无名稿件编号（将由编辑插入）

用于保障大语言模型生成代码安全的检测–修复–验证：一项多语言实证研究

Cheng Cheng

arXiv:2603.00897v1 [cs.SE] 1 Mar 2026

收稿日期：日期 / 录用日期：日期

摘要 大语言模型正越来越多地被用于生成可运行软件。在实践中，安全问题通常通过检测–修复–验证（DRV）循环来处理，该循环检测问题、应用修复并验证结果。本研究针对项目级制品考察此类工作流，并回应四项不足：L1，缺乏配备可执行函数与安全测试的项目级基准测试；L2，除单独研究检测或修复之外，关于流水线级有效性的证据有限；L3，检测报告作为修复指导的可靠性尚不明确；L4，在验证条件下修复可信度与副作用仍不确定。本文引入一个新的基准数据集[1]，由可运行的Web应用项目组成，并配套功能测试与针对性安全测试，同时支持项目、需求与函数三个层级的提示粒度。评估在可比的预算约束下比较仅生成、单轮 DRV 以及有界迭代式 DRV 变体。结果通过基于测试的验证，以安全且正确的产出加以衡量，并分析中间制品，以评估报告可操作性以及修复后失效模式，例如回归问题、语义漂移和新引入的安全问题。

关键词 大语言模型 ·检测、修复与验证工作流 · 软件安全 · 自动化修复 · 项目级基准测试 · 实证评估

1 引言

如今，大语言模型已成为日常软件开发的一部分。它们能够将自然语言需求转换为可运行的代码，并加快各种任务中的原型开发与实现，涵盖从编写单个函数到构建完整的项目级制品。在这种情况下，应从漏洞管理的角度理解安全：漏洞管理将软件保护视为一个持续过程，而非编码时的一次性步骤。典型的漏洞管理工作流始于开发和维护期间识别安全相关问题，随后检测代码或运行系统中的潜在漏洞，分析并将根本原因定位到具体组件或代码区域，再根据风险与影响确定处理优先级。接下来进入修复阶段，开发人员通过应用补丁或重构不安全逻辑来修复漏洞，然后进行验证，确保修复有效且不会破坏预期行为。最后，工作流还包括修复识别与跟踪，将补丁与漏洞关联起来，

Cheng Cheng 加拿大蒙特利尔康考迪亚大学计算机科学与软件工程系 电子邮箱：
cheng.cheng.20171@mail.concordia.ca

[1] https://github.com/Hahappyppy2024/EmpricalVDR

以便进行审计、发布管理和下游传播，使相关系统和版本能够得到一致更新。当大语言模型参与软件开发时，它们也进入这一生命周期；要保障大语言模型生成的代码安全，就需要通过检测、修复和验证进行同样的端到端管理。

基于这一生命周期视角，当代码由大语言模型生成时，可以将漏洞管理落实为一套结构化的检测—修复—验证（DRV）工作流，以保障大语言模型生成的项目制品的安全。初始生成后，通过自动化检测工具或基于模型的分析识别潜在漏洞。随后，根据报告的问题开展有针对性的修复操作，这些操作也可以借助大语言模型。最后，通过安全检查和功能测试重新评估修复后的制品，以确认漏洞已得到缓解且预期行为得以保留。这一检测—修复—验证工作流反映了在实践中将大语言模型生成的制品纳入开发流水线时，漏洞管理的具体实施方式，也为按照可比的验证协议开展多语言实证评估提供了具体依据。

尽管大语言模型在编程工作流中的应用迅速普及，现有研究结果尚未证实端到端安全加固流水线在大语言模型生成的项目中的效果如何。现有文献大多单独评估漏洞检测或漏洞修复，而非开发者在实践中会执行的生成—检测—修复—验证完整工作流。这一研究空白至关重要，因为检测输出可能不稳定，且容易受到表面代码变体的影响，从而削弱其作为修复指导的可靠性，并导致问题遗漏或误报（Ullah 等人，2024年）。与此同时，面向修复的研究日益强调修复后验证的必要性，例如重新运行安全检查和功能测试，以确认缓解措施有效，并排除回归问题和意外副作用（Wang 等人，2025年；Kim 等人，2025年；Zhang 等人，2024a；Wei 等人，2025年）。综上，这些观察促使我们对整个工作流进行系统评估，重点关注流水线级有效性、检测报告对指导修复的作用，以及修复后输出经验证的质量。

局限 1（缺乏基于测试的项目级基准）。现有评估大多依赖简化的产物（例如独立函数或代码片段），或缺乏可执行且与安全相关的测试预言机的数据集。这限制了对大语言模型生成的项目进行研究的能力，因为在现实的验证条件下，功能测试和安全测试会共同决定一个产物是否既正确又安全。因此，涵盖多种语言生态系统并配备功能测试和安全测试的项目级基准测试仍然较少受到关注。

