SeRe：与真实世界审查活动相一致的安全相关代码审查数据集

Zixiao Zhao Yanjie Jiang* Hui Liu 教育部北京大学高可信软件技术教育部重点实验室；北京大学软件与微电子学院、北京大学计算机科学学院与北京大学智能技术学院 中国北京；天津大学计算机学院；北京理工大学 2301111987@stu.pku.edu.cn yanjiejiang@tju.edu.cn liuhui08@bit.edu.cn

Kui Liu 华为软件工程应用技术实验室 中国 brucekuiliu@gmail.com

Lu Zhang 教育部北京大学高可信软件技术教育部重点实验室；北京大学软件与微电子学院 中国北京 zhanglucs@pku.edu.cn

arXiv:2601.01042v1 [cs.SE] 3 Jan 2026

摘要

软件安全漏洞可能导致严重后果，因此及早检测至关重要。尽管代码审查是防御安全缺陷的关键机制，但由于对安全问题关注不足或审查者缺乏相关专业知识，现有的安全相关反馈仍然十分有限。现有数据集和研究主要聚焦于通用代码审查评论，要么缺少安全专用标注，要么规模过小，无法支持大规模研究。为弥补这一空白，我们引入**SeRe**，这是一个**安全相关代码审查数据集**，通过基于主动学习的集成分类方法构建。所提出的方法通过人工标注迭代优化模型预测，在保持合理召回率的同时实现了较高的精确率。利用微调后的集成分类器，我们从373,824个原始审查实例中提取了6,732条安全相关评审，确保其在多种编程语言中具有代表性。统计分析表明，SeRe总体上**与真实世界的安全相关审查分布相一致**。为评估SeRe的实用性以及现有代码审查评论生成方法的有效性，我们以安全相关反馈生成为目标，对最先进的方法进行了基准测试。通过发布SeRe及其基准测试结果，我们旨在推动自动化安全导向代码审查领域的研究，并促进更有效的安全软件工程实践的发展。

CCS 概念

• **软件及其工程 → 软件演化**。

关键词

代码审查评论，安全，数据集

*通讯作者。

允许免费为个人或课堂使用而制作本作品全部或部分内容的数字或纸质副本，前提是这些副本不得用于营利或商业利益，也不得为此而制作或分发，并且副本须包含本声明及首页上的完整引文。必须尊重本作品第三方组件的版权。其他用途请联系所有者/作者。*ICSE '26*，里约热内卢，巴西© 2026年 版权所有者/作者所有。ACM ISBN 979-8-4007-2025-3/26/04
https://doi.org/10.1145/3744916.3764557

ACM 引用格式：
Zixiao Zhao、Yanjie Jiang、Hui Liu、Kui Liu 和 Lu Zhang。2026年。SeRe：一个与真实世界审查活动相契合的安全相关代码审查数据集。收录于 *2026IEEE/ACM 第48届软件工程国际会议（ICSE '26），2026年4月12日至18日*，巴西里约热内卢。ACM，纽约，纽约州，美国，12页。
https://doi.org/10.1145/3744916.3764557

1 引言

安全是现代软件开发中的一项根本关切，因为安全漏洞可能造成严重的财务和运营后果 [34]。由于后期修复往往成本高昂且复杂，因此在开发过程的早期阶段（例如代码审查时）解决安全问题至关重要。代码审查是软件开发中广泛采用的一种实践，通过识别缺陷、落实编码规范并促进知识共享 [44]，在提升软件质量方面发挥着关键作用。在这一过程中，审查者会指出代码中的问题，并与开发者展开讨论；开发者随后会根据反馈完善代码更改。作为一项渐进且持续的过程，代码审查能够以相对较低的成本及早发现潜在的安全缺陷，从而有效降低安全风险。Thompson等人 [41] 的研究表明，代码审查覆盖率与软件项目中安全漏洞的普遍程度呈负相关，这凸显了评审在提升代码安全方面的重要作用。

然而，在实践中，安全相关评论仅占代码审查反馈的一小部分。例如，Di Biase 等人 [14] 发现，在 Chromium 项目中，只有 1% 的代码审查评论与安全相关。安全反馈之所以如此有限，主要是因为审查者缺乏安全专业知识，以及在代码审查中安全问题的优先级较低 [6]。这给构建高质量数据集带来了挑战，因为安全相关讨论往往分散且缺乏明确标识，难以从一般审查数据中识别和提取。

为提高代码审查的效率，自动生成审查评论已得到广泛研究，但现有的大多数方法侧重于生成通用审查评论，而非针对安全问题的反馈。代码审查数据集，例如

Tufaq, [2], 等人 [2]) 主要构建通用数据集，缺乏详细的安全注释。虽然存在一些小规模的安全相关评审数据集，例如 Yu 等人手动整理的一个包含 614 条安全相关评论的数据集 [45], ，但其规模和多样性有限，阻碍了大规模研究和模型开发。此外，由于缺乏针对性的训练数据，基于现有数据集构建的通用代码审查模型可能难以生成与安全相关的反馈。因此，自动构建高质量且可扩展的安全相关代码审查数据集已成为亟待解决的问题。除了数据集构建之外，评估审查模型在安全相关场景中的有效性仍是一个尚未解决的问题。

为应对这些挑战并推动该领域的研究，本文提出了一种基于主动学习构建安全相关代码审查数据集的方法。如果某个审查实例指出代码中潜在的安全风险，或提出改进代码安全性的建议，则将其视为安全相关实例。这项研究面临的一个根本挑战是缺乏初始带标签数据集，这使得难以以较低成本开发有效的分类器。为缓解数据稀缺问题，我们采用迭代式训练与标注策略，将机器辅助处理与人工反馈相结合。具体而言，我们通过多轮迭代来完善训练数据集，并依据模型预测来指导人工标注数据的选择。最终，我们构建了一个包含 3,641 个已标注实例的训练数据集。利用该数据集，我们训练了一个集成分类器，从包含 373,824 个审查实例的原始语料库中筛选并提取安全相关评审，最终构建出一个涵盖 C语言、C++、C#、Java 和 Go 等编程语言的安全相关评审数据集，其中包含 6,732 条安全相关评审。基于该数据集，我们建立了一个基准，用于评估最先进的通用代码审查评论生成方法和大语言模型生成安全相关反馈的表现。实验结果表明，现有方法在生成通用代码审查评论方面表现良好，但生成安全相关反馈的效果仍然有限，这凸显了该领域有待进一步改进。实现代码和数据集已公开发布于 [2]。

本文的主要贡献如下。

- 我们提出了一种基于主动学习构建安全相关代码审查数据集的框架。
- 我们构建了一个高质量、可扩展的安全相关代码审查数据集，其中包含来自多个高质量代码仓库的 6,732 个实例，涵盖多种安全问题和 5 种编程语言。据我们所知，这是目前规模最大的公开可用安全相关代码审查数据集。
- 我们评估了最先进的代码审查评论生成方法在生成安全相关评论方面的性能，分析了其有效性，并考察了其局限性。

本文其余部分安排如下。第2节介绍安全漏洞和主动学习的背景。第3节详细介绍数据集构建方法。第4节提出我们的研究问题，并介绍相关实验设计。第5节分析实验结果并讨论本研究的局限性。第6节和第7节分别介绍相关工作并总结全文。

2 背景

2.1 安全漏洞

软件安全漏洞是指软件系统中存在的缺陷、弱点或错误，攻击者可利用这些问题导致非预期或有害行为，并可能危及系统或其数据的机密性、完整性、可用性或真实性 [37]。这类漏洞通常源于开发过程中的不良编码实践、设计错误或安全措施不足。为在开发早期检测安全漏洞，研究人员和开发者提出了多种方法，包括静态代码分析、动态代码测试和代码审查 [17]。本文主要关注第三种方法。

