CodeFuse-CR-Bench: A Comprehensiveness-aware Benchmark for End-to-End Code Review Evaluation in Python Projects
CodeFuse-CR-Bench：一个用于Python项目端到端代码审查评估的全面性感知基准

Abstract
摘要

Automated code review (CR) is a key application for Large Language Models (LLMs), but progress is hampered by a “reality gap”: existing benchmarks evaluate models on isolated sub-tasks using simplified, context-poor data.
自动化代码审查（CR）是大语言模型（LLMs）的一个关键应用，但其进展受到“现实差距”的阻碍：现有的基准测试使用简化的、缺乏上下文的数据在孤立的子任务上评估模型。

This fails to reflect the holistic context-rich nature of real-world CR.
这未能反映现实世界代码审查整体且富含上下文的本质。

To bridge this gap, we introduce CodeFuse-CR-Bench, the first comprehensiveness-aware benchmark for repository-level CR evaluation.
为了弥合这一差距，我们推出了 CodeFuse-CR-Bench，这是首个用于仓库级代码审查评估的全面性感知基准。

CodeFuse-CR-Bench comprises 601 high-quality instances from 70 Python projects covering nine Pull-Request (PR) problem domains, where each instance provides rich, multi-faceted context including the associated issue, PR details, and repository state, enabling end-to-end evaluation.
CodeFuse-CR-Bench 包含来自 70 个 Python 项目的 601 个高质量实例，涵盖九个拉取请求（PR）问题领域，其中每个实例都提供了丰富的、多层面的上下文，包括相关议题（Issue）、PR 详情和仓库状态，从而实现了端到端评估。

Beyond superficial metrics, we also propose a novel evaluation framework that combines rule-based checks for location and syntax with model-based judgments of review quality.
除了肤浅的指标外，我们还提出了一个新的评估框架，结合了针对位置和语法的基于规则的检查，以及基于模型的审查质量判断。

We present the first large-scale assessment of state-of-the-art LLMs on this comprehensive CR task.
我们展示了最先进的大语言模型（LLMs）在此综合代码审查任务上的首次大规模评估。

Our results establish crucial baselines and reveal that (1) no single LLM dominates all aspects of CR; (2) Gemini 2.5 Pro achieves the highest comprehensive performance; and (3) different LLMs exhibit varying robustness to redundant context.
我们的结果建立了关键基线，并揭示了：（1）没有单一的 LLM 主导代码审查的所有方面；（2）Gemini 2.5 Pro 实现了最高的综合性能；以及（3）不同的 LLM 对冗余上下文表现出不同的鲁棒性。

These findings highlight the necessity of holistic, multi-dimensional evaluation and provide actionable insights for advancing truly intelligent yet practical CR assistants.
这些发现凸显了整体、多维评估的必要性，并为推进真正智能且实用的代码审查助手提供了可操作的见解。

1 Introduction
1 引言

Code Review (CR) is a core practice in modern software development that aims to improve code quality and identify defects through collaborative inspection (Bacchelli & Bird, 2013).
代码审查（CR）是现代软件开发中的核心实践，旨在通过协作检查来提高代码质量并识别缺陷 (Bacchelli & Bird, 2013)。

As Large Language Models (LLMs) increasingly automate complex software engineering tasks, their application to CR holds immense promise for improving software quality and developer productivity.
随着大语言模型（LLMs）日益自动化复杂的软件工程任务，它们在代码审查中的应用对于提高软件质量和开发人员生产力有着巨大的前景。

However, the development of such sophisticated tools is fundamentally constrained by how we measure their performance.
然而，此类复杂工具的开发从根本上受限于我们如何衡量其性能。

Current benchmarks, while valuable, evaluate models in isolated, decontextualized settings, creating a significant and growing “reality gap” between measured performance and real-world efficacy.
当前的基准测试虽然有价值，但在孤立、脱离上下文的环境中评估模型，从而在衡量出的性能与现实世界的效能之间造成了显著且日益扩大的“现实差距”。

The core of this problem lies in a failure to capture the comprehensiveness of the CR process.
这个问题的核心在于未能捕捉到代码审查过程的全面性。

Real-world CR is not a simple text-matching exercise; instead it is a holistic reasoning task that requires a deep understanding of context.
现实世界的代码审查不是简单的文本匹配练习；相反，它是一项需要深刻理解上下文的整体推理任务。

This disconnection manifests in three critical limitations of existing automated CR research, as demonstrated in Table 1:
这种脱节体现在现有自动化代码审查研究的三个关键局限性上，如表 1 所示：

1. Task Fragmentation: The cognitive process of a human reviewer—understanding the initial problem, locating potential issues in a code change, and formulating a coherent review—is often broken down into isolated sub-tasks like comment generation or code refinement.
2. 任务碎片化：人类审查者的认知过程——理解初始问题、定位代码变更中的潜在问题以及制定连贯的审查——通常被分解为孤立的子任务，如评论生成或代码完善。

This fragmentation prevents the evaluation of end-to-end reasoning, a crucial capability for a truly useful automated reviewer.
这种碎片化阻碍了对端到端推理的评估，而这是真正有用的自动化审查者的一项关键能力。

2. Context Poverty: Existing benchmarks typically provide only small, self-contained code snippets and strip away the rich context that is essential for meaningful review, such as the Pull Request (PR) description, the linked issue report, and the broader repository structure.
3. 上下文匮乏：现有的基准测试通常仅提供小的、自包含的代码片段，并剥离了对有意义的审查至关重要的丰富上下文，例如拉取请求（PR）描述、链接的问题报告以及更广泛的仓库结构。

Without this context, a model cannot grasp the intent behind a change, making its review superficial.
没有这些上下文，模型就无法掌握变更背后的意图，从而使其审查变得肤浅。

3. Evaluation Narrowness: Evaluation metrics are often inherited from natural language processing (NLP) tasks (e.g., BLEU).
4. 评估狭隘性：评估指标通常继承自自然语言处理（NLP）任务（例如 BLEU）。

These metrics reward superficial textual similarity but fail to assess the substantive quality of a review.
这些指标奖励肤浅的文本相似性，但未能评估审查的实质性质量。

They cannot distinguish a technically insightful suggestion from a syntactically similar but incorrect one, nor can they verify if a review comment is even placed at the correct location in the code.
它们无法区分技术上有见地的建议与语法相似但不正确的建议，也无法验证审查评论是否甚至被放置在代码的正确位置。

In contrast, as shown in Figure 1, a typical CR process is a holistic reasoning task.
相反，如图 1 所示，典型的代码审查过程是一项整体推理任务。

The code reviewer receives a CR request and the corresponding PR information, which includes PR-related information such as the PR description and associated issue data.
代码审查者接收代码审查请求和相应的 PR 信息，其中包括 PR 相关信息，如 PR 描述和相关的议题数据。

Based on this rich context, the reviewer performs the review by writing comments and revision suggestions, which constitute the CR-related information.
基于这一丰富的上下文，审查者通过编写构成代码审查相关信息的评论和修订建议来执行审查。

To bridge the gap between this reality and current evaluation methods, we introduce CodeFuse-CR-Bench, a Comprehensiveness-Aware benchmark for Repository-level Evaluation of code review.
为了弥合这一现实与当前评估方法之间的差距，我们推出了 CodeFuse-CR-Bench，这是一个用于代码审查仓库级评估的全面性感知基准。

CodeFuse-CR-Bench is designed from the ground up to model this full CR workflow.
CodeFuse-CR-Bench 从设计之初就是为了模拟这一完整的代码审查工作流程。

It comprises 601 high-quality instances curated from 70 real-world Python open-source projects.
它包含从 70 个现实世界的 Python 开源项目中精选出的 601 个高质量实例。

Each instance is a rich, self-contained snapshot of a real review task, encompassing basic information, PR-related information, CR-related information, and repository-level context.
每个实例都是一个真实审查任务的丰富、自包含的快照，涵盖了基本信息、PR 相关信息、代码审查相关信息以及仓库级上下文。

This multi-faceted context enables models to engage in the kind of holistic reasoning that developers perform daily.
这种多层面的上下文使模型能够进行开发人员每天执行的那种整体推理。

Furthermore, to move beyond narrow, syntax-focused metrics, we designed a comprehensive evaluation framework for CodeFuse-CR-Bench.
此外，为了超越狭隘的、关注语法的指标，我们为 CodeFuse-CR-Bench 设计了一个综合评估框架。

This framework integrates both fine-grained rule-based metrics (to assess location accuracy and semantic similarity) and holistic model-based evaluation (which uses the reward model and advanced LLMs-as-judges to score the overall quality, relevance, and correctness of a review).
该框架集成了细粒度的基于规则的指标（用于评估位置准确性和语义相似性）和整体的基于模型的评估（使用奖励模型和先进的 LLM 作为裁判来对审查的整体质量、相关性和正确性进行评分）。

Using this benchmark and framework, we conduct the first large-scale assessment of state-of-the-art LLMs on the comprehensive CR task.
利用该基准和框架，我们就综合代码审查任务对最先进的 LLM 进行了首次大规模评估。

Our results establish crucial baselines and reveal the current capabilities and limitations of LLMs when faced with the complexities of real-world code review.
我们的结果建立了关键基线，并揭示了 LLM 在面对现实世界代码审查的复杂性时当前的能力和局限性。

In summary, this paper makes the following contributions:
总而言之，本文做出了以下贡献：

• We identify and characterize the "comprehensiveness gap" in current CR research, highlighting how task fragmentation, context poverty, and narrow evaluation metrics hinder progress.
• 我们识别并刻画了当前代码审查研究中的“全面性差距”，强调了任务碎片化、上下文匮乏和狭隘的评估指标如何阻碍进展。

• We introduce CodeFuse-CR-Bench, the first comprehensiveness-aware CR benchmark that provides rich, repository-level context to enable the evaluation of end-to-end CR tasks across nine distinct problem domains.
• 我们推出了 CodeFuse-CR-Bench，这是首个全面性感知的代码审查基准，提供丰富的、仓库级的上下文，以实现跨越九个不同问题领域的端到端代码审查任务的评估。

• We propose a novel, multi-faceted evaluation framework that combines rule-based precision with model-based quality assessment to provide a more holistic measure of CR performance.
• 我们提出了一个新的、多层面的评估框架，将基于规则的精确度与基于模型的质量评估相结合，以提供对代码审查性能更全面的衡量。

• We conduct an extensive empirical study on multiple state-of-the-art LLMs, providing the first robust baseline for comprehensive CR and offering insights into future research directions.
• 我们对多个最先进的 LLM 进行了广泛的实证研究，为综合代码审查提供了首个稳健的基线，并为未来的研究方向提供了见解。

---

2 Related Work
2 相关工作

2.1 CR-related Tasks

2.1 代码审查（CR）相关任务

Previously, the vast majority of benchmarks and approaches were constructed to improve CR performance (Jiang et al., 2025).

以前，绝大多数基准测试和方法都是为了提高代码审查（CR）性能而构建的 (Jiang et al., 2025)。

Tufano et al. (Tufano et al., 2021) proposed Trans-Review, which adopted deep learning models to partially automate specific CR tasks.