局限 2（流水线级有效性未经验证）。现有研究通常孤立地评估漏洞检测或漏洞修复。因此，目前仍缺乏证据来证明端到端的生成–检测–修复–验证工作流，能否始终提升大语言模型生成项目相较于仅生成基线的安全且正确产出率。

局限 3（检测可靠性作为修复指导的作用尚不明确）。针对大语言模型生成代码的检测报告可能不稳定，且证据质量参差不齐。将其用作修复指导时，遗漏的研究发现可能导致漏洞未得到处理，而误报则可能促使人们进行不必要或方向错误的编辑。

局限性 4（修复可信度和副作用的不确定性）。基于大语言模型的修复可能会消除报告的问题，但也可能引入回归问题、语义漂移或新的安全缺陷。因此，经过验证的“修复后”项目级输出的可信度，以及实践中占主导地位的失效模式，仍未得到充分刻画。

这些局限性促使我们对基于大语言模型的检测与修复工作流开展系统的实证研究，将其视为一个整体流程，重点关注它们何时能提升安全性和正确性，以及成功或失败的原因。因此，本研究由以下研究问题引导。

因此，本研究围绕以下研究问题展开：
RQ1（流水线级有效性）。在预算约束相当的情况下，带有基于测试的反馈的有界迭代检测–修复–验证工作流，能否提高

项目级制品相较于仅生成和单轮检测–修复基线的安全且正确产出率？这种效果会如何随提示词粒度而变化？
研究问题2（检测可靠性作为修复指导）。将大语言模型生成的检测报告应用于项目级工件时，其可靠性和可用于修复的程度如何？
RQ3（修复可信度与主要失败模式）。在迭代工作流中，基于大语言模型的修复在多大程度上能够缓解报告的漏洞，同时避免引入功能回归？在验证过程中，哪些失效模式最为突出？

为回答这些问题，本研究在可比较的预算约束和基于测试的反馈条件下，评估仅生成、单轮检测–修复以及有界迭代检测–修复–验证等变体。研究引入了一个项目级基准数据集，以支持在项目粒度上进行端到端度量。该数据集由可运行的 Web 应用项目组成，每个项目均配有可执行的功能测试和针对性安全测试；同时支持三种提示粒度（项目级、需求级和函数级），以控制生成和迭代范围。实验设计在不同粒度下比较这些工作流变体，并将每个项目级工件视为分析单位。研究采用安全且正确产出率量化结果：通过功能测试评估正确性，通过安全测试评估安全性，并辅以修复后再检测。除最终结果外，研究还分析中间工件，以评估检测报告的可靠性和可用于修复的程度（RQ2），并刻画验证过程中修复产生的副作用（RQ3），包括功能回归、语义漂移以及新引入的安全问题。结合结果层面的评估与过程层面的追踪，既能评估有效性（RQ1），也能归因失效模式。

本文其余部分安排如下。第 2 节回顾大语言模型辅助的漏洞检测、自动化修复和面向验证的评估方面的相关工作。第 3 节介绍实验设计，包括所研究的工作流与基线、预算和迭代限制、提示粒度、被测系统，以及用于验证的功能与安全测试套件。第 4 节报告 RQ1–RQ3 的实证结果，涵盖流水线级安全且正确产出率、检测可靠性作为修复指导的表现，以及修复可信度与主要失效模式。第 5 节讨论主要研究发现及其对基于大语言模型的漏洞管理工作流的影响。第 6 节讨论效度威胁及缓解措施。第 7 节总结主要研究发现及其对基于大语言模型的漏洞管理工作流的影响。

2 相关工作

大语言模型（LLM）正日益应用于与安全相关的软件工程任务。现有研究涵盖四个密切相关的方向：安全代码生成、漏洞检测与定位、自动化漏洞修复，以及引导模型行为的提示或工作流策略。本研究通过考察基于测试证据、覆盖多种语言实现和提示粒度的检测–修复–验证工作流，将这些方向联系起来。本节其余部分将回顾各领域的相关工作，并阐明它们与本文提出的评估设置之间的关系。

2.1 大语言模型的安全代码生成

越来越多的研究开始探讨大语言模型生成的代码是否默认安全，以及如何改进其安全属性。相关文献逐渐超越功能正确性，转向通过可测试证据联合评估正确性与安全性的评估方案。SecurityEval 提供了与通用弱点枚举相对应的提示词和示例，用于探查容易产生漏洞的代码生成；SALLM 则进一步系统化评估流程和指标，以减少对临时规则或主观判断的依赖（Siddiq 和 Santos，2022年；Siddiq 等人，2024年）。沿着这一以评估为中心的研究方向，Cheng 等人提出对基于大语言模型的代码生成中的安全相关行为进行基准测试（Cheng 和 Yang，2024年），而 CFCEval 则提供了一套结构更完善的安全导向标准，并通过基于证据的检查