为开发者和安全专业人员提供参考框架，人们制定了多项安全标准，包括通用漏洞披露（**CVE**） [43] 和通用弱点枚举（**CWE**） [10]。CVE 是用于识别和编目特定漏洞的标准，而 CWE 则提供了一种分类体系，根据软件漏洞在设计、代码或实现层面的根本弱点来识别和分类这些漏洞。尽管 CWE 力求保持全面，但仍无法涵盖真实场景中遇到的所有类型的软件安全问题。因此，虽然我们将 CWE 列表作为识别安全相关审查评论的指南，但仍聚焦于安全本身这一概念。在实验的偏差分析中，我们借鉴 CWE 分类体系，并通过人工分析代码审查数据，推导出一个经验分类体系。

2.2 主动学习

主动学习 是一种高效的机器学习方法，它利用数据特征或模型反馈，从大量未标注数据中选择最具信息量的样本进行标注 [24, 33]。这一过程在减少对大量人工标注需求的同时，还能提升模型性能。与随机抽样相比，主动学习能够更好地优化决策边界，因此在标注成本高昂或数据分布高度不均衡的场景中尤为有效。

根据数据选择标准，常见的主动学习方法通常分为两类：**基于不确定性**的方法和**基于代表性**的方法。基于不确定性的主动学习侧重于选择模型预测置信度较低的样本。例如，它可以优先选择置信度分数最低的数据点，以优化决策边界；也可以选择多个模型预测结果相互冲突的实例，从而利用不同模型提供的互补信息。相比之下，基于代表性的主动学习强调数据多样性，确保所选样本既对模型具有挑战性，又能代表整体数据分布，从而减少偏差并缓解过拟合。一个典型例子是基于核心集的方法 [35]，该方法通过求解 k中心聚类目标，将主动学习表述为子集选择问题。

在这项工作中，我们结合了基于不确定性和基于代表性的主动学习策略。具体而言，我们使用多个模型进行决策，并选择模型意见不一致的实例进行人工标注，确保模型在不确定性最高的区域获得监督。同时，由于正类样本相对稀少，我们保留所有被所有模型一致判定为正类的实例，以确保少数类得到充分覆盖。该策略有助于缓解类别不平衡造成的学习偏差，从而提升模型在安全相关代码审查分类任务中的能力。

3 数据集构建

3.1 概述

我们提出了一种数据集构建方法（集成分类器），以有效识别安全相关审查评论。概述如图1所示。图中的实线表示控制依赖，虚线表示数据依赖。该方法包含三个部分：**数据准备**、**伪标签生成**和**迭代式主动学习**。该方法以从代码审查平台爬取的原始代码仓库为输入，输出为微调后的集成分类器。工作流程如下：

(1) **数据准备**。我们从 GitHub 上维护良好、具有悠久维护历史的代码仓库中挖掘代码审查数据，构建初始数据库。(2) **伪标签生成**。我们从初始数据库中随机抽取代码审查，并通过本地轻量级模型与远程重量级专家模型协作标注，生成初始**训练集**。(3) **迭代式主动学习**。**分类器微调**使用**训练集**对集成分类器进行微调。随后，**查询选择**从数据库中提取数据批次，并依据微调后分类器的预测结果选择信息量高的样本。接着，将这些样本交由**人工标注**进行标记，进一步完善**训练集**。这一迭代过程持续进行，直至分类器达到可接受的精确率和召回率阈值。

基于上述步骤，我们开发了一个微调后的集成分类器。借助该分类器，我们可以从数据库中筛选出安全相关审查评论。

3.2 数据准备

3.2.1 数据选择与筛选标准。在本研究中，我们选择**GitHub** [1] 作为数据来源，因为它拥有庞大的开发者社区、丰富的代码仓库资源，且与以往研究保持一致。我们挖掘使用 C语言、C语言、C#、Java 和 Go 编写的代码仓库，因为这些语言在实际软件开发中更可能涉及安全相关问题。为确保数据质量，我们主要关注每种语言中按星标数排名前 10,000 的代码仓库，并筛除拉取请求少于 1,000 个的代码仓库。为确保数据质量与合法性，我们仅保留采用流行且可分发许可证的代码仓库[1]。

[1]Apache-2.0、MIT、GPL-3.0、BSD-2.0、BSL-1.0 等许可证。

3.2.2 抓取工具与评审存储。我们使用**GitHubREST API**以及由 Heumüller 等人 [18] 开发的现有工具**ETCR**，从 GitHub 抓取代码审查评论。我们以数据库格式存储抓取的拉取请求、提交、代码审查评论及相关文件信息，以便灵活查询和提取数据。每个拉取请求通常包含多个提交，而每次提交可能影响多个文件。为优化存储，我们在数据库中记录每个拉取请求之前的文件状态，以及每次提交所做的具体更改。此设计使我们能够在后续数据集构建过程中，灵活检索与特定代码审查评论对应的文件在更改前后的内容和状态。

代码审查通常涉及多轮对话，开发者与审查者在审查过程中的互动会产生有价值的反馈。我们保留这些完整的对话记录，以便数据用户在需要时访问。该爬取工具具有可扩展性，能够适应 GitHub 仓库的更新，因此我们构建的数据集也具有可扩展性。

3.2.3 数据集提取与构建。 为便于使用数据集，我们开发了一种灵活的数据提取机制，提取的数据格式为五元组：

$$\langle hunk_{diff}, file_{old}, file_{new}, comments, refinement \rangle$$

其中， $hunk_{diff}$表示审查者评论所针对的具体代码变更片段； $file_{old}$表示变更前文件的旧版本； $file_{new}$表示变更后文件的新版本； $comments$表示开发者与审查者之间的完整对话； $refinement$表示基于审查评论对原始变更进行的优化，通常是采纳审查反馈后对文件所做的变更。

在数据库中，我们记录了每次提交的差异信息，以及每个拉取请求开始时文件的初始状态。按照时间顺序将每个差异应用于基础文件的内容后，我们可以推导出文件在特定时间点的状态（即 $file_{old}$和 $file_{new}$）。我们将同一拉取请求中审查评论之后对代码片段进行的最近一次修改视为优化修改。

3.3 伪标签生成

在数据标注过程中，我们的主要目标是构建一个具有清晰分类边界的初始训练数据集，以促进分类器的早期训练。该数据集应包含明显的正负样本，使其易于区分，从而让分类器能够快速学习数据特征。为高效实现这一目标并减少人工标注工作量，我们采用了模型辅助伪标签生成方法，并结合两阶段模型筛选机制进行初步分类。

在第一阶段，我们从数据库中随机抽取数据样本，并使用多个轻量级开源模型进行**零样本分类**。这些模型通常是大语言模型的蒸馏版本或低参数版本，计算开销较低，对硬件资源的要求也较少，因此适用于成本效益较高的数据筛选任务。通过利用这些模型多样化的知识库（它们可能经过不同数据集的训练），我们可以

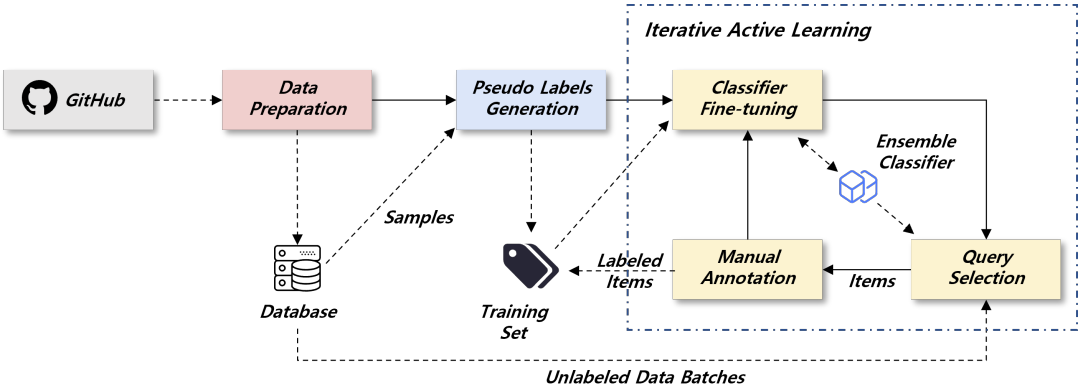

图1：概览

降低模型幻觉等不稳定因素带来的影响。

在第二阶段，系统会汇总轻量级模型的预测结果，以确定正样本和负样本。只有当所有模型一致将某个数据实例分类为正样本（或负样本）时，才会将其视为正样本（或负样本）。随后，这些分类明确的数据点及其对应的伪标签会提交给**专家模型**进行验证。专家模型通常参数更多、性能更强，因此能够提供更可靠的分类结果。