Tufano 等人 (Tufano et al., 2021) 提出了 Trans-Review，该方法采用深度学习模型来部分自动化特定的代码审查任务。

They trained two models to implement two CR sub-tasks: (i) code revision before review and (ii) code revision after review.

他们训练了两个模型来实现两个代码审查子任务：(i) 审查前的代码修订和 (ii) 审查后的代码修订。

Based on this prior work, they (Tufano et al., 2022) updated the models from deep learning models to the pre-training model T5 and named it T5-Review.

基于这项先前的工作，他们 (Tufano et al., 2022) 将模型从深度学习模型更新为预训练模型 T5，并将其命名为 T5-Review。

The results demonstrated that T5-Review can outperform previous deep learning models for automating CR tasks.

结果表明，T5-Review 在自动化代码审查任务方面优于以前的深度学习模型。

Thongtanunam et al. (Thongtanunam et al., 2022) proposed AutoTransform, which leverages a Byte-Pair Encoding (BPE) approach to handle new tokens and a Transformer-based Neural Machine Translation architecture to handle long sequences.

Thongtanunam 等人 (Thongtanunam et al., 2022) 提出了 AutoTransform，它利用字节对编码 (BPE) 方法来处理新标记 (token)，并利用基于 Transformer 的神经机器翻译架构来处理长序列。

It can be used in the task of code revision before review.

它可以用于审查前的代码修订任务。

Zhou et al. (Zhou et al., 2023) evaluated the above three automatic CR tools and pre-trained models for three processes in CR: code revision before review, review comment generation, and code revision after review.

Zhou 等人 (Zhou et al., 2023) 评估了上述三个自动代码审查工具和预训练模型在代码审查中的三个过程：审查前的代码修订、审查评论生成和审查后的代码修订。

The results show that a general-purpose pre-trained model CodeT5 can outperform other models in most cases.

结果表明，通用预训练模型 CodeT5 在大多数情况下都能优于其他模型。

Li et al. (Li et al., 2022a) proposed a review comments generator with pre-training models, which is called AUGER.

Li 等人 (Li et al., 2022a) 提出了一种基于预训练模型的审查评论生成器，称为 AUGER。

They collected empirical review data from 11 notable Java projects and constructed a dataset of 10,882 code changes to evaluate the performance of the proposed approach.
他们从 11 个著名的 Java 项目中收集了经验审查数据，并构建了一个包含 10,882 个代码变更的数据集，以评估所提出方法的性能。

Li et al. (Li et al., 2022b) proposed a pre-trained model that utilized four pre-training tasks tailored specifically for the CR scenario, named CodeReviewer.
Li 等人 (Li et al., 2022b) 提出了一种利用专门针对代码审查场景定制的四个预训练任务的预训练模型，名为 CodeReviewer。

They focused on three key tasks related to CR activities, including code change quality estimation, review comment generation, and code refinement to evaluate the model.
他们专注于与代码审查活动相关的三个关键任务，包括代码变更质量估计、审查评论生成和代码完善，以评估模型。

They also constructed a high-quality benchmark dataset based on our collected data for these three tasks and conducted comprehensive experiments on it.
他们还基于收集到的这三个任务的数据构建了一个高质量的基准数据集，并对其进行了全面的实验。

The experiments demonstrated the SOTA results.
实验展示了最先进 (SOTA) 的结果。

Additionally, some LLM-based CR approaches were proposed.
此外，还提出了一些基于大语言模型（LLM）的代码审查方法。

Guo et al. (Guo et al., 2024) conducted the first empirical study to explore the capabilities of ChatGPT in CR tasks, specifically focusing on automated code revision after reviews.
Guo 等人 (Guo et al., 2024) 进行了首次实证研究，以探索 ChatGPT 在代码审查任务中的能力，特别关注审查后的自动代码修订。

They constructed a new CR dataset with high quality based on the existing benchmark CodeReview (Li et al., 2022b).
他们基于现有的基准 CodeReview (Li et al., 2022b) 构建了一个高质量的新代码审查数据集。

A SOTA CR tool (Li et al., 2022b) was selected as a baseline.
选择了一个 SOTA 代码审查工具 (Li et al., 2022b) 作为基线。

The research study provided insights into the potential of ChatGPT in automating the CR process.
该研究为 ChatGPT 在自动化代码审查过程中的潜力提供了见解。

As an important upstream sub-task of CR, issue-solving has also been extensively studied.
作为代码审查的一个重要上游子任务，议题解决 (issue-solving) 也得到了广泛研究。

The content of issue-solving can provide more complete contexts for CR.
议题解决的内容可以为代码审查提供更完整的上下文。

Some issue-solving benchmarks had been proposed.
已经提出了一些议题解决基准。

Jimenez et al. (Jimenez et al., 2024) proposed a benchmark, SWE-Bench, that evaluates LLMs in resolving an issue (typically a bug report or a feature request) submitted to popular Python GitHub repositories.
Jimenez 等人 (Jimenez et al., 2024) 提出了一个基准 SWE-Bench，该基准评估 LLM 解决提交给流行 Python GitHub 仓库的议题（通常是错误报告或功能请求）的能力。

Zan et al. (Zan et al., 2025) enlarged the SWE-Bench dataset by adding other programming languages' issue-solving and named it Multi-SWE-bench.
Zan 等人 (Zan et al., 2025) 通过增加其他编程语言的议题解决扩展了 SWE-Bench 数据集，并将其命名为 Multi-SWE-bench。

Hu et al. (Hu et al., 2024) proposed FAUN-Eval, a benchmark specifically designed to evaluate the fine-grained issue-solving capabilities of LLMs.
Hu 等人 (Hu et al., 2024) 提出了 FAUN-Eval，这是一个专门设计用于评估 LLM 细粒度议题解决能力的基准。

It can be used to systematically assess LLMs across three distinct tasks: Question-Answer (QA), fault localization, and code editing.
它可用于系统地评估 LLM 在三个不同任务中的表现：问答 (QA)、故障定位和代码编辑。

However, either the above approaches or the benchmarks focus on the sub-tasks of CR.
然而，无论是上述方法还是基准都侧重于代码审查的子任务。

The design and construction are comprehensiveness-unaware.
其设计和构建都没有考虑到全面性。

To fill this gap, we construct a comprehensiveness-aware benchmark and evaluation metrics for CR.
为了填补这一空白，我们构建了一个全面性感知的基准和代码审查评估指标。

We conduct an empirical study on some SOTA LLMs based on this benchmark to evaluate their performance in comprehensive CR tasks.
我们基于此基准对一些 SOTA LLM 进行了实证研究，以评估它们在综合代码审查任务中的表现。

2.2 LLM for Software Engineering Tasks
2.2 软件工程任务中的 LLM

Recently, LLMs have demonstrated revolutionary performance improvements in almost all software-engineering-related tasks.
最近，LLM 在几乎所有软件工程相关任务中都表现出了革命性的性能提升。

Regarding general LLMs, the GPT series (Liang et al., 2024), Claude, Gemini, and others had demonstrated powerful code generation, summarization, and program repair through training on large corpora containing code (Feng et al., 2024; Cao et al., 2024; Wang et al., 2025b; Zhao et al., 2023; Fan et al., 2023).
关于通用 LLM，GPT 系列 (Liang et al., 2024)、Claude、Gemini 等通过在包含代码的大型语料库上进行训练，展示了强大的代码生成、摘要和程序修复能力 (Feng et al., 2024; Cao et al., 2024; Wang et al., 2025b; Zhao et al., 2023; Fan et al., 2023)。

Specifically, a systematic comparative analysis was conducted on three advanced LLMs, including ChatGPT (O1), DeepSeek (R1), and Gemini (2.0 Flash thinking), for Python code generation, evaluating their performance in correctness, code quality, and computational efficiency.
具体而言，对三个先进的 LLM，包括 ChatGPT (O1)、DeepSeek (R1) 和 Gemini (2.0 Flash thinking) 进行了系统的比较分析，针对 Python 代码生成，评估了它们在正确性、代码质量和计算效率方面的表现。

Each of the three LLMs has its own strengths and limitations.
这三个 LLM 各有其优势和局限性。

Their findings underscored the inherent trade-offs between efficiency, accuracy, and quality in AI-generated code.
他们的发现强调了 AI 生成代码在效率、准确性和质量之间固有的权衡。

Sobo et al. (Sobo et al., 2025) investigated the effectiveness of LLMs in generating code for Human-Robot Interaction applications.
Sobo 等人 (Sobo et al., 2025) 调查了 LLM 在为新人机交互应用生成代码方面的有效性。

They compared the performance among ChatGPT 3.5, Gemini 1.5 Pro, and Claude 3.5 Sonnet.
他们比较了 ChatGPT 3.5、Gemini 1.5 Pro 和 Claude 3.5 Sonnet 的性能。

The study highlighted the rapid advancement in LLM capabilities for specialized programming tasks while also identifying persistent challenges in spatial reasoning and adherence to specific constraints.
该研究强调了 LLM 在专业编程任务能力方面的快速进步，同时也指出了在空间推理和遵守特定约束方面持续存在的挑战。

Zhang et al. (Zhang et al., 2024) evaluated the capability of advanced LLMs, including ChatGPT-4 and Claude, in fixing memory corruption vulnerabilities in real-world C/C++ code.
Zhang 等人 (Zhang et al., 2024) 评估了先进 LLM（包括 ChatGPT-4 和 Claude）修复现实世界 C/C++ 代码中内存损坏漏洞的能力。

They analyzed both the strengths and limitations of LLMs in automated program repair on genuine code.
他们分析了 LLM 在针对真实代码的自动程序修复中的优势和局限性。

Sun et al. (Sun et al., 2025) conducted the examination of prevalent automated evaluation methods for assessing the quality of summaries generated by LLMs and found that the results of the GPT-4 evaluation method are most closely aligned with human evaluation.
Sun 等人 (Sun et al., 2025) 对评估 LLM 生成摘要质量的流行自动评估方法进行了检查，发现 GPT-4 评估方法的结果与人类评估最为一致。

They also discussed the limitations of LLMs in generating summarization in logic programming languages.
他们还讨论了 LLM 在生成逻辑编程语言摘要方面的局限性。

All the software-engineering-related tasks mentioned above are highly relevant to CR in technical terms.
上述所有与软件工程相关的任务在技术上都与代码审查高度相关。

Therefore, it is possible for LLMs to perform comprehensive CR tasks.
因此，LLM 有可能执行综合代码审查任务。

In our paper, we select several representative LLMs and evaluate their comprehensive CR capabilities using our constructed benchmark and designed evaluation metrics.
在本文中，我们选择了几个具有代表性的 LLM，并使用我们构建的基准和设计的评估指标来评估它们的综合代码审查能力。

---

3 CodeFuse-CR-Bench Benchmark
3 CodeFuse-CR-Bench 基准

Having established the "comprehensiveness gap" in current CR research, this section details the design and construction of CodeFuse-CR-Bench.
在确定了当前代码审查研究中的“全面性差距”之后，本节将详细介绍 CodeFuse-CR-Bench 的设计和构建。

