%**	**40.4%**	**1.0%**
w/o context augmentation	4.244	4.207	3.976	3.709	3.953	16.4%	35.9%	**1.0%**
w/o review exemplar retrieval	4.565	4.529	4.065	4.100	4.293	16.9%	37.0%	**1.0%**
w/o systematic guidance	4.321	4.309	3.929	3.916	4.110	16.4%	35.4%	**1.0%**

The highest result in each group of results is highlighted in bold.

and highly related code context can indeed help LLMs better understand code changes, leading to more useful review comments. Providing exemplar reviews that reflect the experience of human reviewers can help LLMs more effectively assess review priorities and generate valuable feedback. Moreover, offering more detailed and precise instructions and guidance to LLMs can effectively enhance the usefulness of their comments.

Notably, LAURA's performance gains in code review generation are not simply the sum of its three components, but the result of their interactions. The sum of the performance drop from removing each component individually exceeds the drop from removing all three (i.e., the baseline models). The slight advantage of removing only the review exemplar retrieval component, along with our evaluation observation that removing the systematic guidance component occasionally leads to comments targeting the exemplar instead of the diff, suggests that systematic guidance not only guides LLMs in reviewing but also helps them interpret the extended diff provided by the context augmentation component. Likewise, the extended diff helps LLMs better understand the review exemplars, enhancing the review exemplar retrieval component's effectiveness. Together, these components reinforce each other, offering additional benefits to LLMs.

RQ2 Summary: The three components of LAURA – context augmentation, review exemplar retrieval, and systematic guidance – all have a significant positive impact on the performance of LLMs. When used in combination, these components produce a compounding effect, further enhancing the relevance and usefulness of the review comments generated by LLMs.

Case #1

Diff

```
...
[207]:+        if isinstance(
[208]:+            request,
[209]:+            (EmbeddingChatRequest, EmbeddingCompletionRequest, ScoreRequest)):
[210]:+
[211]:+            operation = "score" if isinstance(request, ScoreRequest) \
[212]:+                else "embedding generation"
[213]:+            if token_num > self.max_model_len:
[214]:+                raise ValueError(
...
[219]:+            return TextTokensPrompt(prompt=input_text,
[220]:+                                    prompt_token_ids=input_ids)
[221]:+
[222]:+        # Score API
```

Extended Code Context

```
[223]:         if isinstance(request,
[224]:                 (EmbeddingChatRequest, EmbeddingCompletionRequest)):
[225]:         if token_num > self.max_model_len:
[226]:             raise ValueError(
...
[231]:         return TextTokensPrompt(prompt=input_text,
[232]:                                 prompt_token_ids=input_ids)
[233]:
...
```

Ground Truth

[Line #225]: This code looks **duplicated**.

LAURA-GPT Generation

Comment #1
Line: [207]-[220]
Review comment: The newly added validation for `ScoreRequest` and `EmbeddingRequest` is clear and ensures input length does not exceed `max_model_len`. However, there is a **redundancy in the validation logic for `EmbeddingRequest` and `ScoreRequest` between lines [207]-[220] and [223]-[230]**. The logic for checking `token_num > self.max_model_len` is repeated, which can be consolidated for better maintainability and readability.

Case #2

Diff

```
...
[171]:+
[172]:+    public static final PropertyDescriptor MAX_STRING_LENGTH = new
PropertyDescriptor.Builder()
[173]:+            .name("Max String Length")
[174]:+            .displayName("Max String Length")
[175]:+            .description("The maximum allowed length of a string value when
parsing the JSON document")
[176]:+            .required(true)
[177]:+            .defaultValue(DEFAULT_MAX_STRING_LENGTH)
[178]:+            .addValidator(StandardValidators.DATA_SIZE_VALIDATOR)
[179]:+            .build();
[180]:+
...
```

Review Exemplar

```
...
[189]:+    PropertyDescriptor MAX_JSON_FIELD_STRING_LENGTH = ...
[190]:+            .name("Max JSON Field String Length")
[191]:+            .displayName("Max JSON FieldString Length")
...
```

[Line #191]: In this case, recommend **removing the `displayName`** since it should be the same as the name (although there appears to be a space missing between `Field` and `String`).