现实世界中的安全相关代码审查实例相对少见。为避免不平衡初始数据集，并确保模型性能不偏向多数类，我们对经验证的带标签样本进行**平衡采样**，力求使正样本与负样本的比例接近 1:1。

3.4 迭代式主动学习

伪标签数据集用作初始训练集，我们基于指令微调来训练集成分类器。为了更好地适应特定的分类任务，我们将伪标签过程中使用的本地模型作为基础模型。每个模型最终层的参数均解除冻结，并添加一个分类头。训练目标是使模型能够根据给定的指令和评论内容，准确判断代码审查评论是否与安全相关。

为了评估每个模型的性能，我们从原始数据库中随机划分出一个验证数据集，并人工为其标注正类和负类。该验证数据集的分布与真实世界代码审查数据的分布一致。

伪标注数据集的特点是分类边界清晰，且模糊实例数量有限。然而，在该数据集上训练的模型可能难以区分困难案例，因而无法达到预期的阈值。这时，迭代式微调便派上了用场。具体而言，我们从数据库中提取未标注数据批次，将其分别输入各个分类器，并将分类结果分为三组：（1）全部预测为正类；（2）模型预测不一致；（3）全部预测为负类。对于

```
# Task description
Please determine whether the reviewer has
explicitly identified security issues or
stability issues in the code implementation,
or explicitly suggested improvements to
enhance the security or stability of the code.

# Code review input
[CODE REVIEW COMMENT]
```

图2：用于分类的提示模板

预测不一致的样本（视为困难案例），我们会将其提交给人工标注员。在早期阶段，预测不一致的样本数量较多，因此我们采用简单随机抽样，选取一部分样本进行标注。样本量按照以下公式确定 [11]:

$$n = \frac{N \cdot Z^2 \cdot p \cdot (1-p)}{e^2 \cdot (N-1) + Z^2 \cdot p \cdot (1-p)}$$

其中，n 表示所需样本量，N 表示总体总量，Z 对应所选置信水平的 Z 值，p 表示总体中具有某一特定特征的比例，e 表示可接受的抽样误差。

鉴于正样本稀缺，确保有足够数量的正实例对于维持训练数据集的平衡至关重要。因此，模型预测为（1）“全部预测为正”的数据会经过人工确认并予以保留，而预测为（3）“全部预测为负”的数据则会被丢弃。随后，在每次迭代中，人工标注的数据会被纳入训练数据集，用于重新训练模型。

在**伪标签生成**和**迭代式主动学习**的整个过程中，我们始终使用一致的提示词，以确保不同阶段的分类任务遵循相同的输入格式并保持语义一致性。具体提示词内容见 2。我们只提及安全这一概念，不对其本身作出解释。这种设计保留了

模型对安全的理解，避免引入人为偏差。

3.5 实现

3.5.1 数据标注。我们邀请了 6 名精通软件安全且具有软件审查经验的志愿者参与标注过程。标注过程详述如下：首先，标注人员熟悉安全编码指南（例如 OWASP），以建立安全相关性的内涵定义。CWE 分类体系则用作外延定义——如果某个实例能够明确映射到某个 CWE类别，就将其标记为安全相关。如果无法明确映射到 CWE，则依据内涵定义作出标注决策。每名志愿者独立为审查条目标注正面或负面标签。出现分歧时，志愿者会进行讨论，直至达成共识。我们使用 Fleiss' Kappa 统计量 [20] 来衡量不同标注人员之间的标注一致性。通过计算 Fleiss' Kappa系数（0.88），我们发现标注人员之间的一致性较高，表明人工标注的准确率和可靠性令人满意。

3.5.2 数据划分与分类器训练。 我们将数据库中的全部未标注数据集随机划分为100个批次，每个批次的大小设为3,776。我们使用前15个批次进行迭代训练，并随机预处理另外2个批次，分别作为验证集和测试集，其规模分别为3,743和3,727。遵循传统的训练原则，验证数据集用于模型训练，测试数据集用于评估模型的泛化能力。由于需要考虑上下文大小限制、分类器性能和标注成本，对于对话式评论，我们仅关注审查者的第一条评论进行分类。这一设置适用于模型训练、推理和人工标注。我们采用一组多样化的先进模型：Gemma-2-9B-IT [38], Meta-Llama-3-8B-Instruct [3], DeepSeek-7B-Chat [12], Ministral- 8B-Instruct-2410 [26], 和Qwen2.5-7B-Instruct [40] 作为基础模型。选择这些模型是因为它们性能出色，并且在开源模型中具有代表性。我们使用**DeepSeek-V3** [13]作为专家模型，构建了一个包含3,000个样本的初始伪标注训练数据集，并保持正样本与负样本1:1的均衡比例。训练阶段的训练轮数和学习率分别设为30和0.00005。我们使用AdamW优化器，并进行500个预热步的线性预热。在协作迭代过程中，共标注了3,641个训练样本。由于训练过程中需要较高的精确率，我们将评估函数调整为动态加权F1分数，其定义如下：

$$\mathrm{F1}_{\mathrm{dynamic}} = \frac{(1+\alpha)\cdot(w_p\cdot\mathrm{precision})\cdot(w_r\cdot\mathrm{recall})}{w_p\cdot\mathrm{precision}+\alpha\cdot w_r\cdot\mathrm{recall}}$$

其中，α是一个超参数，在我们的实验中设为 2；w_p和 w_r 是阈值惩罚参数，用于惩罚较低的精确率和召回率值。对于抽样公式中的参数，我们将 Z 值设为 1.96，对应 95% 的置信度；将 p 设为 0.5，以使样本量最大化；将 e 设为 0.05。

3.5.3 查询策略。 为确定推理过程中最有效的查询策略，我们在验证集上评估了多种投票方案，并比较其对模型性能的影响。基于这些实验结果，我们选择五选四策略作为最优方案——仅当 5 个模型中至少有 4 个模型达成一致时，才将样本标记为正例；否则标记为负例。值得注意的是，该策略在分类器评估和数据集构建中始终保持一致。

4 研究设计

4.1 研究问题

为了评估所提出数据集的质量，以及现有代码审查评论生成方法生成与安全相关的审查评论的能力，我们研究了以下研究问题：

RQ1：代码审查评论中有多少比例与安全相关？它们是如何分类的？

RQ1 是本研究的基础。由于与安全相关的代码审查评论十分稀缺，手动构建此类数据集颇具挑战。为了更好地分析所构建数据集的组成，我们对真实场景中的与安全相关的代码审查评论进行分类。

RQ2：所提出的方法在识别安全相关代码审查评论方面有多准确？与现有方法相比表现如何？

由于数据集的构建依赖于分类方法，我们旨在评估其精确率和召回率，以确保数据集的可靠性。

RQ3：Sere 数据集与真实场景相比如何？它是否保持了较高的精确率并引入较低的偏差？

我们旨在通过基于抽样的评估验证数据集的质量，主要关注精确率和代表性等评估指标。

RQ4：最先进的代码审查评论生成方法在 Sere 数据集上的表现如何？

RQ4 使用我们的大规模数据集评估安全相关代码审查自动生成领域的最新水平。评估结果为进一步研究安全相关代码审查自动生成提供了动力。

4.2 实证类别体系

根据对测试数据集中正实例的观察和总结，我们构建了一个包含 14 个类别的分类体系。不同于以往依赖预定义类别收集数据的研究，我们通过将评审内容映射到 CWE 标签或安全编码关键词，并进行语义分组，从实际数据中提炼出类别。类别命名力求简洁且覆盖广泛，以支持下游评估。随后，人工标注员对安全相关数据进行标注，并赋予相应的类别标签。如有意见分歧，志愿者会通过讨论达成共识。计算得出的卡帕系数为 0.81。我们将这个标注子集称为**代表性集合**，并将其用于**RQ1** 和**RQ3**。