We present the benchmark overview, benchmark construction pipeline, and benchmark characteristics in Section 3.1, Section 3.2, and Section 3.3, respectively.
我们分别在 3.1 节、3.2 节和 3.3 节中介绍基准概览、基准构建流程和基准特性。

3.1 Benchmark Overview
3.1 基准概览

CodeFuse-CR-Bench comprises 601 Python CR task instances, each carefully curated to reflect real-world development scenarios.
CodeFuse-CR-Bench 包含 601 个 Python 代码审查任务实例，每个实例都经过精心策划以反映现实世界的开发场景。

Fig. 2 illustrates an overview of a typical CR task instance.
图 2 展示了一个典型代码审查任务实例的概览。

Each instance is associated with 22 structured fields, as summarized in Table 2.
每个实例都与 22 个结构化字段相关联，如表 2 所总结。

These instances can be systematically categorized into the following four types to support comprehensive CR:
为了支持全面的代码审查，这些实例可以系统地分为以下四种类型：

• Basic information: Instance ID, Owner/Repo, and Language, providing fundamental identification of the task.
• 基本信息：实例 ID、所有者/仓库和语言，提供任务的基本标识。

• PR-related information: Encompasses metadata critical to understanding the change intent and context, including Pull No., Title, Created at, Base Commit, Body, Issue Problem Statement, Hint Text, Resolved issue No., Commit Patch to Review, Head Commit, Head Commit Message, Problem Domain, and Difficulty.
• PR 相关信息：包含对理解变更意图和上下文至关重要的元数据，包括拉取请求编号 (Pull No.)、标题、创建时间、基础提交 (Base Commit)、正文、议题问题陈述、提示文本、解决的议题编号、待审查的提交补丁、头部提交 (Head Commit)、头部提交信息、问题领域和难度。

• CR-related information: Captures the review process itself, including Review Comment Text, Diff Hunk, Review Path, and Review Effort, enabling analysis of reviewer behavior and feedback quality.
• 代码审查相关信息：捕获审查过程本身，包括审查评论文本、差异块 (Diff Hunk)、审查路径和审查工作量，从而能够分析审查者的行为和反馈质量。

• Repository-Level Context Information: Provides broader project-level context necessary for cross-file reasoning and impact analysis, including Merge Commit Patch and Merge Commit.
• 仓库级上下文信息：提供跨文件推理和影响分析所需的更广泛的项目级上下文，包括合并提交补丁 (Merge Commit Patch) 和合并提交 (Merge Commit)。

3.2 Benchmark Construction Pipeline
3.2 基准构建流程

As depicted in Fig. 3, the construction pipeline of CodeFuse-CR-Bench consists of five steps, namely, (1) Repository Selection; (2) PR Crawling and Attribute-based Filtering; (3) PR Classification; (4) Feature Labeling; and (5) Manual Selection & Annotation.
如图 3 所示，CodeFuse-CR-Bench 的构建流程包含五个步骤，即 (1) 仓库选择；(2) PR 爬取与基于属性的过滤；(3) PR 分类；(4) 特征标注；以及 (5) 人工选择与标注。

We next briefly elaborate on them.
接下来我们简要阐述这些步骤。

3.2.1 Repository Selection
3.2.1 仓库选择

To make CodeFuse-CR-Bench more representative, we adopt a strict approach to selecting CR task instances from various open-source repositories.
为了使 CodeFuse-CR-Bench 更具代表性，我们采用严格的方法从各种开源仓库中选择代码审查任务实例。

We focus on Python as it is one of the most popular languages on GitHub, possessing a mature and diverse open-source ecosystem.
我们专注于 Python，因为它是 GitHub 上最流行的语言之一，拥有成熟且多样化的开源生态系统。

This ensures a rich source of projects with standardized and high-quality CR practices, which is essential for our benchmark’s validity.
这确保了拥有标准化和高质量代码审查实践的丰富项目来源，这对于我们基准的有效性至关重要。

Specifically, we first search for the top 1,000 starred Python repositories on GitHub, a Git-based code hosting and collaboration platform.
具体来说，我们首先在 GitHub（一个基于 Git 的代码托管和协作平台）上搜索星标数排名前 1,000 的 Python 仓库。

A higher star rating indicates that the repository has better popularity.
较高的星标评级表明该仓库拥有更好的受欢迎程度。

To ensure the quality of the repository and acquire a large pool of potential CR data, we further sort the projects by the number of PRs and filter out projects with less than 1,500 PRs, referring to Li's study (Li et al., 2022b).
为了确保仓库的质量并获取大量潜在的代码审查数据，我们参考 Li 的研究 (Li et al., 2022b)，进一步按 PR 数量对项目进行排序，并过滤掉 PR 数量少于 1,500 个的项目。

Additionally, we only keep the repositories that were maintained in the last year (from 2024-08-15 to 2025-08-15), i.e., the repositories that had commit or PR records over the past year.
此外，我们只保留在过去一年（从 2024-08-15 到 2025-08-15）内维护过的仓库，即在过去一年中有提交或 PR 记录的仓库。

The process mentioned above aims to keep only active projects and remove repositories that are forked from other repositories, as the PR number is not inherited.
上述过程旨在仅保留活跃项目，并移除从其他仓库分叉 (fork) 出来的仓库，因为 PR 编号是不会继承的。

This process yielded 230 Python projects that meet our criteria for activity and maturity.
该过程产生了 230 个符合我们活跃度和成熟度标准的 Python 项目。

3.2.2 PR Crawling & Attribute-based Filtering
3.2.2 PR 爬取与基于属性的过滤

In this step, we aim to crawl high-quality PR data.
在这一步中，我们的目标是爬取高质量的 PR 数据。

To ensure the collected PR data contains high-quality CR, we filter it based on the following attributes:
为了确保收集到的 PR 数据包含高质量的代码审查，我们基于以下属性对其进行过滤：

(1) We only include PRs that have at least 1 closing issue reference;
(1) 我们仅包含至少有一个关闭议题 (closing issue) 引用的 PR；

(2) We only collect the PR that is merged into the main branch.
(2) 我们仅收集已合并到主分支的 PR。

A "Merged" status indicates that the code changes associated with the PR were accepted and incorporated into its parent repository.
"已合并"状态表明与该 PR 相关的代码变更已被接受并合并到其父仓库中。

We conduct this process by using GitHub GraphQL API.
我们使用 GitHub GraphQL API 执行此过程。

The PRs selected based on these attributes undergo rigorous CR, thereby ensuring that the CR data selected under these PRs is of high quality (Zheng et al., 2025).
基于这些属性选择的 PR 经历了严格的代码审查，从而确保在这些 PR 下选择的代码审查数据具有高质量 (Zheng et al., 2025)。

3.2.3 PR Classification
3.2.3 PR 分类

Since each PR instance contains multiple commits, our goal in this step is to identify high-quality commits within each PR and extract their corresponding high-quality CR-related information.
由于每个 PR 实例包含多个提交 (commit)，我们在这一步的目标是识别每个 PR 中的高质量提交，并提取其对应的高质量代码审查相关信息。

Firstly, we design several heuristic rules to evaluate the commits and give an evaluation score.
首先，我们设计了几条启发式规则来评估提交并给出一个评估分数。

The details of heuristic rules are shown in Table 3.
启发式规则的详细信息如表 3 所示。

They are categorized into three types: high-impact rules, medium-impact rules, and low-impact rules.
它们分为三类：高影响规则、中等影响规则和低影响规则。

A higher impact level signifies greater importance and carries more weight in the scoring (high weight: 3.0, medium weight: 2.0, low weight: 1.0).
较高的影响级别意味着更重要，并在评分中占有更大的权重（高权重：3.0，中等权重：2.0，低权重：1.0）。

We assign the highest weight to rules indicating direct, actionable review feedback (e.g., has_resolved_review_comments), as these are the strongest indicators of a meaningful CR process.
我们将最高权重分配给指示直接、可操作审查反馈的规则（例如 has_resolved_review_comments），因为这些是有意义的代码审查过程的最强指标。

Rules related to best practices and commit clarity receive medium weight, while secondary indicators like issue references have a low weight.
与最佳实践和提交清晰度相关的规则获得中等权重，而像议题引用这样的次要指标权重较低。

Description presents the scoring details of each heuristic rule.
“描述”一栏展示了每条启发式规则的评分细节。

By implementing a weighted sum scoring, we obtain an overall score for each commit.
通过实施加权求和评分，我们获得每个提交的总体得分。

Based on the overall score, we rank the commits in each PR and extract the target commit with the highest score.
基于总体得分，我们对每个 PR 中的提交进行排名，并提取得分最高的目标提交。

After getting the target commit, high-quality CR task instances can be extracted from the target commit.
在获得目标提交后，可以从目标提交中提取高质量的代码审查任务实例。

Specifically, we extract and review comments based on whether referenced lines were actually changed in the merged commit, or the review thread was resolved, outdated, or collapsed.
具体来说，我们提取并审查评论，依据是引用的行是否在合并提交中实际发生了更改，或者审查线程是否已解决、过时或折叠。

Review comments containing the aforementioned attributes will have their corresponding CRs labeled as high-quality CR task instances.
包含上述属性的审查评论，其对应的代码审查将被标记为高质量代码审查任务实例。

3.2.4 Feature Labeling
3.2.4 特征标注

In this step, we classify the three attributes: problem domain, difficulty, and review effort.
在这一步中，我们对三个属性进行分类：问题领域、难度和审查工作量。

Problem domain refers to the category that the PR's associated issue problem statement belongs to.
问题领域指的是 PR 相关的议题问题陈述所属的类别。

Referring to SWE-Bench (Jimenez et al., 2024), an issue-solving benchmark, there are nine categories of PR problem domains, as summarized in Table 4.
参考议题解决基准 SWE-Bench (Jimenez et al., 2024)，共有九类 PR 问题领域，如表 4 所总结。

Difficulty means how difficult it would be to implement a PR.
难度意味着实施一个 PR 的困难程度。

It can be categorized as low, medium, and high.
它可以分为低、中和高三类。

Review effort means the effort required to review a code change.
审查工作量意味着审查代码变更所需的工作量。

It is on a scale of 1 to 5.
它的范围是 1 到 5。

5 means the most effort.
5 表示最大的工作量。

If the repository contains annotations for the above three attributes, we directly utilize the labels from the repository.
如果仓库包含上述三个属性的标注，我们会直接使用仓库中的标签。

Otherwise, we employ the LLM-as-a-judge approach (Gu et al., 2025).
否则，我们采用“LLM 即裁判”（LLM-as-a-judge）的方法 (Gu et al., 2025)。

We leverage the Qwen3-235B-A22B model (Yang et al., 2025), constructing prompts from the Title, Body, Commit Patch to Review, and Head Commit Message fields in Table 2 as input to classify the three attributes.
我们利用 Qwen3-235B-A22B 模型 (Yang et al., 2025)，根据表 2 中的标题、正文、待审查的提交补丁和头部提交信息字段构建提示词作为输入，以此对这三个属性进行分类。