Ground Truth

[Line #174]: Since this is a new property, **the displayName is not needed**.
```suggestion
```

LAURA-DS Generation

Comment #2
Line: [173]
Review comment: **The `displayName` is redundant** here since it matches the `name` exactly. It's better to omit it unless there's a need for a different display name.
Code suggestion: Remove the `displayName` line since it's identical to `name`.

Fig. 4. Two real cases of how LAURA assists LLMs in generating useful review comments.

V. Threats to Validity

Threats to internal validity. The main threat to internal validity is the potential risk of data leakage. We sampled test data after the release of the latest model we used (DeepSeek v3) to avoid overlaps with the training data of LLMs. However, we cannot fully guarantee that the test data wasn't used in model optimization post-release. Another threat is the length limit of extended diffs, as defining the optimal code context scope is challenging. Too little may miss details; too much may add noise, distract LLMs, and degrade output quality. Further research is needed to address this. In addition, the limitations of prompt exploration pose another threat to internal validity. Although we designed a composite prompt for generating code review comments based on prior research and experimental observations, there may exist better prompt designs and ordering strategies that could further enhance LAURA's performance.

Threats to external validity. A threat to external validity lies in the quality of the dataset used during the research process. We used rule-based and LLM-based filters to remove low-quality comments, but some may remain. For the evaluation set, we relied on manual annotation to minimize this issue. Another threat to external validity concerns the choice of programming languages. Our experiments focused on C, C++, Java, and Python, so it's unclear how well the results generalize to other languages. Our methods may serve as a reference for future research. Baseline selection may also pose a threat to external validity. Although we included open-source LLMs, closed-source LLMs, and state-of-the-art pretrained models as baselines, we did not incorporate some baselines due to factors such as differences in dataset structure and variations in programming language coverage. There is a promising avenue to evaluating LAURA on more LLMs.

Threats to construct validity. A threat to construct validity lies in the choice of LLMs. Although LAURA is model-agnostic, we used ChatGPT-4o and DeepSeek v3 as representative state-of-the-art models. This choice may influence results. Another threat to construct validity concerns the selection of evaluation metrics. LLM-based evaluation metrics have inherent variability across datasets, and while we mitigated this by merging metrics and adding scoring criteria, limitations persist. Our human evaluation focuses on basic factual alignment, so the practical usefulness of mismatched review comments remains uncertain.

VI. Related Work

In this section, we introduce two areas related to our study: code review automation and large language models, and we discuss the relevant studies in each of these fields.

A. Code Review Automation

The code review process requires substantial time and effort from both parties. To improve efficiency, many studies have explored automating parts, including reviewer recommendation, review necessity prediction, identifying code areas for review, pre-review and post-review refinement, code review generation, and comment suggestion.

Early studies mainly focused on reviewer recommendation, using methods such as file path analysis [6], traceability graphs [7], load balancing [8], collaborative filtering [9], and balancing expertise and workload [10]. Other approaches considered cross-project collaboration [12] and composite analyses of file paths, project details, and author info [13]. Some research also aimed to predict which parts of the submitted code would need comments [14].

Code review automation has advanced to include tasks like review necessity prediction, comment generation, and code refinement. Notable works include T5CR [18], CodeReviewer

[19], and CommentFinder [15]. T5CR, based on the Text-To-Text Transfer Transformer (T5) model [54], handles method-level code for pre-review optimization, comment generation, and post-review refinement. CodeReviewer uses T5 architecture with CodeT5 initialization [35], processes line-level diffs, and supports review necessity prediction, comment generation, and code refinement across nine languages. CommentFinder is an information retrieval (IR)-based method that uses Bag of Words (BoW) and similarity measures to find similar code snippets and recommend relevant review comments.

So far, several works have attempted to apply LLMs to code review automation tasks. Lu et al. [20] used the LLaMA model for review necessity prediction, comment generation, and code refinement, achieving results comparable to pre-trained models via parameter-efficient fine-tuning. Guo et al. [23] showed that ChatGPT-3.5 outperformed CodeReviewer in post-review refinement. Pornprasit et al. [40] explored ChatGPT-3.5's performance in both pre- and post-review refinement, analyzing few-shot learning effects. Tufano et al. [26] compared ChatGPT-3 with three existing methods and found it achieved state-of-the-art performance in code refinement, but lagged in review generation. However, these studies provide only a limited exploration of LLMs and have not fully unlocked their potential for code review automation tasks.

In this work, we use line-level code diff hunks as the raw input and focus on code review generation, aiming to further explore ways to improve LLM performance on this task.