4.3 分类器评估

我们首先评估集成分类器在 Yu 等人[45] 和 Paul 等人[31], 分别手动构建的数据集上的召回率。随后，我们将以下方法与分类器进行比较：基于收集到的 105 个安全相关关键词进行关键词匹配，这是一种相关工作中常用的传统技术[30, 31, 45]；以及大语言模型，包括**GPT-4o** [28], **Gemini-1.5-Pro** [39], 和 **DeepSeek-V3**，并使用图2所示的提示词。我们采用准确率、精确率、召回率和 F1 分数作为评估指标。这些指标分别反映集成分类器的总体正确性、识别安全相关评论的精确程度，以及对实际安全相关评论的覆盖率。

4.4 基于抽样的数据集分析

我们首先从数据集中随机抽取 447 个项目（确保置信度为 95 %），并按照第 3.5.1 节所述流程进行人工标注。标注人员独立将每个样本标注为与安全相关或无关，并通过讨论解决分歧。随后，我们根据标注结果计算数据集的精确率。为评估构建的数据集可能存在的偏差，我们选择费希尔精确检验 [15]，因为该检验在处理小型数据集时能提供可靠的评估结果，非常适用于稀疏的安全相关代码审查数据场景。我们基于抽样进行检验。每次从构建的数据集中随机抽取一组样本，其规模与代表性集合相同。随后，按照代表性集合的标注流程，对样本进行安全类别的人工标注，并对标注后的样本数据与代表性集合进行费希尔精确检验。我们重复此过程三次，并记录每次检验中各子类别的 p 值。我们设定显著性水平 $\alpha = 0.05$，这意味着，当零假设（抽样数据与代表性集合之间不存在分布差异）实际上为真时，我们接受有 5% 的概率错误地拒绝该假设。

4.5 安全相关代码审查的生成

4.5.1 研究对象选择。在 RQ4 评估阶段，我们重点考察最先进的开源代码审查评论生成模型的表现，并将其作为基线：**AUGER** [21], **DISCOREV** [5], **LLaMA- Reviewer** [25] 和**CodeReviewer** [22]。对于 CodeReviewer、LLaMA-Reviewer 和 AUGER，原作者提供了预训练模型检查点，我们直接使用这些检查点进行实验。具体而言，LLaMA-Reviewer 模型包含两个微调版本：前缀微调和 LoRA 微调。对于 DISCOREV，作者提供了完整的可复现软件包，使我们能够基于原始代码和训练设置复现该方法，并评估复现模型。由于 AUGER 专为 Java 设计，为确保公平比较，我们选择 SeRe 中的 Java 语言场景作为 AUGER 的评估数据集。大语言模型（LLM）在代码生成和审查等多种软件工程任务中展现出强大能力。然而，它们在生成与安全相关的代码审查评论方面的效果仍不明确。因此，我们选择**GPT-4o** 和**DeepSeek-V3**作为具有代表性的大语言模型，并评估它们在零样本和少样本设置下的表现。

```
# Task description
You are a senior code reviewer specializing in
code security. Below is a code patch (in git diff
format) and the context of the file where the
patch is applied. Directly provide a code review
comment focused exclusively on identifying
potential security issues or suggesting
improvements to enhance the security of the code.
```

```
# Task input
Patch: [CODE PATCH]
Context: [CODE CONTEXT]
```

```
# Task output
Review Comment:
```

图 3：用于零样本评论生成的提示

4.5.2 数据预处理。为了更好地评估这些代码审查评论生成方法，我们按照 Tufano 等人提出的方法对数据集进行预处理 [42], , 包括移除非英语数据、替换评论中的链接，以及剔除过长的代码更改或上下文。值得注意的是，我们移除了代码更改词元数超过 2,048 或上下文词元数超过 10,000 的样本。最终，我们从原始的 6,731 个数据点中整理出一个包含 4,788 个评估样本的基准数据集。由于本次评估旨在考察这些方法的单轮生成表现，我们仅保留对话数据中审查者的第一条评论。

4.5.3 提示设计。 我们针对零样本和少样本设置，为大语言模型设计了提示策略。在实际的少样本评估中，我们从与原始数据所属代码仓库不同的代码仓库中选取两个样本作为少样本示例。由于上下文信息篇幅较长，我们选择不将其纳入少样本示例，以避免可能削弱与任务相关的特征，并确保向模型提供更集中的学习信号。我们使用的提示模板如图 3 和图 4 所示。

4.5.4 评估指标。我们使用多项评估指标来评估评论生成方法的性能，包括 **BLEU** [29],**ROUGE- L** [23], **METEOR** [4], **精确匹配**和 **语义相似度**。BLEU、ROUGE-L、METEOR 和精确匹配是标准评估指标，用于衡量生成评论与真实标注之间相似度的不同方面。BLEU 和 ROUGE-L 主要衡量词汇和结构上的相似度；METEOR 通过词干提取、同义关系和释义识别来纳入语义匹配；精确匹配则评估生成的评论是否与参考评论完全一致。对于语义相似度，我们使用 **Sentence-BERT (SBERT)** [32] 为生成评论和参考评论分别生成嵌入向量，然后计算这些嵌入向量之间的**余弦相似度**，以衡量其语义相似度。这种方法能够捕捉生成评论与参考评论在语义上的接近程度，克服传统基于词元的评估指标的局限性，因为它关注的是含义，而非词语是否完全匹配。

```
# Task description
You are a senior code reviewer specializing in
security. Below are [N] examples of code patches and
their corresponding review comments focusing only on
security-related concerns.
After the examples, you will find a new code patch and
its context. Please provide a detailed code review
comment for the new patch, focused exclusively on
identifying potential security issues or suggesting
improvements to enhance the security of the code.

# Examples
Patch: [CODE PATCH]
Review Comment: [CODE REVIEW COMMENT]
                                                  ×N
# Task input
Patch: [CODE PATCH]
Context: [CODE CONTEXT]

# Task output
Review Comment:
```

图4：用于少样本评论生成的提示词

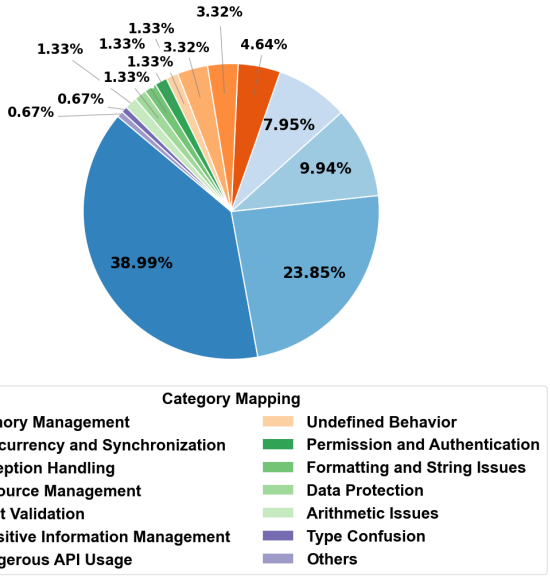

图 5：安全相关代码审查评论在 14 个子类别中的分布。

4.6 实验环境

我们的实验在 Ubuntu 22.04.5 LTS 操作系统上进行，训练框架使用 PyTorch 实现。硬件配置包括两块 NVIDIA A800 80GB GPU，用于支持模型训练和推理。对于所有需要访问大语言模型的操作，我们均使用官方 API。

5 结果分析

5.1 RQ1：安全相关评论的比例与分类

在 RQ1 中，我们考察了代码审查反馈中安全相关评论的比例及其内部构成。通过深入分析一个包含 3,727 条代码审查评论的代表性数据集，我们识别出 151 条安全相关评论，占评论总数的 4.05%。

我们对安全相关数据进行了人工分析，并制定了一套经验分类体系，将安全相关评论划分为 14 个类别。这些类别包括内存管理、并发与同步、异常处理、资源管理（指对文件句柄或网络连接等非内存资源的管理）、输入验证、敏感信息管理、危险 API 使用、未定义行为（指代码行为未由语言或平台规范明确定义）、权限与身份验证、格式与字符串问题、数据保护、算术问题、类型混淆以及其他（包括不符合预定义子类别的评论，或安全问题过于模糊、无法归入特定类别的评论）。