At last, we get 40,124 CR task instances.
最终，我们获得了 40,124 个代码审查任务实例。

3.2.5 Manual Selection & Annotation
3.2.5 人工选择与标注

In this step, we manually select high-quality CR task instances that cover nine types of PR problem domains from the 40,124 CR task instances that have passed the filtering process above to serve as the benchmark.
在这一步中，我们从上述通过过滤过程的 40,124 个代码审查任务实例中，人工筛选出涵盖九种 PR 问题领域的高质量实例作为基准。

Specifically, we invite two of the authors who have more than 5 years of Python development experience to conduct the selection.
具体来说，我们邀请了两位拥有超过 5 年 Python 开发经验的作者来进行筛选。

Firstly, we find that a non-negligible percentage of review comments, while linked to source code lines in the commit patch, are unlikely to result in code revision in the next round.
首先，我们发现有不可忽视的一部分审查评论，虽然链接到了提交补丁中的源代码行，但不太可能导致下一轮的代码修订。

This kind of review comment is regarded as a noise comment.
这类审查评论被视为噪声评论。

For example, if a review: (1) that is a simple case (e.g., “looks good to me”, “Thanks!” or “Nice”);
例如，如果一条审查评论：(1) 是简单的情况（例如，“看起来不错”，“谢谢！”或“很好”）；

(2) only requests formatting changes with no impact on code logic (e.g., “fix indentation”, “add spaces”);
(2) 仅要求进行不影响代码逻辑的格式更改（例如，“修复缩进”，“添加空格”）；

(3) requests adding tests (these do not change the code under review);
(3) 要求添加测试（这些不会改变正在审查的代码）；

(4) asks for clarification or explanation without suggesting changes (e.g., “please explain”, “what does this do?”);
(4) 要求澄清或解释而不建议更改（例如，“请解释一下”，“这是做什么的？”）；

(5) refer previous comments that cannot be identified (e.g., “same as before”, “see above”);
(5) 引用无法识别的先前评论（例如，“同上”，“见上文”）；

(6) only requests adding comments or documentation (e.g., “add Javadoc”, “document this method”);
(6) 仅要求添加注释或文档（例如，“添加 Javadoc”，“为该方法添加文档”）；

(7) that is difficult to identify (e.g., “At least here it is clear that the equals method of the implementers of TreeNode is important”), it is considered as a noise and irrelevant review comment.
(7) 难以识别的情况（例如，“至少在这里很明显，TreeNode 实现者的 equals 方法很重要”），则被视为噪声和不相关的审查评论。

Instances that include these comments are not selected in the final benchmark.
包含这些评论的实例不会被选入最终的基准中。

In addition, we remove instances whose issue problem statement has images, external hyperlinks, references to specific commit SHAs, and references to other pull requests or issues (Jimenez et al., 2024).
此外，我们移除了其议题问题陈述中包含图片、外部超链接、特定提交 SHA 的引用以及对其他拉取请求或议题的引用的实例 (Jimenez et al., 2024)。

The instance with fewer than 40 words in the problem statement is also removed.
问题陈述少于 40 个单词的实例也会被移除。

They are not considered in our scenario.
它们不被考虑在我们的场景中。

Based on the above rules, we have filtered out 7,086 instances.
基于上述规则，我们要过滤掉 7,086 个实例。

After that, referring to SWE-Bench (Jimenez et al., 2024) and Multi-SWE-Bench (Zan et al., 2025), we design a questionnaire about instance quality assessment and ask the two developers to complete the questionnaire for each instance.
之后，参考 SWE-Bench (Jimenez et al., 2024) 和 Multi-SWE-Bench (Zan et al., 2025)，我们设计了一份关于实例质量评估的问卷，并要求两位开发人员为每个实例填写问卷。

This questionnaire covers the following areas: (1) Problem statement and patch alignment;
该问卷涵盖以下方面：(1) 问题陈述与补丁的一致性；

(2) Review scope and comment coverage;
(2) 审查范围和评论覆盖面；

(3) Defects identified in the patch;
(3) 补丁中识别出的缺陷；

(4) Difficulty and review effort;
(4) 难度和审查工作量；

(5) Overall patch quality and risk;
(5) 整体补丁质量和风险；

(6) Dataset suitability; and (7) Confidence.
(6) 数据集适用性；以及 (7) 置信度。

Thereby, we select the 601 high-quality instances from 70 projects by manual annotation.
因此，我们通过人工标注从 70 个项目中选出了 601 个高质量实例。

3.3 Benchmark Characteristics
3.3 基准特性

3.3.1 Comprehensiveness-aware Benchmark
3.3.1 全面性感知基准

CodeFuse-CR-Bench is the pioneering CR benchmark to introduce the comprehensiveness-aware concept.
CodeFuse-CR-Bench 是首个引入全面性感知概念的代码审查基准。

Table 1 lists the difference of CodeFuse-CR-Bench and other representative benchmarks presented in Section 2.1, where ● means containing relevant information, ○ means containing no relevant information, and ◑ means only containing partial relevant information.
表 1 列出了 CodeFuse-CR-Bench 与 2.1 节中介绍的其他代表性基准的差异，其中 ● 表示包含相关信息，○

表示不包含相关信息，◑ 表示仅包含部分相关信息。

We highlight a significant oversight in prior research: the comprehensiveness of a CR task.
我们强调了先前研究中一个被严重忽视的问题：代码审查任务的全面性。

Most of the other CR benchmarks lack basic information and repository-level context information.
大多数其他代码审查基准缺乏基本信息和仓库级上下文信息。

They only have method-level patches and review comments, which significantly deviate from real CR scenarios.
它们只有方法级的补丁和审查评论，这与真实的代码审查场景严重偏离。

Consequently, they make automatic CR approaches miss some context information that contributes to CR.
因此，它们使自动代码审查方法错过了一些有助于代码审查的上下文信息。

Further, the performance of LLMs in real-world CR remains unknown.
此外，LLM 在现实世界代码审查中的表现仍然未知。

Regarding some issue-solving benchmarks, they also lack some PR and CR-related information.
关于一些议题解决基准，它们也缺乏一些 PR 和代码审查相关信息。

CodeFuse-CR-Bench fills this gap by including basic information, repository-level context information, and more completed PR-related and CR-related information.
CodeFuse-CR-Bench 通过包含基本信息、仓库级上下文信息以及更完整的 PR 相关和代码审查相关信息填补了这一空白。

Thereby, they can achieve repository-level CR.
因此，它们可以实现仓库级代码审查。

It serves as a benchmark that better simulates real-world CR, aiming to more accurately reflect the true performance of CR approaches.
它作为一个能更好地模拟现实世界代码审查的基准，旨在更准确地反映代码审查方法的真实性能。

3.3.2 Strict Filter Process
3.3.2 严格的过滤过程

We ensure the quality of the selected CR task instances from several aspects.
我们从多个方面确保所选代码审查任务实例的质量。

First, we ensure that the selected projects are of high quality because high-quality projects are generally mature and active.
首先，我们确保所选项目具有高质量，因为高质量的项目通常成熟且活跃。

They also have excellent maintenance and a complete PR and CR process.
它们还拥有出色的维护以及完整的 PR 和代码审查流程。

We select Python projects from the top 1,000 projects based on the number of stars, with an average of 21k stars.
我们根据星标数量从排名前 1,000 的项目中选择 Python 项目，平均星标数为 21k。

Also, we keep projects with more than 1,500 PRs, making the collection focus on those with rich, mature, and standardized CR practices.
此外，我们保留了 PR 数量超过 1,500 个的项目，使收集重点集中在那些拥有丰富、成熟且标准化的代码审查实践的项目上。

Retaining only projects maintained within the past year ensures that the analysis focuses on “active” projects whose CR practices reflect contemporary standards, thereby guaranteeing the research’s cutting-edge relevance and practical applicability.
仅保留过去一年内维护的项目，确保了分析集中在“活跃”项目上，这些项目的代码审查实践反映了当代标准，从而保证了研究的前沿相关性和实际适用性。

In addition, we ensure that we select merged PRs and PRs with at least 1 closing issue reference.
此外，我们确保选择已合并的 PR 以及至少有一个关闭议题引用的 PR。

On the one hand, merged PRs indicate that the PR has passed the project’s quality gates (including CR and automated testing).
一方面，已合并的 PR 表明该 PR 已经通过了项目的质量门控（包括代码审查和自动化测试）。

On the other hand, PRs associated with closing issues can make sure that every PR has a clear, traceable development intent and rich context.
另一方面，与关闭议题相关的 PR 可以确保每个 PR 都有清晰、可追溯的开发意图和丰富的上下文。

Furthermore, we design complete heuristic rules to select a representative commit for each PR.
此外，我们设计了完整的启发式规则来为每个 PR 选择一个具有代表性的提交。

These rules can ensure the selected commits: (1) directly reflect collaboration and improvements during CR (e.g., has_resolved_review_comments);
这些规则可以确保所选提交：(1) 直接反映代码审查期间的协作和改进（例如，has_resolved_review_comments）；

(2) possess clear, structured, and traceable contextual information (e.g., conventional_commit, issue_reference);
(2) 拥有清晰、结构化且可追溯的上下文信息（例如，conventional_commit, issue_reference）；

(3) adhere to best practices for atomicity and focus (e.g., reasonable_commit_size, focused_file_changes);
(3) 遵守原子性和聚焦的最佳实践（例如，reasonable_commit_size, focused_file_changes）；

(4) exclude noise commits generated by version control operations that do not reflect development intent (e.g., exclude_merge_commit).
(4) 排除由不反映开发意图的版本控制操作产生的噪声提交（例如，exclude_merge_commit）。

All these rules can ensure the selected commits have high quality.
所有这些规则都能确保所选提交具有高质量。

---

4 Evaluation Metric Design
4 评估指标设计

A comprehensive benchmark like CodeFuse-CR-Bench requires an equally comprehensive evaluation framework.
像 CodeFuse-CR-Bench 这样的综合基准测试需要一个同样综合的评估框架。

Existing evaluation metrics on CR only focus on the method-level patch to review, and they only involve evaluation of semantics similarity and consistency, which can not adapt to comprehensive CR.
现有的代码审查评估指标仅关注待审查的方法级补丁，且仅涉及语义相似性和一致性的评估，无法适应综合代码审查。

We propose a comprehensive evaluation metric to evaluate the quality of comprehensive CR.
我们提出了一种综合评估指标来评估综合代码审查的质量。

Fig. 4 shows an overview of the evaluation metric.
图 4 展示了评估指标的概览。

It consists of two parts, model-based evaluation and rule-based evaluation.
它由两部分组成：基于模型的评估和基于规则的评估。

4.1 Model-base Evaluation
4.1 基于模型的评估

Model-based evaluation is a black-box evaluation designed to leverage the powerful semantic understanding capabilities of LLMs to score and evaluate the quality of CR.
基于模型的评估是一种黑盒评估，旨在利用 LLM 强大的语义理解能力来对代码审查的质量进行评分和评估。

It aims to simulate human judgment of CR's "semantic quality" and "usefulness".
它旨在模拟人类对代码审查"语义质量"和"有用性"的判断。

In our paper, we use two kinds of evaluators in model-based evaluation, namely, the reward model and LLM-as-a-judge (Son et al., 2024).
在本文中，我们在基于模型的评估中使用了两种评估器，即奖励模型和 LLM 即裁判 (LLM-as-a-judge) (Son et al., 2024)。