B. Large Language Model

Language models have evolved from early statistical models to neural, pre-trained models, and now to large language models (LLMs) with over 10B parameters, trained on vast datasets [55]. LLMs show superior performance and emergent capabilities [56], often matching or surpassing task-specific pre-trained models. Notable LLMs include ChatGPT-4o, which excels across benchmarks [57], and DeepSeek v3 [58] among open-source models. Our research builds on ChatGPT-4o and DeepSeek v3, applying the LAURA framework to both.

So far, LLMs have been widely used in software engineering tasks like vulnerability prediction [22], [59], [60] and commit message generation [21], [61], [62]. Many studies have explored ways to improve the performance of LLMs on specific tasks. Generally, LLMs can be further enhanced through methods such as augmenting input information [63], retrieval-augmented generation [64], and prompt engineering [65]. Augmenting input information is based on the intuitive idea that providing the model with richer contextual information helps it better understand the task and produce better results. Retrieval-augmented generation [66] enhances the model's task awareness by incorporating knowledge from external databases, thereby strengthening the model's generation capabilities. Prompt engineering [67] involves designing prompts for the model, guiding it through instructions on the actions it must perform with the given input, and ensuring that the LLMs achieve high performance in downstream tasks.

VII. Conclusion

In this paper, we introduce LAURA, a novel LLM-based generation framework for code review, which enhances the quality of code review comments generated by LLMs through three components: context augmentation, review exemplar retrieval, and systematic guidance. We applied LAURA to ChatGPT-4o and DeepSeek v3, and demonstrated through both LLM-based and human evaluations that LAURA significantly improves their performance on code review generation tasks. Experimental results show that LAURA outperforms direct use of LLMs for code review generation and substantially surpasses the performance of the pretrained model CodeReviewer.

Furthermore, our findings indicate that each of the three components – context augmentation, review exemplar retrieval, and systematic guidance – contributes to improving LLM performance in code review generation tasks, with the combination of all three yielding the best results. To further enhance the quality of generated code review comments, future research should explore additional useful input contexts and better methods for input integration.

Acknowledgments

This work is supported by the National Natural Science Foundation of China Grants (62202048, 62232003, 62141209, 62402482, and 62572048).

References

[1] G. Bavota and B. Russo, "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)*, pp. 81–90, IEEE, 2015.

[2] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects," in *Proceedings of the 11th working conference on mining software repositories*, pp. 192–201, 2014.

[3] C. Sadowski, E. Söderberg, L. Church, M. Sipko, and A. Bacchelli, "Modern code review: a case study at google," in *Proceedings of the 40th international conference on software engineering: Software engineering in practice*, pp. 181–190, 2018.

[4] M. Zhou, Q. Chen, A. Mockus, and F. Wu, "On the scalability of linux kernel maintainers' work," in *Proceedings of the 2017 11th joint meeting on foundations of software engineering*, pp. 27–37, 2017.

[5] J. Corbet, "On linux kernel maintainer scalability [lwn.net]." Website, 2016. https://lwn.net/Articles/703005/.

[6] P. Thongtanunam, C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida, and K.-i. Matsumoto, "Who should review my code? a file location-based code-reviewer recommendation approach for modern code review," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pp. 141–150, IEEE, 2015.

[7] E. Sülün, "Suggesting reviewers of software artifacts using traceability graphs," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1250–1252, 2019.

[8] S. Asthana, R. Kumar, R. Bhagwan, C. Bird, C. Bansal, C. Maddila, S. Mehta, and B. Ashok, "Whodo: Automating reviewer suggestions at scale," in *Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*, pp. 937–945, 2019.

[9] A. Chueshev, J. Lawall, R. Bendraou, and T. Ziadi, "Expanding the number of reviewers in open-source projects by recommending appropriate developers," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 499–510, IEEE, 2020.

[10] E. Mirsaeedi and P. C. Rigby, "Mitigating turnover with code review recommendation: Balancing expertise, workload, and knowledge distribution," in *Proceedings of the ACM/IEEE 42nd international conference on software engineering*, pp. 1183–1195, 2020.

[11] M. Chouchen, A. Ouni, M. W. Mkaouer, R. G. Kula, and K. Inoue, "Whoreview: A multi-objective search-based approach for code reviewers recommendation in modern code review," *Applied Soft Computing*, vol. 100, p. 106908, 2021.