图 5 展示了这些类别在安全相关评论中的分布。结果显示，**内存管理**（38.99%）和**并发与同步**（23.85%）是最常见的类别，合计占安全相关评论的 60% 以上。其他值得注意的类别包括异常处理（9.94%）和资源管理（7.95%）。相比之下，算术问题、类型混淆等类别所占比例很低，各自均不足 2%。这种分布凸显了代码审查中发现的常见安全问题模式，尤其是对内存管理和并发问题的频繁关注；这些问题在开发过程中往往容易被忽视，却在代码审查中受到重点关注。

表1：不同方法的性能比较

方法	精确率	召回率	准确率	F1
关键词	0.1357	0.1258	0.9321	0.1306
Gemini-1.5-Pro	0.5482	0.7152	0.9646	**0.6207**
GPT-4o	0.4158	**0.7682**	0.9469	0.5395
DeepSeek-V3	0.5968	0.4901	0.9659	0.5382
集成	**0.9275**	0.4238	**0.9753**	0.5818

> **RQ1 的回答**
>
> 与安全相关的代码审查评论约占评论总数的**4%**，其中大多数主要涉及内存管理和并发同步问题。

5.2 RQ2：数据集构建方法的性能

Yu 等人[45] 和 Paul 等人[31]提出的数据集上的召回率分别为 75% 和 85%。如表1所示，实验结果表明，我们的集成方法取得了最高的精确率（92.75%）和准确率（97.53%），显著优于传统的（关键词匹配）和基于大语言模型的

方法。值得注意的是，高精确率对于确保数据集的可靠性和质量至关重要，因为它能最大限度地减少假阳性。相比之下，GPT-4o 和 Gemini-1.5-Pro 的召回率较高（分别为 76.82% 和 71.52%），但精确率明显较低（分别为 41.58% 和 54.82%），这表明在识别安全相关评论时出现了更多假阳性。关键词匹配方法的表现最差，其精确率（13.57%）和召回率（12.58%）均显著低于其他方法。

这些结果验证了我们的集成方法在精确率和准确率方面的优势，有效减少了假阳性，并确保了构建的数据集的质量。召回率相对较低，反映了我们有意优先考虑精确率的设计选择，这可能会导致遗漏一些安全相关评论。这种权衡是有意为之，因为我们的主要目标是构建高质量的安全相关数据集，而非最大限度地提高分类器的召回率。与现有方法相比，我们的方法展现出更高的精确率，验证了集成学习和人工标注在提升分类性能方面的有效性。此外，GPT-4o 和 Gemini-1.5-Pro 较高的召回率表明，大语言模型有潜力识别潜在的安全相关评论，尽管其较高的假阳性率限制了它们在实际应用中的可靠性。

为进一步研究召回率与精确率之间的权衡，我们采用替代策略抽取最优投票策略遗漏的安全相关审查评论。随后，我们分析这些假阴性的特征。最常见的遗漏案例类型包括：**表单交错**（代码与自然语言交错，导致安全上下文被掩盖）；**主题交错**（安全问题与非安全问题混杂，使其不那么突出）；以及**罕见类别**（代表性不足的安全问题类型，通常取决于不常见的特定代码库上下文）。

RQ2 的回答

我们的方法在保持可接受召回率的同时实现了较高的精确率，从而确保构建的数据集具备良好的质量和稳健性。

5.3 RQ3：数据集质量分析

我们从四个角度进行分析我们的数据集的精确率，并简要比较我们的数据集与现有安全相关数据集。基于**RQ1**中的类别，我们分析数据集的构成及其潜在偏差。

人工评估结果表明，94% 的样本被正确标注，这说明最终的数据集具有较高的可靠性。如果一条代码审查记录映射到某个 CWE 类别，我们就将其视为真实漏洞。基于抽样的验证显示，84% 的安全相关评审对应真实漏洞。其余内容包括不安全实践或未映射的漏洞。随后，我们从数据集规模、多样性、偏差和可扩展性等方面，将 **Sere** 与现有的安全相关代码审查数据集进行比较。Sere 包含 6,732 条评审记录，远多于 Yu 的数据集（614 条）和 Paul 的数据集（516 条）。Sere 来源于 GitHub 上 311 个高星级、多语言项目，

表 2：从原始数据到构建的数据集：不同编程语言的评审数据分布

语言	#代码仓库		#评审		#评论	
	Raw	筛选后	Raw	筛选后	Raw	筛选后
C	62	55	37,555	1,534	74,120	3,312
C#	90	75	103,545	1,454	198,929	3,436
C++	53	51	81,910	1,269	154,516	2,824
Go	103	88	92,471	1,358	177,429	3,087
Java	51	42	58,343	1,117	107,114	2,473
总计	359	311	373,824	6,732	712,108	15,132

而 Yu 的数据集取自 OpenStack 和 Qt 社区中的 4 个项目，Paul 的数据集则来源于 ChromiumOS 项目。Yu 和 Paul 的数据集受到预设安全相关关键词所引入的偏差影响。相比之下，Sere 基于严谨的安全定义构建，不会预先偏向特定类型的安全问题。Yu 和 Paul 的数据集高度依赖人工标注和事后人工审查，而 Sere 的构建过程则更加自动化，因此具有更强的可扩展性。

我们的数据集源自开源项目，涵盖多种编程语言，包括 C语言、Go、C#、Java 和 C++，这些语言在开源社区中具有广泛的代表性。表 2 展示了原始数据集和我们构建的数据集中，不同编程语言对应的代码仓库、代码审查条目以及单条评论的分布情况（单个代码审查条目可能包含多条评论）。这些语言之间的代码仓库、评审和评论分布相对均衡，没有任何一种语言在数据集中占据压倒性优势。构建的数据集中不同语言的相对比例总体上与原始数据集保持一致。这表明，我们的筛选标准并未对特定语言产生不成比例的偏向或排除，在确保获得聚焦于安全相关代码审查数据的子集的同时，保留了原始数据集的多样性和代表性。

为验证构建的数据集的分布是否与真实场景相符，我们从构建的数据集中进行了多次抽样，在表 3 中分别记为 S.1、S.2 和 S.3，并使用费希尔精确检验评估样本与代表性集合在类别分布方面的一致性。结果表明，对于与**并发与同步** 和**异常处理**相关的类别，p 值明显偏小，说明这些类别可能存在细微的分布偏移。不过，尽管存在这些潜在差异，检验仍顺利通过，且未拒绝零假设，表明没有明确证据显示分布存在显著差异。对于大多数其他类别，p 值显著高于预先设定的 0.05 显著性水平。这进一步证实了数据的可靠性以及零假设的稳健性，也表明构建的数据集的分布与真实世界的安全相关代码审查数据高度吻合。

表 3：组件偏差分析结果

类别	检验结果（p 值）		
	S.1 S.2	S.3	
格式与字符串问题	1.0	1.0	1.0
类型混淆	1.0	1.0	1.0
权限与身份验证	1.0	1.0	1.0
未定义行为	1.0	1.0	0.68
输入验证	1.0	0.62	0.77
内存管理	0.73	0.55	0.48
并发与同步	0.51	0.06	0.12
资源管理	1.0	0.34	0.65
危险 API 使用	0.21	0.72	0.45
敏感信息管理	1.0	1.0	0.21
算术问题	0.68	0.1	0.45
异常处理	0.54	0.12	0.07
数据保护	1.0	0.5	1.0
其他	1.0	1.0	1.0

RQ3 的答案

构建的数据集的偏差可以忽略不计，且与真实世界数据分布高度吻合，从而确保了其代表性和实际适用性。

5.4 RQ4：审查生成方法的性能

表 4 总结了我们的基准上各代码审查评论生成方法的性能。为简洁起见，我们将前 5 个模型称为“通用代码审查模型”，并将基于 DeepSeek 和 GPT 的生成称为“基于大语言模型（LLM）的方法”。对于通用代码审查模型，括号内报告的是它们在原论文所用测试数据集上的性能，以供比较。在多编程语言场景下，LlamaReviewer 的前缀版本取得了最高的 BLEU 分数（3.67）和最高的 ROUGE-L 分数（10.25%），而基于零样本提示的 DeepSeek-V3 在 METEOR（13.7%）和语义相似度（39.82%）方面表现最佳。然而，所有方法在精确匹配上的表现都很差，结果几乎为零，这表明它们难以生成与真实标注完全一致的评论。