评估生成的实现是否符合安全预期（Cheng 和 Yang，2025年）。除基准测试之外，近期研究还将安全性融入模型训练和生成时控制：SafeCoder 通过指令微调提升安全性（He 等人，2024年），而 SCodeGen 则引入受约束解码，实时引导输出生成可信实现（Qu 等人，2025年）。最后，BaxBench 和 CWEval 等较新的基准测试通过结合可执行测试与面向漏洞利用或结果驱动的验证，推动更贴近现实且可复现的评估，并凸显了代码正确却不安全的现象十分普遍（Vero 等人，2025年；Peng 等人，2025年）。与这些基于基准测试的研究相辅相成的是，来自 S&P/CCS 的实证证据表明，人工智能编程助手会显著影响开发者的安全行为及其输出的安全质量，这说明不能默认代码具有安全性（Pearce 等人，2022年；Perry 等人，2023年）。总体而言，这些研究发现促使人们关注安全感知型代码生成以及严格、以测试为依据的评估，也契合了研究检测–修复–验证流程时采用循证工作流的必要性。

2.2 漏洞检测

先前关于漏洞检测的工作可大致分为传统静态分析、动态分析、基于学习的检测以及近期基于大语言模型的方法，它们在对程序语义、运行时行为和数据驱动建模的依赖程度上有所不同。

传统静态分析。 静态漏洞检测在不执行程序的情况下分析程序代码，并依赖数据流和污点传播等显式程序语义。STASE（Zhang 等人，2024年c）通过应用静态污点分析来识别配置相关漏洞，体现了这一路线，能够提供可解释且可扩展的检测，同时仍对分析精度和上下文复杂性较为敏感。

动态分析。 动态方法通过观察运行时行为来识别漏洞。DeFi-Warder（Wang 等人，2023b）通过运行时语义检测去中心化金融应用中的代币级漏洞，体现了这一范式，不过动态技术本质上受执行覆盖率和输入探索的限制。

基于学习的检测。

基于学习的检测。 基于学习的方法利用程序的学习得到的表示，从数据中推断漏洞模式，这些表示通常基于结构化图构建。MVD 利用流敏感图神经网络进行内存漏洞检测（Li 等人，2022年a）。后续研究加入了解释机制，以提升可解释性并支持面向分析人员的应用，Coca 和 CFExplainer 就是其中的代表（Cao 等人，2024年；Zhang 等人，2024年b）。在检测–修复工作流中，还需考量检测输出作为可操作指导的质量，包括研究发现的稳定性、证据的具体程度以及定位的精确性。

漏洞定位 漏洞定位用于识别导致安全漏洞的精确代码位置——通常精确到函数级、语句级或行级——是连接漏洞检测与后续修复的关键桥梁。与粗粒度检测相比，定位对语义理解和粒度提出了更高要求，因为它不仅要判断是否存在漏洞，还必须指出漏洞在程序中的具体位置。早期的基于学习的方法直接对脆弱代码模式建模，以实现细粒度定位：VulDeeLocator 利用神经模型对代码表示进行语句级定位，表明学习到的特征可以超越文件级或函数级检测（Li 等人，2022年b）。沿着这一方向，VulTeller 将定位与漏洞描述生成相结合，使模型能够

突出显示脆弱区域，并以自然语言解释其中的问题（Zhang 等人，2023年）。近期研究则探索利用大语言模型，通过更强的上下文推理能力和配置选择来改进定位：ENVUL 研究基于大语言模型的定位，并探讨任务特定微调如何与提示策略相互作用，从而产生有效的配置（Tian 等人，2025年）。与此同时，VulPCL 联合处理漏洞预测、分类和定位，将定位与互补的分析任务相结合，以提供更具可操作性的代码级输出（Liu 等人，2024年）。总体而言，相关研究正朝着更具上下文感知能力、更全面的定位方法发展，将检测信号与可操作的修复指导相连接。

2.3 漏洞修复

自动化漏洞修复旨在生成或推荐代码变更，在消除安全漏洞的同时保留程序功能。该领域的早期研究涵盖了多种技术，包括基于规则的补丁生成、模式挖掘和基于学习的方法。Hu 等人于 2025年发表的自动化漏洞修复 SoK 对这一领域进行了全面概述，系统地评述了现有方法、工具、数据集和评估实践，并指出了补丁正确性、泛化和评估可靠性等常见挑战。