[12] D. Kong, Q. Chen, L. Bao, C. Sun, X. Xia, and S. Li, "Recommending code reviewers for proprietary software projects: A large scale study," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 630–640, IEEE, 2022.

[13] P. Pandya and S. Tiwari, "Corms: a github and gerrit based hybrid code reviewer recommendation approach for modern code review," in *Proceedings of the 30th ACM joint European software engineering conference and symposium on the foundations of software engineering*, pp. 546–557, 2022.

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

[15] Y. Hong, C. Tantithamthavorn, P. Thongtanunam, and A. Aleti, "Commentfinder: a simpler, faster, more accurate code review comments recommendation," in *Proceedings of the 30th ACM joint European software engineering conference and symposium on the foundations of software engineering*, pp. 507–519, 2022.

[16] O. Shuvo, P. Mahbub, and M. M. Rahman, "Recommending code reviews leveraging code changes with structured information retrieval," in *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 194–206, IEEE, 2023.

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

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

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

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

[21] A. Eliseeva, Y. Sokolov, E. Bogomolov, Y. Golubev, D. Dig, and T. Bryksin, "From commit message generation to history-aware commit message completion," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 723–735, IEEE, 2023.

[22] Y. Sun, D. Wu, Y. Xue, H. Liu, H. Wang, Z. Xu, X. Xie, and Y. Liu, "Gptscan: Detecting logic vulnerabilities in smart contracts by combining gpt with program analysis," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pp. 1–13, 2024.

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

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

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

[26] R. Tufano, O. Dabić, A. Mastropaolo, M. Ciniselli, and G. Bavota, "Code review automation: strengths and weaknesses of the state of the art," *IEEE Transactions on Software Engineering*, 2024.

[27] D. W. Fiske and L. Fogg, "But the reviewers are making different criticisms of my paper! diversity and uniqueness in reviewer comments.," 1992.

[28] Y. Zhang, Y. Zhang, Z. Sun, Y. Jiang, and H. Liu, "Laura: Enhancing code review generation with context-enriched retrieval-augmented llm." Figshare, 2025. https://doi.org/10.6084/m9.figshare.27367194.

[29] Y. Tian, Y. Zhang, K.-J. Stol, L. Jiang, and H. Liu, "What makes a good commit message?," in *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, (New York, NY, USA), p. 2389–2401, Association for Computing Machinery, 2022.

[30] Tree-sitter, "Introduction - tree-sitter." Website, 2025. https://tree-sitter.github.io/tree-sitter/.

[31] H. Wang, X. Xia, D. Lo, Q. He, X. Wang, and J. Grundy, "Context-aware retrieval-based deep commit message generation," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 4, pp. 1–30, 2021.

[32] L. Zhang, H. Zhang, C. Wang, and P. Liang, "Rag-enhanced commit message generation," *arXiv preprint arXiv:2406.05514*, 2024.

[33] A. Ochiai, "Zoogeographic studies on the soleoid fishes found in japan and its neighbouring regions," *Bulletin of Japanese Society of Scientific Fisheries*, vol. 22, pp. 526–530, 1957.

[34] Y. Wang, H. Le, A. D. Gotmare, N. D. Bui, J. Li, and S. C. Hoi, "Codet5+: Open code large language models for code understanding and generation," *arXiv preprint arXiv:2305.07922*, 2023.

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

[36] Y. Kartal, E. K. Akdeniz, and K. Özkan, "Automating modern code review processes with code similarity measurement," *Information and Software Technology*, vol. 173, p. 107490, 2024.

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

[38] J. Bae, S. Kwon, and S. Myeong, "Enhancing software code vulnerability detection using gpt-4o and claude-3.5 sonnet: A study on prompt engineering techniques," *Electronics*, vol. 13, no. 13, p. 2657, 2024.

[39] T. Ahmed, K. S. Pai, P. Devanbu, and E. Barr, "Automatic semantic augmentation of language model prompts (for code summarization)," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pp. 1–13, 2024.

[40] C. Pornprasit and C. Tantithamthavorn, "Fine-tuning and prompt engineering for large language models-based code review automation," *Information and Software Technology*, vol. 175, p. 107523, 2024.

[41] X. Amatriain, "Prompt design and engineering: Introduction and advanced methods," *arXiv preprint arXiv:2401.14423*, 2024.