总体而言，通用代码审查模型在词汇相似度方面表现良好，例如 BLEU 和 ROUGE-L。这可能是因为这些方法在训练过程中主要依赖此类评估指标作为评估标准，因此生成的评论能够很好地符合真实世界的代码审查格式。相比之下，大语言模型在 METEOR 和语义相似度等基于语义的指标上表现更佳，但在词汇相似度方面则有所不足。这可能是由于缺少微调，导致其用词更加多样，同时仍能准确表达语义。尤其是，大语言模型在零样本和少样本设置下的表现差异很小；在大多数情况下，零样本设置的表现略优于少样本设置。这表明，提供额外示例未必能提升大语言模型在此任务中的表现。

应用于安全相关评估数据集时，与公开的实验结果相比，通用代码审查模型的表现明显下降，这表明它们在处理特定安全语境方面存在局限。这主要是因为安全相关数据十分稀缺，仅占训练集的一小部分，导致模型无法有效处理此任务。此外，这些方法还受到参数容量和知识范围的限制：若没有安全相关数据用于微调，它们就无法将相关知识迁移到此任务中。相比之下，得益于丰富的训练语料库，大语言模型展现出较强的迁移能力。然而，大语言模型生成的评论与人工审查者撰写的评论之间仍存在显著差距。

示例说明。为了清晰、直观地了解不同方法在安全相关代码审查评论生成中的效果，我们给出了图6所示的一个简单示例。受篇幅限制，我们聚焦于任务输入中的 $hunk_{diff}$ 部分。由于大语言模型通常会生成冗长的回复，我们提取并展示了其中与真实标注最相关的部分。此外，由于 DeepSeek（零样本提示和 2样本提示）的回复十分相似，我们只展示其中一个。

原始代码存在潜在的安全问题，即资源释放不当。真实标注指出，如果 reader != nil 且 err != nil，就不应关闭 reader。DIS-COREV 专注于重新组织函数调用，却忽略了核心安全问题。LLaMA-Reviewer（Prefix）对 defer 语句的位置表示不确定，而 LLaMA-Reviewer（Lora）则完全忽略了安全问题，直接照搬代码注释。CodeReviewer 对是否有必要关闭 reader 提出了疑问，却没有指出具体的安全漏洞。DeepSeek-V3（零样本提示/2样本提示）承认资源管理的重要性，并指出需要幂等的“Close”调用，但未能识别出由于提前退出而可能导致资源未能正确关闭的具体情形。GPT-4o（零样本提示）关注的是未定义行为，而没有解决特定条件下的资源释放问题。只有 GPT-4o（2样本提示）提出了直接应对提前退出潜在风险的方案，建议在 defer Close() 前检查 reader，以确保资源得到妥善管理。

RQ4 的回答

大语言模型在语义表达方面准确率更高，而通用代码审查生成方法的用词则更为精确，这表明这两种方法在生成有效的安全相关反馈方面都有改进空间。

5.5 有效性威胁

内部效度。 首要威胁是数据泄漏，即评估基准可能包含与受评方法训练集重叠的数据。为缓解这一问题，我们对数据进行预处理，使其形式可能与原始训练数据存在显著差异，参见 [42]。此外，Cao 等人 [7] 发现，受污染的数据并不总会影响模型性能；在某些情况下，模型在截止日期之后收集的数据上接受评估时，表现甚至会更好

表 4：代码审查生成方法的性能

方法	BLEU	ROUGE-L	METEOR	精确匹配	语义相似度
AUGER	3.67 (15.79)	11.72% (22.04%)	6.25% (18.59%)	0.0% (4.14%)	16.36% (28.43%)
DISCOREV	3.22 (7.42)	9.37% (13.89%)	6.9% (13.51%)	0.00% (0.24%)	16.70% (29.81%)
LLaMA-Reviewer^{Prefix}	3.67 (5.19)	10.25% (11.58%)	8.62% (10.83%)	0.00% (0.0%)	23.50% (24.45%)
LLaMA-Reviewer^{Lora}	3.12 (5.7)	7.82% (10.39%)	5.38% (8.29%)	0.00% (0.16%)	17.24% (22.87%)
CodeReviewer	3.15 (5.44)	7.73% (10.18%)	5.53% (8.29%)	0.02% (0.06%)	22.09% (23.17%)
DeepSeek-V3$^{0\text{-shot}}$	0.98	5.87%	13.7%	0.00%	39.82%
DeepSeek-V3$^{2\text{-shot}}$	0.81	5.03%	12.33%	0.00%	39.43%
GPT-4o$^{0\text{-shot}}$	0.89	5.37%	13.43%	0.00%	38.76%
GPT-4o$^{2\text{-shot}}$	0.66	4.42%	11.78%	0.00%	38.18%

```
@@ -1,48 +1,138 @@
…
-       reader, err := cs.Get(ref)
-       if reader == nil {
-               return nil, errors.New("Chunk not present")
…
+       reader, err := cs.Get(r)
+       if reader == nil && err == nil {
+               return nil, nil
// It's reasonable for a chunk to not be present.
+       } else if err != nil {
+               return nil, err
        }
+       defer reader.Close()
```
Hunk$_{diff}$

Ground Truth: If reader != nil and err != nil, reader will not be closed.

DISCOREV: I think this should be moved up above the `cs.Get` call, so that it's closer to the other functions that need it.

LLaMA-Reviewer(Prefix): I'm not sure this is the right place for this.

LLaMA-Reviewer(Lora): It's reasonable for a chunk to not be present.

CodeReviewer: Why do we need to close the reader here?

DeepSeek(0-shot/2-shot): … The patch introduces a `defer reader.Close()` statement, which is a good practice for resource management. However, ensure that the `Close` method on the `reader` is idempotent and safe to call multiple times, as it might be called in error scenarios as well. This prevents potential resource leaks or unexpected behavior …

GPT(0-shot): … The newly added `defer reader.Close()` ensures proper resource handling, which is good. However, ensure that `reader.Close()` in `cs.Get(r)` is robust and safe to call even if an error occurs during the reader initialization. If `cs.Get(r)` can fail in a way that leaves `reader` partially usable, this might introduce undefined behavior, particularly in cases of nested resources …

GPT(2-shot): … While the `defer reader.Close()` ensures the resource is released eventually, any code executed above the defer statement might cause premature exits (from panics or an interrupt) and leave the resource unclosed. Suggestion: Check whether `reader` is non-`nil` immediately before deferring its `Close()` method to ensure proper lifecycle management in edge cases. …

图6：示例。

。另一项威胁是用于评估大型语言模型的提示词设计。为尽量减少偏差，我们设计的提示词尽可能清晰直接，避免使用可能造成影响的模糊或误导性表述模型的输出。最后，由于数据集构建过程中抽样存在随机性，分类器训练的可复现性令人担忧。不过，我们保留了最终构建的训练数据集，确保其他人能够复现并验证我们的结果。

外部效度。 第一个威胁在于我们的方法和评估中对模型的选择。为此，我们选择了研究社区中普遍使用且有效的模型。另一个潜在威胁来自数据源，因为不同来源可能产生不同结果。为缓解这一问题，我们选择广受认可的代码仓库作为数据源。最后，大语言模型生成的输出也存在可复现性问题，因为其行为可能会受到生成过程中的随机性或其他因素影响。为缓解这一问题，我们保留了大语言模型生成的输出，确保我们的结果可以得到独立验证。

构念效度。 首要威胁是人工标注中个人偏差可能产生的影响。为此，我们通过 κ统计量衡量标注者间一致性，以确保一致性。另一个威胁是代码审查生成评估指标的局限性。我们选择了代码审查评论生成任务中常用的评估指标。未来，我们计划纳入更多评估指标，以更全面地评估模型性能。