4.1.1 Reward Model
4.1.1 奖励模型

Reward model (Yang et al., 2024) is an AI model that learns human preferences.
奖励模型 (Yang et al., 2024) 是一种学习人类偏好的 AI 模型。

Its core task is: given an input and one or more model outputs (Output/Candidate), it outputs a scalar score (Reward Score) that predicts the degree of human preference for that output (i.e., the extent to which it is perceived as high quality).
它的核心任务是：给定一个输入和一个或多个模型输出（输出/候选项），它输出一个标量分数（奖励分数），预测人类对该输出的偏好程度（即，它被认为是高质量的程度）。

In order to train a reward model to evaluate comprehensive CR, we need to collect the corresponding training set and design the training strategy.
为了训练一个奖励模型来评估综合代码审查，我们需要收集相应的训练集并设计训练策略。

Data Collection. Firstly, we define the positive CR data and the negative CR data.
数据收集。首先，我们定义正样本 CR 数据和负样本 CR 数据。

We collect data from GitHub repositories with over 100 stars.
我们从星标数超过 100 的 GitHub 仓库收集数据。

To prevent data leakage, we filter out the repositories used to select CR task instances in Fig. 3.
为了防止数据泄露，我们过滤掉了图 3 中用于选择代码审查任务实例的仓库。

We use commits as the collection unit and adopt the `has_resolved_review_comments` and `has_referenced_line_changed_comments` rules from Table 3 as labeling criteria.
我们以提交为收集单位，并采用表 3 中的 `has_resolved_review_comments`（有已解决的审查评论）和 `has_referenced_line_changed_comments`（有引用行变更的评论）规则作为标注标准。

If both rules are True, the commit is classified as positive CR data and labeled as 1.
如果两条规则均为真，则该提交被归类为正样本代码审查数据并标记为 1。

Otherwise, the commit is classified as negative CR data and labeled as 0.
否则，该提交被归类为负样本代码审查数据并标记为 0。

At last, we collect 174,661 positive data and 114,458 negative data.
最终，我们收集了 174,661 条正样本数据和 114,458 条负样本数据。

Training Strategy. Our reward model is built upon the Qwen3-8B (Yang et al., 2025) pretrained LLM, where we replace the final LM head layer with a Multi-Layer Perceptron consisting of two linear layers and a ReLU activation function.
训练策略。我们的奖励模型建立在 Qwen3-8B (Yang et al., 2025) 预训练 LLM 之上，我们将最后的 LM 头层替换为一个由两个线性层和一个 ReLU 激活函数组成的多层感知机。

The first linear layer has dimensions of (hidden_size, hidden_size), while the second layer has dimensions of (hidden_size, 1).
第一个线性层的维度为 (hidden_size, hidden_size)，而第二个线性层的维度为 (hidden_size, 1)。

The final output represents a relevance score indicating how well the CR correlates with the given query and patch.
最终输出代表一个相关性分数，指示代码审查与给定查询和补丁的相关程度。

The training loss comprises two components.
训练损失包含两个部分。

The first component follows a similar approach to the reward model training in InstructGPT (Ouyang et al., 2022), employing a contrastive learning-style Bayesian Personalized Ranking (BPR) loss (Rendle et al., 2012).
第一部分采用与 InstructGPT (Ouyang et al., 2022) 中的奖励模型训练类似的方法，采用对比学习风格的贝叶斯个性化排名 (BPR) 损失 (Rendle et al., 2012)。

Specifically, for the same query and patch, we sample positive and negative reviews along with their corresponding code context (reviewed file content as default) and diff hunks to achieve this objective.
具体来说，针对相同的查询和补丁，我们对正样本和负样本审查及其相应的代码上下文（默认为被审查的文件内容）和差异块进行采样以实现此目标。

The loss is formulated as: (Equation 1)
损失公式如下：（公式 1）

Additionally, considering our application scenario of classifying review relevance, we introduce an auxiliary classification objective using binary cross-entropy loss (Wang et al., 2025a), that is, (Equation 2)
此外，考虑到我们对审查相关性进行分类的应用场景，我们引入了使用二元交叉熵损失的辅助分类目标 (Wang et al., 2025a)，即，（公式 2）

where $p_i = \sigma(r_\theta(x_i, y_i))$.
其中 $p_i = \sigma(r_\theta(x_i, y_i))$。

Specifically, r_θ denotes the reward model parameterized by θ, x represents the input consisting of query and patch information, y^+ and y^- denote positive and negative reviews respectively, σ is the sigmoid function, $l_i \in \{0, 1\}$ is the binary relevance label for the i-th sample, and p_i is the predicted probability of relevance.
具体而言，r_θ 表示由 θ 参数化的奖励模型，x 表示由查询和补丁信息组成的输入，y^+ 和 y^- 分别表示正样本和负样本审查，σ 是 sigmoid 函数，$l_i \in \{0, 1\}$ 是第 i 个样本的二元相关性标签，p_i 是预测的相关性概率。

The total training loss is the weighted combination of these two components: (Equation 3)
总训练损失是这两部分的加权组合：（公式 3）

where λ controls the relative importance of $\mathcal{L}_{BCE}$. We set it to 1 by default.
其中 λ 控制 $\mathcal{L}_{BCE}$ 的相对重要性。我们默认将其设置为 1。

Implementation Details. For efficient training, we employ Low-Rank Adaptation (LoRA) (Hu et al., 2022a) as our parameter-efficient fine-tuning method with a rank of 32 and an alpha value of 16, which significantly reduces the number of trainable parameters while maintaining model performance.
实施细节。为了高效训练，我们采用低秩适应 (LoRA) (Hu et al., 2022a) 作为参数高效微调方法，秩设为 32，alpha 值设为 16，这在保持模型性能的同时显著减少了可训练参数的数量。

The model is trained for 1 epoch with a learning rate of $5e-5$.
模型训练 1 个 epoch，学习率为 $5e-5$。

To accommodate the comprehensive CR context, we set the maximum sequence length to 24,576 tokens, enabling the model to process lengthy code patches and associated reviews effectively.
为了适应综合代码审查上下文，我们将最大序列长度设置为 24,576 个 token，使模型能够有效地处理冗长的代码补丁和相关审查。

We utilize Flash Attention 2 (Dao, 2024) for memory-efficient attention computation, with mixed precision training with bfloat16.
我们利用 Flash Attention 2 (Dao, 2024) 进行内存高效的注意力计算，并使用 bfloat16 进行混合精度训练。

The training process incorporates 100 warm-up steps for learning rate scheduling to ensure stable convergence.
训练过程包含 100 个预热步骤用于学习率调度，以确保稳定收敛。

The distributed training is conducted using DeepSpeed ZeRO (Rajbhandari et al., 2020) Stage 2.
分布式训练使用 DeepSpeed ZeRO (Rajbhandari et al., 2020) Stage 2 进行。

All experiments are conducted on 32 NVIDIA A100 GPUs with a total effective batch size of 64.
所有实验均在 32 个 NVIDIA A100 GPU 上进行，总有效批量大小为 64。

4.1.2 LLM-as-a-Judge
4.1.2 LLM 即裁判

To evaluate the CR in a general perspective, we also adopt LLM-as-a-judge to evaluate the CR quality.
为了从一般角度评估代码审查，我们也采用 LLM 即裁判 (LLM-as-a-judge) 来评估代码审查质量。

Specifically, we design a prompt that consists of CR information and evaluation criteria to request an LLM to provide a score for the CR quality.
具体来说，我们设计了一个包含代码审查信息和评估标准的提示词，要求 LLM 为代码审查质量打分。

Referring to some research about CR usefulness investigation (Yang et al., 2023; Turzo & Bosu, 2024), we request LLM to provide a score of CR based on four perspectives, that is, Functionality, Quality, Style, and Documentation.
参考一些关于代码审查有用性调查的研究 (Yang et al., 2023; Turzo & Bosu, 2024)，我们要求 LLM 基于四个视角提供代码审查评分，即功能性、质量、风格和文档。

For each perspective, we request LLM to analyze five dimensions of Correctness, Relevance, Clarity, Consistency and Language.
对于每个视角，我们要求 LLM 分析正确性、相关性、清晰度、一致性和语言这五个维度。

Each dimension will be provided a score by LLM and then an average overall score will be calculated at last.
每个维度将由 LLM 给出一个分数，最后计算出平均总分。

The detail prompt is shown in Fig. 5.
详细的提示词如图 5 所示。

We use OpenAI o3-2025-04-16 (OpenAI, 2025b) model to achieve LLM-as-a-judge.
我们使用 OpenAI o3-2025-04-16 (OpenAI, 2025b) 模型来实现 LLM 即裁判。

Thereby, we can obtain a score provided by LLM.
从而，我们可以获得 LLM 提供的分数。

By combining scores based on the reward model and LLM-as-a-judge, respectively, we get an average score with model-based evaluation.
通过分别结合基于奖励模型和 LLM 即裁判的分数，我们得到了基于模型评估的平均分。

4.2 Rule-base Evaluation
4.2 基于规则的评估

To achieve a comprehensive evaluation metric, we also adopt a rule-based evaluation.
为了实现综合评估指标，我们也采用基于规则的评估。

Rule-based evaluation is a white-box evaluation, which aims to determine the “formal correctness” and “superficial similarity” of CR.
基于规则的评估是一种白盒评估，旨在确定代码审查的“形式正确性”和“表面相似性”。

We automatically extract structured defect information from the CR reports generated by the LLMs, including file paths, line numbers, and review comments.
我们从 LLM 生成的代码审查报告中自动提取结构化的缺陷信息，包括文件路径、行号和审查评论。

Then we conduct some heuristic rules which consists of location similarity, semantics similarity and defects match to implement rule-based evaluation.
然后，我们执行一些启发式规则，包括位置相似性、语义相似性和缺陷匹配，以实施基于规则的评估。

4.2.1 Location Similarity
4.2.1 位置相似性

Location similarity aims to evaluate the matching between the predicted CR location and the CR ground truth.
位置相似性旨在评估预测的代码审查位置与代码审查真实值之间的匹配程度。

It includes file path matching, line number proximity, and diff hunk match level.
它包括文件路径匹配、行号接近度和差异块匹配级别。

Specifically, if the file path where the predicted CR locates is exact match with the ground truth, we give a score of 1. Otherwise, we give a score of 0.
具体来说，如果预测代码审查所在的文件路径与真实值完全匹配，我们给 1 分。否则，我们给 0 分。

Regarding the line number accuracy, if it is an exact match with the ground truth, we give a score of 1.
关于行号准确性，如果它与真实值完全匹配，我们给 1 分。

If the line difference $diff_{line}$ is more than 5 lines, we give a score of 0.1.
如果行差 $diff_{line}$ 超过 5 行，我们给 0.1 分。

Otherwise, we conduct a decay function to calculate the line number accuracy Acc_{LN}, which is shown as follows. (Equation 4)
否则，我们执行一个衰减函数来计算行号准确性 Acc_{LN}，如下所示。（公式 4）