[42] GitHub, "Github graphql api documentation - github docs." Website, 2025. https://docs.github.com/en/graphql.

[43] GitHub, "Comparing commits - github docs." Website, 2025. https://docs.github.com/en/pull-requests/committing-changes-to-your-project/viewing-and-comparing-commits/comparing-commits.

[44] A. Bosu, M. Greiler, and C. Bird, "Characteristics of useful code reviews: An empirical study at microsoft," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pp. 146–156, IEEE, 2015.

[45] G. Rong, Y. Yu, Y. Zhang, H. Zhang, H. Shen, D. Shao, H. Kuang, M. Wang, Z. Wei, Y. Xu, *et al.*, "Distilling quality enhancing comments from code reviews to underpin reviewer recommendation," *IEEE Transactions on Software Engineering*, vol. 50, no. 7, pp. 1658–1674, 2024.

[46] M. Golzadeh, A. Decan, D. Legay, and T. Mens, "A ground-truth dataset and classification model for detecting bots in github issue and pr comments," *Journal of Systems and Software*, vol. 175, p. 110911, 2021.

[47] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen, "Boa: A language and infrastructure for analyzing ultra-large-scale software repositories," in *2013 35th International Conference on Software Engineering (ICSE)*, pp. 422–431, IEEE, 2013.

[48] M. Schall, T. Czinczoll, and G. De Melo, "Commitbench: A benchmark for commit message generation," in *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 728–739, IEEE, 2024.

[49] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and psychological measurement*, vol. 20, no. 1, pp. 37–46, 1960.

[50] E. Reiter, "A structured review of the validity of bleu," *Computational Linguistics*, vol. 44, no. 3, pp. 393–401, 2018.

[51] J. Lu, X. Li, Z. Hua, L. Yu, S. Cheng, L. Yang, F. Zhang, and C. Zuo, "Deepcrceval: Revisiting the evaluation of code review comment generation," in *International Conference on Fundamental Approaches to Software Engineering*, pp. 43–64, Springer, 2025.

[52] A. Joshi, S. Kale, S. Chandel, and D. K. Pal, "Likert scale: Explored and explained," *British journal of applied science & technology*, vol. 7, no. 4, p. 396, 2015.

[53] S. Mahajan, N. Abolhassani, and M. R. Prasad, "Recommending stack overflow posts for fixing runtime exceptions using failure scenario matching," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1052–1064, 2020.

[54] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, vol. 21, no. 140, pp. 1–67, 2020.

[55] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.

[56] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, *et al.*, "Emergent abilities of large language models," *arXiv preprint arXiv:2206.07682*, 2022.

[57] OpenAI, "Hello gpt-4o — openai." Website, 2024. https://openai.com/index/hello-gpt-4o/.

[58] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, *et al.*, "Deepseek-v3 technical report," *arXiv preprint arXiv:2412.19437*, 2024.

[59] X. Zhou, T. Zhang, and D. Lo, "Large language model for vulnerability detection: Emerging results and future directions," in *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*, pp. 47–51, 2024.

[60] Z. Ságodi, G. Antal, B. Bogenfürst, M. Isztin, P. Hegedűs, and R. Ferenc, "Reality check: Assessing gpt-4 in fixing real-world software vulnerabilities," in *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*, pp. 252–261, 2024.

[61] Y. Wu, Y. Li, and S. Yu, "Commit message generation via chatgpt: How far are we?," in *Proceedings of the 2024 IEEE/ACM First International Conference on AI Foundation Models and Software Engineering*, pp. 124–129, 2024.

[62] J. Li, D. Faragó, C. Petrov, and I. Ahmed, "Only diff is not enough: Generating commit messages leveraging reasoning and action of large language model," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 745–766, 2024.

[63] J. Lu, Z. Li, C. Shen, L. Yang, and C. Zuo, "Exploring the impact of code review factors on the code review comment generation," *Automated Software Engineering*, vol. 31, no. 2, p. 71, 2024.

[64] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang, "Retrieval-augmented generation for large language models: A survey," *arXiv preprint arXiv:2312.10997*, 2023.

[65] J. White, S. Hays, Q. Fu, J. Spencer-Smith, and D. C. Schmidt, "Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design," in *Generative AI for Effective Software Development*, pp. 71–108, Springer, 2024.

[66] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.

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