6 相关工作

6.1 自动化代码审查生成

代码审查生成自动化已从基于检索的技术发展到先进的深度学习和大语言模型（LLM） [9]。早期方法（如 Gupta 等人提出的 DeepMem [16], ）采用 LSTM 模型检索相关的历史审查评论，而 Siow 等人 [36] 则利用注意力机制增强语义理解。Hong 等人 [19] 在 CommentFinder 中引入了格式塔模式匹配，通过余弦相似度识别相似方法。尽管这些方法有所贡献，但它们受限于对历史数据的依赖，且缺乏生成能力。

深度学习的兴起标志着向生成式模型的转变。Tufano 等人 [42] 利用文本到文本迁移 Transformer（T5），直接根据源代码生成审查评论，展示了预训练模型在这一任务中的有效性。Li 等人 [22] 开发了 CodeReviewer，纳入了四项专为代码审查设计的预训练任务，从而提升了

审查评论生成和代码变更质量估计的效果。AUGER [21] 通过将生成的评论明确关联到具体代码行，进一步提高了精确率。

近年来，大语言模型的发展进一步提升了自动化代码审查的水平。Lu 等人 [25] 提出了 LLaMA-Reviewer，利用参数高效微调（PEFT），以更少的可训练参数实现了具有竞争力的性能。Nashaat 等人 [27] 提出了 CodeMentor，通过融入基于人类反馈的强化学习（RLHF），使模型输出符合领域特定要求。Yu 等人 [47] 提出了 Carllm，通过提供精准的位置、解释和修复建议，着重提升审查评论的准确性与清晰度。Ben 等人 [5] 提出了 DIS-COREV，采用跨任务知识蒸馏整合质量估计、评论生成和代码优化等审查任务，取得了最先进的 BLEU 和 CodeBLEU 分数。这些进展凸显了大语言模型及领域特定适配在提升自动化代码审查质量和相关性方面日益重要的作用。

6.2 安全相关的代码审查

代码审查在软件安全中的作用已得到广泛研究，研究结果突出了其在漏洞检测方面的优势与局限性。Di Biase 等人 [14] 发现，Chromium 的审查评论中只有 1% 涉及安全问题，而且经常忽略特定于语言的问题，例如 C语言++ 内存错误和跨站脚本攻击。Paul 等人 [31] 确定了影响 Chromium OS 安全缺陷检测的因素，包括审查时间、审查者经验和代码复杂度，并强调了纠缠的代码变更和审查者疲劳所带来的风险。

Thompson 和 Wagner [41] 分析了 3,126 个开源项目，发现全面的评审与漏洞减少之间存在很强的相关性，但审查者数量的影响很小。Yu 等人 [45] 表明，竞态条件和资源泄漏经常被标记出来，而跨站脚本攻击和 SQL 注入则常常被遗漏，凸显了对更明确的审查者指南的需求。

为改进以安全为重点的评审，Paul [30] 开发了一个用于识别安全相关评审的分类器，而 Charoenwet 等人 [8] 发现，结合使用静态应用安全测试工具可以提高漏洞检测能力，但会产生 76% 的无关警告。Yu 等人 [46] 评估了安全审查中的大语言模型，结果表明，GPT-4 等模型的表现优于静态分析工具，但在冗长性和逻辑不一致方面仍存在困难，而提示设计会显著影响其性能。

6.3 代码审查数据集

现有研究已提出多个用于研究代码审查的数据集，大致可分为通用数据集和安全相关数据集。对于通用数据集，Tufano 等人 [42] 编制了一个整合 Stack Overflow 讨论内容与 CodeSearchNet Java 数据集的数据集，经预处理后包含超过 148 万个样本。Li 等人 [21] 从 GitHub 上的 11 个 Java 代码仓库构建了另一个数据集，所选项目至少包含 4,000 个拉取请求和 100 名贡献者。除 Java 外，Li 等人 [22] 还推出了一个多语言数据集，涵盖九种编程语言，以及来自 1,000 多个 GitHub 仓库的 790 万个拉取请求，旨在支持审查评论生成等任务。Yu 等人 [47] 进一步改进了标注质量，采用 ChatGPT 进行自动化数据标注和验证。

针对安全相关代码审查，Yu 等人 [45] 从 OpenStack 和 Qt 项目中整理了一个数据集。他们通过基于关键词的提取和人工标注，识别出 614 条安全相关审查评论，并将其分为 15 种安全缺陷类型。该数据集与 CWE 分类体系保持一致，从而增强了其有效性。

7 结论与未来工作

本文介绍了 **SeRe**，这是一个安全相关代码审查数据集，通过一种基于主动学习驱动的集成分类器的自动化方法构建而成。SeRe 包含来自多种编程语言的 6,732 条安全相关评审，与现实世界中的评审分布高度一致，为研究人员和从业者提供了高质量的基准。此外，我们还研究了当前最先进的代码审查评论生成方法的有效性，评估了它们在安全相关语境中的表现。研究结果揭示了这些方法的优势与局限，并突出了安全相关代码审查所带来的挑战。

未来，我们将扩展数据集，以涵盖更多安全问题和项目。此外，我们将研究将数据构建方法应用于其他类型代码审查（例如重构建议）的可行性，以评估其更广泛的实用价值。另外，我们将利用该数据集开发先进的代码审查生成方法，以提升准确率和相关性。

致谢

作者谨向匿名审查者致以谢意，感谢他们提出富有见地的评论和建议。本研究部分得到了国家自然科学基金委员会（62232003、62172037 和 62172214）以及 CCF-华为昇腾基金的资助。

参考文献

[1] 2025. *GitHub*. https://github.com/ [2] 2025. 复现包。https://github.com/caagc/Sere [3] AI@Meta. 2024. Llama 3 模型卡片。(2024)。https://github.com/meta-llama/ll ama3/blob/main/MODEL_CARD.md [4] Satanjeev Banerjee 和 Alon Lavie. 2005. METEOR：一种用于机器翻译评估的自动指标，与人工判断的相关性更高。载于机器翻译和/或摘要的内在与外在评估指标 *ACL* 研讨会论文集。65–72。[5] Oussama Ben Sghaier 和 Houari Sahraoui. 2024. 通过跨任务知识蒸馏改进代码审查连续任务的学习。*ACM*软件工程论文集1, FSE (2024), 1086–1106. [6] Larissa Braz 和 Alberto Bacchelli. 2022. 现代代码审查中的软件安全：开发者视角。载于第*30*届 *ACM* 欧洲软件工程会议与软件工程基础研讨会联合会议论文集。810–821。[7] Jialun Cao、Wuqi Zhang 和 Shing-Chi Cheung. 2024. 担心数据污染？评估代码语言模型中的应对措施。*arXiv* 预印本 *arXiv:2403.16898* (2024)。[8] Wachiraphan Charoenwet、Patanamon Thongtanunam、Van-Thuan Pham 和 Christoph Treude. 2024. 安全代码审查静态分析工具的实证研究。载于第*33*届 *ACM SIGSOFT* 国际软件测试与分析研讨会论文集。691–703。[9] Xiangping Chen、Xing Hu、Yuan Huang、He Jiang、Weixing Ji、Yanjie Jiang、Yanyan Jiang、Bo Liu、Hui Liu、Xiaochen Li 等. 2025. 基于深度学习的软件工程：进展、挑战与机遇。《中国科学：信息科学》68, 1 (2025), 111102。[10] Steve Christey、J Kenderdine、J Mazella 和 B Miles. 2013. 通用弱点枚举。*MITRE*公司(2013)。