In addition, to avoid unduly penalizing essentially correct predictions due to minor line number offsets, we introduce a more robust metric: diff hunk similarity.
此外，为了避免因轻微的行号偏移而过度惩罚本质上正确的预测，我们引入了一个更稳健的指标：差异块相似性。

This metric evaluates whether the LLM's predicted CR location and the ground truth location reside within the same code change block.
该指标评估 LLM 预测的代码审查位置和真实位置是否位于同一个代码变更块内。

Specifically, We analyze whether the predicted CR's line number resides within the same diff hunk as the ground truth.
具体来说，我们分析预测的代码审查行号是否与真实值位于同一个差异块内。

If it does, we assign a score of 1.
如果是，我们分配 1 分。

If the predicted CR is in a different diff hunk, we assign a score of 0.
如果预测的代码审查位于不同的差异块中，我们分配 0 分。

If the predicted CR's line number is not present in any diff hunk in the patch to review, we collect the five lines above and below the CR line and calculate the overlap ratio between this range and the ground truth diff hunk's line number range as the diff hunk similarity score.
如果预测的代码审查行号不存在于待审查补丁的任何差异块中，我们将收集代码审查行上下的五行，并计算该范围与真实差异块行号范围之间的重叠率作为差异块相似性分数。

Thereby, combining with the file path matching and line number accuracy, diff hunk similarity can contribute to a more comprehensive and fairer location evaluation.
因此，结合文件路径匹配和行号准确性，差异块相似性有助于进行更全面、更公平的位置评估。

We assign weights of 70%, 15%, and 15% to file path matching, line number accuracy, and diff hunk similarity, respectively, ultimately yielding a composite location similarity score.
我们分别赋予文件路径匹配、行号准确性和差异块相似性 70%、15% 和 15% 的权重，最终得出一个综合的位置相似性分数。

4.2.2 Semantics Similarity
4.2.2 语义相似性

In order to evaluate the semantics performance of review comments, we use BLEU (Papineni et al., 2002), a popular evaluation metric that is implemented in neural machine translation and conversation systems.
为了评估审查评论的语义性能，我们使用 BLEU (Papineni et al., 2002)，这是一种在神经机器翻译和对话系统中实施的流行评估指标。

BLEU is now widely used in code-related tasks, such as code comment generation (Guo et al., 2023), document generation (Hu et al., 2022b), review comment generation (Li et al., 2022b) and so on.
BLEU 现在广泛用于代码相关任务，如代码注释生成 (Guo et al., 2023)、文档生成 (Hu et al., 2022b)、审查评论生成 (Li et al., 2022b) 等。

Its core idea is to compare the n-gram overlap between the text generated by the model and the ground truth.
其核心思想是比较模型生成的文本与真实值之间的 n-gram 重叠。

The greater the overlap, the higher the BLEU score.
重叠越多，BLEU 分数越高。

The specific calculation process of BLEU is given as follows: (Equation 5)
BLEU 的具体计算过程如下：（公式 5）

where ω_n is the weight of n-gram and p_n is the precision of n-gram.
其中 ω_n 是 n-gram 的权重，p_n 是 n-gram 的精度。

Usually, the maximum value of n is 4, which is represented by BLEU-4.
通常，n 的最大值为 4，用 BLEU-4 表示。

BP is the brevity penalty factor for generated review comment length, which is shown as follows: (Equation 6)
BP 是针对生成的审查评论长度的简短惩罚因子，如下所示：（公式 6）

where l_c is the length of generated review comment and l_r represents the reference review comment.
其中 l_c 是生成的审查评论的长度，l_r 代表参考审查评论。

In this paper, we employ BLEU-4 as the semantics similarity.
在本文中，我们采用 BLEU-4 作为语义相似性。

4.2.3 Defects Match
4.2.3 缺陷匹配

Since a CR task may involve reviewing multiple review comments and locations within a commit, we design a defect match rule to evaluate performance under multi-location review scenarios.
由于一个代码审查任务可能涉及审查一个提交内的多个审查评论和位置，我们设计了一个缺陷匹配规则来评估多位置审查场景下的性能。

Specifically, for each predicted review comment and location (i.e., predicted defect), we calculate the average of the location similarity and semantics similarity scores between it and each ground-truth review comment and location, namely sub-defect-match scores.
具体来说，对于每个预测的审查评论和位置（即预测缺陷），我们计算其与每个真实审查评论和位置之间的位置相似性和语义相似性分数的平均值，即子缺陷匹配分数。

We select the highest sub-defect-match score among them.
我们在其中选择最高的子缺陷匹配分数。

If this score exceeds the set threshold, we consider the predicted review comment and location to be correctly matched to the defect information.
如果该分数超过设定的阈值，我们认为预测的审查评论和位置已正确匹配到缺陷信息。

Based on the number of correctly matched defects, we can calculate precision, recall, and F1 scores.
基于正确匹配的缺陷数量，我们可以计算精确率、召回率和 F1 分数。

Precision = ... (Equation 7)
精确率 = ...（公式 7）

Recall = ... (Equation 8)
召回率 = ...（公式 8）

F1 = ... (Equation 9)
F1 = ...（公式 9）

The total score of defect match is calculated by averaging the F1 score and the average of sub-defect-match scores.
缺陷匹配的总分是通过平均 F1 分数和子缺陷匹配分数的平均值计算得出的。

After getting the total defect score, we combine and average it with location similarity and semantics similarity score to get a rule-based score.
在获得总缺陷分数后，我们将其与位置相似性和语义相似性分数结合并取平均值，得到基于规则的分数。

At last we get an overall evaluation score based on the average of the model-based score and the rule-based score.
最后，我们基于基于模型的分数和基于规则的分数的平均值得到一个总体评估分数。

5 Experiment Design
5 实验设计

In this section, we first provide the detailed distribution of the benchmark, introduce the models, context acquisition strategy, and the prompt used in the experiments, and provide a detailed description of the experimental setups.
在本节中，我们首先提供基准的详细分布，介绍实验中使用的模型、上下文获取策略和提示词，并提供实验设置的详细描述。

After that, we answer the following four research questions that the experiments aim to address:
之后，我们将回答实验旨在解决的以下四个研究问题：

RQ1: What is the LLMs' performance of comprehensive CR in CodeFuse-CR-Bench benchmark?
RQ1： LLM 在 CodeFuse-CR-Bench 基准中的综合代码审查表现如何？

RQ2: What is the LLM's performance of comprehensive CR in different PR types?
RQ2： LLM 在不同 PR 类型中的综合代码审查表现如何？

RQ3: How does different contextual information contribute to CR performance?
RQ3： 不同的上下文信息如何对代码审查性能做出贡献？

RQ4: What is the performance of the reward model?
RQ4： 奖励模型的性能如何？

5.1 Benchmark Distribution
5.1 基准分布

After implementing the pipeline in Section 3.2, we construct a benchmark with 601 Python CR task instances, which were created from 2016-11-06 to 2025-07-07.
在实施 3.2 节中的流程后，我们构建了一个包含 601 个 Python 代码审查任务实例的基准，这些实例创建于 2016-11-06 至 2025-07-07 之间。

The problem domain distribution is shown in Fig. 6a.
问题领域分布如图 6a 所示。

It can be found that bug fixes constitute the problem domain with the highest volume of high-quality CRs.
可以发现，错误修复构成了具有最高数量高质量代码审查的问题领域。

This indicates robust software maintenance practices and a thriving CR ecosystem for this category of issues.
这表明此类议题拥有稳健的软件维护实践和繁荣的代码审查生态系统。

The review effort and difficulty distribution are shown in Fig. 6b.
审查工作量和难度分布如图 6b 所示。

Most instances are in a medium difficulty of PR task implementation.
大多数实例处于 PR 任务实施的中等难度。

The most common review effort is 3, which also indicates that most CRs are assigned to tasks of moderate difficulty.
最常见的审查工作量是 3，这也表明大多数代码审查被分配给中等难度的任务。

The 601 instances involve 70 Python projects.
这 601 个实例涉及 70 个 Python 项目。

Fig. 7 illustrates the project distribution of the benchmark.
图 7 展示了基准的项目分布。

We present the top 17 projects containing the highest number of instances.
我们展示了包含实例数量最多的前 17 个项目。

These projects account for 55.41% of the total instances.
这些项目占总实例数的 55.41%。

Most of these projects are well-known within the Python community (e.g., pandas, scikit-learn et al.).
这些项目大多数在 Python 社区中广为人知（例如 pandas, scikit-learn 等）。

This indicates that these projects are well-maintained with high-quality CRs.
这表明这些项目维护良好，拥有高质量的代码审查。

5.2 Studied LLMs
5.2 研究的 LLM

We consider mainstream LLMs—encompassing both open-source and proprietary models—that have been widely adopted in recent software-engineering-related studies (Fan et al., 2025; Batole et al., 2025; Wang et al., 2024; Yin et al., 2024).
我们考虑主流 LLM——包含开源和专有模型——它们在最近的软件工程相关研究中已被广泛采用 (Fan et al., 2025; Batole et al., 2025; Wang et al., 2024; Yin et al., 2024)。

For open-source LLMs, we select DeepSeek-v3.1 (Guo et al., 2025), Kimi-K2-0905-preview (Team, 2025) Qwen3-235B-A22B (Yang et al., 2025).
对于开源 LLM，我们选择了 DeepSeek-v3.1 (Guo et al., 2025)、Kimi-K2-0905-preview (Team, 2025) 和 Qwen3-235B-A22B (Yang et al., 2025)。

For closed-source LLMs, we choose the commonly used commercial models: Claude-Sonnet-4-20250514 (Anthropic, 2025), Gemini 2.5 Pro (Comanici et al., 2025), GPT-4o (Hurst et al., 2024), GPT-5 (OpenAI, 2025a).
对于闭源 LLM，我们选择了常用的商业模型：Claude-Sonnet-4-20250514 (Anthropic, 2025)、Gemini 2.5 Pro (Comanici et al., 2025)、GPT-4o (Hurst et al., 2024) 和 GPT-5 (OpenAI, 2025a)。

They have excellent performance in code-related tasks in previous benchmarks.
它们在先前的基准测试中的代码相关任务上表现出色。

5.3 Studied Context Acquisition Strategies
5.3 研究的上下文获取策略

As a repository-level CR benchmark, CodeFuse-CR-Bench can provide repository-level contextual information as part of the LLM input to help the LLM achieve comprehensive CR.
作为仓库级代码审查基准，CodeFuse-CR-Bench 可以提供仓库级上下文信息作为 LLM 输入的一部分，以帮助 LLM 实现综合代码审查。

Due to the large size of modern software repositories, it is not feasible to feed the entire repository to the model.
由于现代软件仓库规模庞大，将整个仓库输入模型是不可行的。

It is necessary to retrieve the most relevant information.
有必要检索最相关的信息。

Referring to SWE-Bench (Jimenez et al., 2024), we adopt two types of context acquisition strategies that might be beneficial for comprehensive CR: Retrieval-based Acquisition and Oracle-based context acquisition.
参考 SWE-Bench (Jimenez et al., 2024)，我们采用两种可能对综合代码审查有益的上下文获取策略：基于检索的获取和基于 Oracle 的上下文获取。