[11] William G. Cochran. 1977. 抽样技术，第*3*版。抽样技术，第3版。[12] DeepSeek-AI. 2024. DeepSeek 大语言模型：以长期主义扩展开源语言模型。*arXiv* 预印本 *arXiv:2401.02954* (2024)。https://github.com/deepseek-ai/DeepSeek-LLM [13] DeepSeek-AI. 2024. DeepSeek-V3 技术报告。arXiv:2412.19437 [cs.CL] https://arxiv.org/abs/2412.19437 . Marco Di Biase、Magiel Bruntink 和 [14] Alberto Bacchelli. 2016. 代码审查的安全视角：以 Chromium 为例。载于 *2016 IEEE* 第*16*届国际源代码分析与操作工作会议（*SCAM*）。IEEE，21–30。[15] Ronald Aylmer Fisher. 1970. 面向研究工作者的统计方法。载于统计学的突破：方法论与分布。施普林格，66–70。[16] Anshul Gupta 和 Neel Sundaresan. 2018. 利用深度学习实现智能代码审查。载于第*24*届 *ACM SIGKDD* 知识发现与数据挖掘国际会议（*KDD'18*）深度学习日论文集。[17] Nima Shiri Harzevili、Alvine Boaye Belle、Junjie Wang、Song Wang、Zhen Ming、Nachiappan Nagappan 等. 2023. 利用机器学习和深度学习自动检测软件漏洞的综述。*arXiv* 预印本 *arXiv:2306.11673* (2023)。[18] Robert Heumüller、Sebastian Nielebock 和 Frank Ortmeier. 2021. 充分利用代码审查！更多数据带来更深入的学习。载于第*29*届欧洲软件工程会议与软件工程基础研讨会联合会议论文集。1505–1509。[19] Yang Hong、Chakkrit Tantithamthavorn、Patanamon Thongtanunam 和 Aldeida Aleti. 2022. CommentFinder：更简单、更快速、更准确的代码审查评论推荐。载于第*30*届 *ACM* 欧洲软件工程会议与软件工程基础研讨会联合会议论文集。507–519。[20] Helena Chmura Kraemer. 1980. 卡帕系数的扩展。生物统计学(1980)，207–216。[21] Lingwei Li、Li Yang、Huaxi Jiang、Jun Yan、Tiejian Luo、Zihan Hua、Geng Liang 和 Chun Zuo. 2022. AUGER：利用预训练模型自动生成审查评论。载于第*30*届 *ACM* 欧洲软件工程会议与软件工程基础研讨会联合会议论文集。1009–1021。[22] Zhiyu Li、Shuai Lu、Daya Guo、Nan Duan、Shailesh Jannu、Grant Jenks、Deep Majumder、Jared Green、Alexey Svyatkovskiy、Shengyu Fu 等. 2022. 通过大规模预训练实现代码审查活动自动化。载于第*30*届 *ACM* 欧洲软件工程会议与软件工程基础研讨会联合会议论文集。1035–1047。[23] Chin-Yew Lin. 2004. Rouge：用于自动评估摘要的软件包。载于文本摘要领域的新进展。74–81。[24] Peng Liu、Lizhe Wang、Rajiv Ranjan、Guojin He 和 Lei Zhao. 2022. 主动深度学习综述：从模型驱动到数据驱动。*ACM*计算调查（*CSUR*）54, 10s (2022), 1–34。[25] Junyi Lu、Lei Yu、Xiaojia Li、Li Yang 和 Chun Zuo. 2023. Llama-Reviewer：通过参数高效微调，利用大语言模型推进代码审查自动化。载于*2023 IEEE* 第*34*届国际软件可靠性工程研讨会（*ISSRE*）。IEEE，647–658。[26] Ministraux. 2024. Ministral-8B-Instruct-2410. https://mistral.ai/en/news/minis traux[27] Mona Nashaat 和 James Miller. 2024. 面向自动化软件审查，利用组织数据高效微调语言模型。*IEEE*软件工程汇刊(2024)。[28] OpenAI. 2025. GPT-4o：优化版 GPT-4 语言模型。https://openai.com /index/hello-gpt-4o 访问日期：2025-03-01。[29] Kishore Papineni、Salim Roukos、Todd Ward 和 Wei-Jing Zhu。2002。BLEU：一种机器翻译自动评估方法。载于 计算语言学协会第*40*届年会论文集。311–318。[30] Rajshakhar Paul。2022。Astor：一种识别安全代码评审的方法。载于第*37*届*IEEE/ACM*自动化软件工程国际会议论文集。1–3。[31] Rajshakhar Paul、Asif Kamal Turzo 和 Amiangshu Bosu。2021。为什么安全缺陷在代码评审期间会被忽视？——对Chromium OS项目的病例对照研究。载于 *2021 IEEE/ACM*第*43*届国际软件工程大会（*ICSE*）。IEEE，1373–1385。[32] Nils Reimers 和 Iryna Gurevych。2019。Sentence-BERT：使用孪生BERT网络的句子嵌入。载于 *2019*年自然语言处理实证方法会议论文集。计算语言学协会。https://arxiv.org/abs/1908 10084 。[33] Pengzhen Ren、Yun Xiao、Xiaojun Chang、Po-Yao Huang、Zhihui Li、Brij B Gupta、Xiaojiang Chen 和 Xin Wang。2021。深度主动学习综述。*ACM*计算调查（*CSUR*）54，9（2021），1–40。[34] David Rice。2007。《极客经济学：不安全软件的真实代价》（第1版）。[35] Ozan Sener 和 Silvio Savarese。2017。卷积神经网络的主动学习：一种核心集方法。*arXiv*预印本 *arXiv:1708.00489*（2017）。[36] Jing Kai Siow、Cuiyun Gao、Lingling Fan、Sen Chen 和 Yang Liu。2020。Core：实现代码更改的评审推荐自动化。载于 *2020 IEEE*第*27*届软件分析、演化与再工程国际会议（*SANER*）论文集。IEEE，284–295。[37] Benjamin Strout。2023。漏洞研究人员手册——发现、报告漏洞的综合指南。[38] Gemma团队。2024。Gemma。（2024）。doi：10.34740/KAGGLE/M/3301[39] Gemini团队、Petko Georgiev、Ving Ian Lei、Ryan Burnell、Libin Bai、Anmol Gulati、Garrett Tanzer、Damien Vincent、Zhufeng Pan、Shibo Wang 等。2024。Gemini 1.5：在数百万个上下文令牌中解锁多模态理解。*arXiv*预印本 *arXiv:2403.05530*（2024）。[40] Qwen团队。2024。Qwen2.5：基础模型家族。https://qwenlm.git hub.io/blog/qwen2.5/ [41] Christopher Thompson 和 David Wagner。2017。开源项目中现代代码审查与安全性的大规模研究。载于 第*13*届软件工程预测模型与数据分析国际会议论文集。83–92。[42] Rosalia Tufano、Simone Masiero、Antonio Mastropaolo、Luca Pascarella、Denys Poshyvanyk 和 Gabriele Bavota。2022。使用预训练模型提升代码审查自动化水平。载于 第*44*届软件工程国际会议论文集。2291–2302。[43] 通用漏洞披露。2005。通用漏洞披露。*MITRE*公司，*[*在线*]* 可用地址：*https://cve. mitre. org/index. html*（2005）。[44] Zezhou Yang、Cuiyun Gao、Zhaoqiang Guo、Zhenhao Li、Kui Liu、Xin Xia 和 Yuming Zhou。2024。现代代码审查综述：进展、挑战与机遇。*arXiv*预印本 *arXiv:2405.18216*（2024）。[45] Jiaxin Yu、Liming Fu、Peng Liang、Amjed Tahir 和 Mojtaba Shahin。2023。通过代码审查进行安全缺陷检测：对OpenStack和Qt社区的研究。载于 *2023 ACM/IEEE*软件工程与度量实证国际研讨会（*ESEM*）。IEEE，1–12。[46] Jiaxin Yu、Peng Liang、Yujia Fu、Amjed Tahir、Mojtaba Shahin、Chong Wang 和 Yangxiao Cai。2024。深入了解基于大语言模型的安全代码评审：能力、障碍与影响因素。*arXiv*预印本 *arXiv:2401.16310*（2024）。[47] Yongda Yu、Guoping Rong、Haifeng Shen、He Zhang、Dong Shao、Min Wang、Zhao Wei、Yong Xu 和 Juhong Wang。2024。微调大语言模型以提升自动化代码审查的准确率与可理解性。*ACM*软件工程与方法学汇刊34，1（2024），1–26。