In RQ3, we will validate the performance of these two strategies.
在 RQ3 中，我们将验证这两种策略的性能。

Retrieval-based Context Acquisition.
基于检索的上下文获取。

Since the problem description and the code implementing the functionality may share overlapping vocabulary, we employ BM25 Stephen2004 to retrieve code files similar to the problem statement within the project.
由于问题描述和实现该功能的代码可能共享重叠的词汇，我们采用 BM25 Stephen2004 来检索项目内与问题陈述相似的代码文件。

BM25 is a ranking function commonly used in information retrieval to measure the relevance between a document and a query.
BM25 是信息检索中常用的一种排序函数，用于衡量文档与查询之间的相关性。

As a kind of sparse retrieval method, BM25 offers fast computation speed and low resource consumption, making it highly suitable for file retrieval scenarios in large-scale software projects.
作为一种稀疏检索方法，BM25 提供了快速的计算速度和低资源消耗，使其非常适合大规模软件项目中的文件检索场景。

In RQ3, we will separately analyze the impact of using the top-1, top-3 and top-5 relevant code files retrieved via BM25 as context on the effectiveness of LLM-based CR.
在 RQ3 中，我们将分别分析使用通过 BM25 检索到的前 1 个、前 3 个和前 5 个相关代码文件作为上下文，对基于 LLM 的代码审查有效性的影响。

Oracle-based Context Acquisition.
基于 Oracle 的上下文获取。

As a baseline, we also consider the oracle-based retrieval.
作为一个基线，我们也考虑基于 Oracle 的检索。

Specifically, we retrieve the combination of changed files in diff(base commit, commit to review) ∪ diff(base commit, merged commit).
具体来说，我们检索 diff(基础提交, 待审查提交) ∪ diff(基础提交, 合并提交) 中更改文件的组合。

The reason lies in the fact that there are some other commits related to the target commit in the PR when extracting the target commit in PR as the CR target.
原因在于，当提取 PR 中的目标提交作为代码审查目标时，PR 中还存在其他与目标提交相关的提交。

These related commits may serve as supplementary modifications to the target commit.
这些相关提交可以作为对目标提交的补充修改。

Therefore, we choose the change files from commits in the same PR as the largest context space.
因此，我们选择同一 PR 中提交的更改文件作为最大的上下文空间。

5.4 Experimental Setting
5.4 实验设置

We use the evaluation framework proposed in Section 4 to assess the CR generated by LLMs.
我们使用第 4 节中提出的评估框架来评估 LLM 生成的代码审查。

We set the generation temperature of LLMs to 0.6 (except GPT-5 for 1), top-p to 0.95.
我们将 LLM 的生成温度设置为 0.6（GPT-5 设为 1），top-p 设置为 0.95。

The context window is set based on the maximum context window length specified in each LLM's system card.
上下文窗口是根据每个 LLM 系统卡中指定的最大上下文窗口长度设置的。

Note that, when employing different context retrieval strategies, once the token length exceeds the context window, we remove the retrieved files outside the context window.
请注意，当采用不同的上下文检索策略时，一旦 token 长度超过上下文窗口，我们将移除上下文窗口之外的检索文件。

When the input remains longer than the context window length after removing all retrieved files, we consider the LLM unable to provide a correct CR for that instance and assign an evaluation score of 0.
当移除所有检索文件后输入仍然长于上下文窗口长度时，我们认为 LLM 无法为该实例提供正确的代码审查，并分配 0 的评估分。

To mitigate issues stemming from the randomness of model generation, the experimental results presented in this paper are obtained by conducting three repeated experiments and averaging the results.
为了减轻模型生成随机性带来的问题，本文展示的实验结果是通过进行三次重复实验并取平均值获得的。

The prompt template is shown in Fig. 8, which consists of the issue tag including the problem statement, the code tag including the context, and the patch tag including the patch to review.
提示词模板如图 8 所示，它由包含问题陈述的 issue 标签、包含上下文的 code 标签以及包含待审查补丁的 patch 标签组成。

The generated CR format is also described in the review tag, which involves the functional implementation, code quality, and defect information.
生成的代码审查格式也在 review 标签中描述，其中涉及功能实现、代码质量和缺陷信息。

5.5 RQ1: What is the LLMs' performance of comprehensive CR in CodeFuse-CR-Bench benchmark?
5.5 RQ1：LLM 在 CodeFuse-CR-Bench 基准中的综合代码审查表现如何？

We conduct experiments under the oracle-based context as the baseline setting to explore the performance of mainstream LLMs in comprehensive CR.
我们在基于 Oracle 的上下文作为基线设置下进行实验，以探索主流 LLM 在综合代码审查中的表现。

Table 5 shows the results, which reveal distinct strengths and weaknesses across models.
表 5 显示了结果，揭示了各个模型之间明显的优势和劣势。

This suggests that Gemini is currently the most well-rounded performer in generating contextually grounded and technically sound CR feedback.
这表明 Gemini 目前在生成基于上下文且技术合理的代码审查反馈方面表现最为全面。

On the model-based evaluation dimension, GPT-5 attains the best score (64.80), followed closely by Gemini 2.5 Pro (63.65).
在基于模型的评估维度上，GPT-5 获得了最高分（64.80），紧随其后的是 Gemini 2.5 Pro（63.65）。

This demonstrates that GPT-5 generates review comments with the highest perceived quality in terms of semantic quality and usefulness, even if its overall integration across dimensions is not optimal.
这表明 GPT-5 在语义质量和有用性方面生成的审查评论具有最高的感知质量，即使它在各维度上的整体整合并非最佳。

In contrast, Claude-Sonnet-4-20250514 excels in the rule-based evaluation (33.31) and significantly outperforms models like GPT-4o (8.10) and GPT-5 (18.30), indicating its strong adherence to location similarity, semantics similarity, and defect match.
相比之下，Claude-Sonnet-4-20250514 在基于规则的评估中表现出色（33.31），并显著优于 GPT-4o（8.10）和 GPT-5（18.30）等模型，表明其在位置相似性、语义相似性和缺陷匹配方面有很强的依从性。

This highlights Claude's strength in producing formally correct and precise feedback, albeit sometimes at the cost of broader contextual integration.
这凸显了 Claude 在生成形式正确且精确的反馈方面的优势，尽管有时是以牺牲更广泛的上下文整合为代价的。

Notably, several models exhibit significant imbalances.
值得注意的是，几个模型表现出显著的不平衡。

For instance, while GPT-5 performs best in model-based scoring, its rule-based performance lags, resulting in a lower comprehensive score (41.96) than Gemini.
例如，虽然 GPT-5 在基于模型的评分中表现最佳，但其基于规则的表现滞后，导致其综合得分（41.96）低于 Gemini。

Similarly, GPT-4o shows moderate model-based performance but severely underperforms in rule-based checks, suggesting potential issues with factual grounding or hallucination in CR scenarios.
同样，GPT-4o 显示出中等的基于模型的性能，但在基于规则的检查中表现严重不佳，这表明在代码审查场景中可能存在事实依据不足或幻觉的问题。

These findings underscore that no single LLM dominates across all dimensions of comprehensive code review.
这些发现强调，没有任何单一的 LLM 能够在综合代码审查的所有维度上占据主导地位。

Instead, performance varies significantly depending on the evaluation axis, emphasizing the necessity of our multi-faceted assessment framework.
相反，性能因评估维度的不同而显著差异，这强调了我们需要多层面评估框架的必要性。

The results also highlight the importance of integrating both model-based and rule-based metrics to avoid overestimating the practical utility of generated CRs.
结果还强调了整合基于模型和基于规则的指标的重要性，以避免高估生成的代码审查的实际效用。

Answer to RQ1: Gemini 2.5 Pro achieves the highest comprehensive CR performance on CodeFuse-CR-Bench, outperforming other LLMs by balancing strong model-based and rule-based scores, while GPT-5 and Claude-Sonnet excel in semantic quality and formal correctness, respectively, highlighting the necessity of multi-dimensional evaluation for realistic CR assessment.
RQ1 回答： Gemini 2.5 Pro 在 CodeFuse-CR-Bench 上取得了最高的综合代码审查性能，通过平衡强大的基于模型和基于规则的分数超越了其他 LLM，而 GPT-5 和 Claude-Sonnet 分别在语义质量和形式正确性方面表现出色，凸显了对现实代码审查评估进行多维评估的必要性。

5.6 RQ2: What is the LLM's performance of comprehensive CR in different PR types?
5.6 RQ2：LLM 在不同 PR 类型中的综合代码审查表现如何？

In Section 3.2.4, we label the CR task instances into nine problem domains based on the PR types.
在 3.2.4 节中，我们根据 PR 类型将代码审查任务实例标记为九个问题领域。

In this RQ, we conduct a fine-grained analysis of their comprehensive CR capabilities within each domain, with results presented in Table 6.
在这个 RQ 中，我们对每个领域内的综合代码审查能力进行了细粒度分析，结果如表 6 所示。

It can be found that Gemini 2.5 Pro achieves the highest comprehensive score in all nine problem domains.
可以发现，Gemini 2.5 Pro 在所有九个问题领域中都获得了最高的综合得分。

It outperforms the second-best model in each domain by margins ranging from 5.93% to 28.17%, with particularly strong performance in DU (58.86), TC (55.83), and DE (53.35).
它在每个领域都以 5.93% 到 28.17% 的幅度优于排名第二的模型，尤其是在 DU (58.86)、TC (55.83) 和 DE (53.35) 方面表现强劲。

This uniform dominance suggests that Gemini 2.5 Pro not only generalizes well across different types of code changes but also effectively leverages domain-specific context—such as test requirements, security constraints, or formatting rules—to generate more accurate and actionable CR feedback.
这种统一的优势表明，Gemini 2.5 Pro 不仅在不同类型的代码变更中具有良好的泛化能力，而且还能有效地利用特定领域的上下文——如测试要求、安全约束或格式规则——来生成更准确和可操作的代码审查反馈。

The results highlight its robustness and adaptability in handling the heterogeneous nature of modern CR practices.
结果突显了其在处理现代代码审查实践的异构性质方面的鲁棒性和适应性。

Answer to RQ2: Gemini 2.5 Pro consistently outperforms all other LLMs in comprehensive CR across all nine problem domains, demonstrating its superior generalization and context utilization in diverse CR tasks.
RQ2 回答： Gemini 2.5 Pro 在所有九个问题领域的综合代码审查中始终优于所有其他 LLM，展示了其在多样化代码审查任务中卓越的泛化能力和上下文利用能力。

5.7 RQ3: How does different contextual information contribute to CR performance?
5.7 RQ3：不同的上下文信息如何对代码审查性能做出贡献?

In Section 5.3, we propose two types of context acquisition strategies that may be helpful for LLMs to achieve comprehensive CR.
在 5.3 节中，我们提出了两种可能有助于 LLM 实现综合代码审查的上下文获取策略。

In this RQ, we evaluate the effectiveness of these strategies under varying context availability.
在这个 RQ 中，我们评估了这些策略在不同上下文可用性下的有效性。

Particularly, regarding retrieval-based context acquisition, we adopt BM25 to retrieve top-1, top-3 and top-5 relevant files as the context, respectively.
特别是关于基于检索的上下文获取，我们采用 BM25 分别检索前 1 个、前 3 个和前 5 个相关文件作为上下文。

Results, as shown in Table 7, demonstrate that Gemini 2.5 Pro achieves the highest comprehensive score across all context configurations, including both oracle-based and BM25-based settings.
如表 7 所示的结果表明，Gemini 2.5 Pro 在所有上下文配置（包括基于 Oracle 和基于 BM25 的设置）中都获得了最高的综合得分。

Notably, its performance under the BM25 top-1 context (52.24) is nearly on par with that under the oracle-based setting (52.37), with only a marginal gap of 0.13.
值得注意的是，其在 BM25 top-1 上下文下的表现（52.24）几乎与基于 Oracle 设置下的表现（52.37）持平，仅有 0.13 的微小差距。

This indicates that Gemini can achieve near-optimal review quality by leveraging just the single most relevant retrieved file, highlighting its strong capability in contextual relevance filtering and efficient information utilization.
这表明 Gemini 仅利用检索到的最相关的一个文件就能实现接近最佳的审查质量，凸显了其在上下文相关性过滤和高效信息利用方面的强大能力。

Furthermore, Gemini maintains stable performance across different retrieval depths (top-1: 52.24, top-3: 51.45, top-5: 51.63), suggesting robustness to context size and resilience to potential noise in larger retrieval sets.
此外，Gemini 在不同的检索深度（top-1: 52.24, top-3: 51.45, top-5: 51.63）下保持稳定的性能，这表明其对上下文大小具有鲁棒性，并且对较大检索集中的潜在噪声具有恢复力。

This stability contrasts with other models—such as GPT-5 and GPT-4o—whose scores fluctuate more significantly with context size, indicating higher sensitivity to irrelevant or redundant information.
这种稳定性与其他模型——如 GPT-5 和 GPT-4o——形成对比，后者的分数随上下文大小波动更显著，表明对不相关或冗余信息的敏感度更高。

These findings also [suggest that] Gemini 2.5 Pro is the most context-efficient and robust model for practical, scalable CR systems where oracle-level context is unavailable.
这些发现也[表明] Gemini 2.5 Pro 是对于实际、可扩展的代码审查系统而言最具有上下文效率和鲁棒性的模型，尤其是在无法获得 Oracle 级上下文的情况下。

Answer to RQ3: Gemini 2.5 Pro achieves near-oracle performance in comprehensive code review using only the top-1 retrieved context, demonstrating superior context efficiency and robustness across retrieval-based and oracle-based acquisition strategies.
RQ3 回答： Gemini 2.5 Pro 仅使用检索到的 top-1 上下文就在综合代码审查中实现了接近 Oracle 的性能，展示了在基于检索和基于 Oracle 的获取策略中卓越的上下文效率和鲁棒性。

5.8 RQ4: What is the performance of the reward model?
5.8 RQ4：奖励模型的性能如何？

In Section 4.1.1, we incorporate a reward model as a key component of our model-based evaluation framework.
在 4.1.1 节中，我们将奖励模型作为基于模型的评估框架的关键组件纳入其中。

To assess its reliability and validity in distinguishing high-quality from low-quality code reviews, we conduct a comparison study to evaluate the classification performance of the reward model.
为了评估其在区分高质量与低质量代码审查方面的可靠性和有效性，我们进行了一项比较研究来评估奖励模型的分类性能。

The input to the reward model consists of two elements: the query and the patch to review.
奖励模型的输入由两个元素组成：查询和待审查的补丁。

The query itself is composed of the context (i.e., the full content of the reviewed file, used as default context), review context, and the review location information.
查询本身由上下文（即，被审查文件的全部内容，用作默认上下文）、审查上下文和审查位置信息组成。

We split the dataset described in Section 4.1.1 into a 90% training set and a 10% test set to evaluate the model's classification effectiveness.
我们将 4.1.1 节中描述的数据集划分为 90% 的训练集和 10% 的测试集，以评估模型的分类有效性。

A CR is classified as positive if the reward model outputs a score greater than 0.5; otherwise, it is classified as negative.
如果奖励模型输出的分数大于 0.5，则代码审查被归类为正样本；否则，被归类为负样本。

As shown in Table 8, the reward model achieves an accuracy of 75.03% and an F1 score of 80.64%, demonstrating its strong discriminative capability.
如表 8 所示，奖励模型达到了 75.03% 的准确率和 80.64% 的 F1 分数，展示了其强大的判别能力。

Notably, even when the context information is removed, the model maintains competitive performance, with accuracy and F1 scores of 74.30% and 79.57%, respectively—suggesting that the review context, location and patch already provide substantial signal for review quality assessment.
值得注意的是，即使移除了上下文信息，该模型仍保持着具有竞争力的性能，准确率和 F1 分数分别为 74.30% 和 79.57%——这表明审查上下文、位置和补丁已经为审查质量评估提供了大量的信号。

For comparison, we evaluate two SOTA LLMs—Kimi-K2-0711-preview and Gemini 2.5 Pro—on the same binary classification task.
为了进行比较，我们在相同的二分类任务上评估了两个 SOTA LLM——Kimi-K2-0711-preview 和 Gemini 2.5 Pro。

The prompt provided to each LLM mirrors the input format used by the reward model, and the LLM's output is directly mapped to a binary label based on whether it expresses approval or disapproval of the review.
提供给每个 LLM 的提示词反映了奖励模型使用的输入格式，并且 LLM 的输出根据其表达的是对审查的批准还是不批准直接映射为二元标签。

The results show significantly lower performance: Kimi achieves 43.68% accuracy and 49.58% F1, while Gemini reaches 51.30% accuracy and 61.36% F1.
结果显示性能显著较低：Kimi 达到了 43.68% 的准确率和 49.58% 的 F1 分数，而 Gemini 达到了 51.30% 的准确率和 61.36% 的 F1 分数。

These findings indicate that the fine-tuned reward model substantially outperforms general-purpose LLMs in this specialized classification task, validating its suitability as a reliable evaluator within our comprehensive CR assessment framework.
这些发现表明，经过微调的奖励模型在这个专门的分类任务中大大优于通用 LLM，验证了其作为我们综合代码审查评估框架内可靠评估者的适用性。

In our final evaluation pipeline, the reward model is applied with the reviewed file context to ensure maximal fidelity to real review scenarios.
在我们的最终评估流程中，奖励模型与被审查的文件上下文一起应用，以确保对真实审查场景的最大保真度。

Answer to RQ4: The reward model we construct demonstrates objective performance in evaluating the quality of CRs, achieving an F1 score exceeding 80% and an accuracy exceeding 75%.
RQ4 回答： 我们构建的奖励模型在评估代码审查质量方面表现出客观的性能，实现了超过 80% 的 F1 分数和超过 75% 的准确率。

6 Threats to Validity
6 效度威胁

Threats to Internal Validity.
内部效度威胁。

For threats to internal validity, the first concern is the selection of prompts.
关于内部效度威胁，第一个担忧是提示词的选择。

The performance of LLMs is sensitive to prompt design.
LLM 的性能对提示词设计很敏感。

Using an equivalent but differently phrased prompt may result in significant performance differences.
使用等效但措辞不同的提示词可能会导致显著的性能差异。

To mitigate this threat, we ultimately chose the one that yielded the best results after trying several formats.
为了减轻这一威胁，我们在尝试了几种格式后最终选择了产生最佳结果的一种。

However, since we did not cover all available prompt formats, the one we selected may not be the optimal one.
然而，由于我们没有覆盖所有可用的提示词格式，我们选择的那个可能不是最佳的。

Secondly, since the setting of LLM hyperparameters (e.g., temperature), the outputs of LLM may exhibit randomness.
其次，由于 LLM 超参数（例如温度）的设置，LLM 的输出可能表现出随机性。

To mitigate this issue, we conduct three repeated experiments and average the results.
为了减轻这个问题，我们进行了三次重复实验并对结果取平均值。

Threats to External Validity.
外部效度威胁。

The first threat to external validity is the programming language generalization.
外部效度的第一个威胁是编程语言的泛化性。

CodeFuse-CR-Bench is a benchmark based on Python.
CodeFuse-CR-Bench 是一个基于 Python 的基准。

So it can not be used to evaluate comprehensive CR on other programming languages such as Java or C++.
因此，它不能用于评估其他编程语言（如 Java 或 C++）的综合代码审查。

In the future, we plan to address this limitation by continuously expanding CodeFuse-CR-Bench to cover as many programming languages as possible.
未来，我们计划通过不断扩展 CodeFuse-CR-Bench 以覆盖尽可能多的编程语言来解决这一局限性。

The second external validity is the type of generalization.
第二个外部效度是类型的泛化性。

CodeFuse-CR-Bench consists of instances including nine types of problem domains.
CodeFuse-CR-Bench 包含涵盖九种问题领域的实例。

However, there may exist some CRs involving some other types.
然而，可能存在涉及其他类型的代码审查。

We will try to expand the number of types in the future.
我们将尝试在未来扩展类型的数量。

Another threat is the model generalization.
另一个威胁是模型的泛化性。

Due to limited resources, our experiments are not able to cover all the available LLMs, thus not fully reflecting the performance of all LLMs in actual development scenarios, which may slightly affect the representativeness of our experiments.
由于资源有限，我们的实验无法覆盖所有可用的 LLM，因此不能完全反映所有 LLM 在实际开发场景中的表现，这可能会轻微影响我们实验的代表性。

7 Conclusion
7 结论

CR is an important component for building stable, maintainable, and high-quality software products.
代码审查是构建稳定、可维护和高质量软件产品的重要组成部分。

Many CR benchmarks have been proposed to evaluate different kinds of automatic CR approaches.
许多代码审查基准已被提出来评估不同类型的自动代码审查方法。

But there is a lack of comprehensiveness in existing benchmarks and approaches, which are detached from real CR scenarios.
但是，现有的基准和方法缺乏全面性，脱离了真实的代码审查场景。

In this paper, we introduce CodeFuse-CR-Bench, a comprehensiveness-aware CR benchmark to fill this gap.
在本文中，我们推出了 CodeFuse-CR-Bench，这是一个全面性感知的代码审查基准，旨在填补这一空白。

The design of the benchmark can address the limitations of real-world CR.
该基准的设计可以解决现实世界代码审查的局限性。

Correspondingly, we design an evaluation metric to achieve comprehensive CR evaluation.
相应地，我们设计了一种评估指标来实现综合代码审查评估。

We conduct an empirical study to explore SOTA LLMs' performance in the real-world CR scenario.
我们进行了一项实证研究，以探索 SOTA LLM 在现实世界代码审查场景中的表现。

This more realistic CR benchmark encourages creative automatic CR solutions that can have immediate applicability in open-source software development.
这个更现实的代码审查基准鼓励创造性的自动代码审查解决方案，这些方案可以在开源软件开发中具有即时适用性。

We believe that CodeFuse-CR-Bench will offer valuable evaluation for the future development of LLM-based automatic CR approaches.
我们相信 CodeFuse-CR-Bench 将为基于 LLM 的自动代码审查方法的未来发展提供有价值的评估。