2.3.1 基于模式和学习的漏洞修复。

基于学习的修复方法因能够从数据中归纳出可泛化的修复模式而受到关注。VulRepair（Fu 等人，2022年）是一种具有代表性的序列到序列方法，它使用基于T5的模型将漏洞修复表述为代码翻译问题，证明了神经模型生成漏洞修复补丁的可行性。与直接生成补丁的方法互为补充，VulMatch（Cao 等人，2025年）专注于从历史修复中提取并匹配修复模式，利用反复出现的修复模板来提升修复效果和可解释性。

2.3.2 基于大语言模型的漏洞修复

对大语言模型修复能力的早期探索。 与此同时，早期研究探索了大语言模型在未经任务专属微调的情况下所具备的修复能力。Pearce等人（2021）研究了利用大语言模型进行零样本漏洞修复，确立了重要的基线，并揭示了在没有明确训练信号的情况下将大语言模型用于漏洞修复的潜力与局限性。总体而言，这些研究展现了自动化漏洞修复从模式驱动和模型驱动方法向更灵活的数据驱动修复范式的演进。

对基于大语言模型的修复的实证认识。 近期研究则明确将大语言模型作为自动化漏洞修复的核心组件加以考察。VRpilot（Kulsum等人，2024年）对基于大语言模型的漏洞修复进行了深入的案例研究，分析推理策略和补丁验证反馈如何影响修复性能与可靠性。这项研究并未提出单一的端到端解决方案，而是为理解大语言模型在漏洞修复场景中的行为提供了实证见解。

提示词驱动的迭代式大语言模型修复。后续研究旨在通过改进提示和适配策略，增强基于大语言模型的修复能力。De-Fitero-Dominguez 等人（2024）提出了一种利用大语言模型进行自动化代码漏洞修复的方法，表明精心设计提示词和任务表述能够显著影响修复结果。LLM4CVE（Fakih，2025）进一步拓展了这一方向，实现迭代式漏洞修复，使大语言模型能够结合漏洞信息和中间反馈，逐步优化修复候选。

通用弱点枚举引导的大语言模型修复精细化控制。除了提示和迭代之外，近期研究还在探索对大语言模型行为进行更精细化的控制，以修复漏洞。精细化线索优化方法将局部线索与敏感的微调策略相结合，引导大语言模型进行更准确、更有针对性的修复，旨在提升不同漏洞类型下的补丁正确性和稳定性（Zhang 等人，2025）。总体而言，这些研究表明，基于大语言模型的漏洞修复框架正从一次性补丁生成转向迭代式、反馈驱动且能感知线索的修复方式。

2.4 提示工程

早期关于提示的研究证实，它可作为微调的替代方案，有效控制模型行为。Liu 等人（2023）对基于提示的学习进行了全面综述，系统梳理了各类提示方法，并依据其设计原则与应用场景加以分类。

一类主要工作聚焦于通过结构化提示引出多步推理。思维链（Chain-of-thought）（Wei 等人，2022年）提示表明，显式提示中间推理步骤能够显著提升推理密集型任务的表现。后续研究表明，大语言模型即便在零样本设置下，也可借助简单的触发短语完成此类推理（Kojima 等人，2022年）。为提高推理的稳健性，自一致性会聚合多条推理路径并选择一致的结果（Wang 等人，2023a），而由简到繁提示则将复杂问题分解为更简单的子问题，从而实现渐进式推理（Zhou 等人，2023）。思维树（Yao 等人，2023）进一步推广了这些思路，将推理构建为对多条中间思维轨迹的结构化搜索。

除引导推理外，近期研究还探索了提示优化与复用。Wan 等人（2024年）研究了指令和示例在自动提示优化中的相对作用，而局部化零阶提示优化则提出了无需梯度访问即可改进提示词的黑盒优化策略（Hu 等人，2024年）。思维缓冲区（Yang 等人，2024年）引入了可复用的推理模板，以增强跨任务的思维增强推理；而结合提示优化的动态奖励则通过根据奖励反馈调整提示词，实现免微调自对齐（Singla 等人，2024年）。作为这些方法的补充，DSPy（Khattab 等人，2024年）将提示视为可编程、可优化的流水线，并将声明式规范编译为可自我改进的提示工作流。

3 方法论

本节将研究方法描述为一个端到端流水线（图 1）。我们构建了一个多语言 EduCollab 基准，其中包含配对的功能测试套件和安全测试套件。随后，使用两个大语言模型，在不同的提示粒度和工作流条件下进行漏洞检测与修复。最后，通过重新运行相应的安全目标及完整的功能测试套件，对每个提出的补丁进行验证。我们按目标记录验证结果，并按模型–语言–工作流设置汇总，以支持 RQ1–RQ3 的分析。

3.1 数据集构建

该数据集包含三个独立实现的 Web 应用程序：EduCollab-PHP、EduCollab-JS 和 EduCollab-Python，它们均遵循 MVC 架构。这三个应用程序都以课程为中心且支持多租户，并以课程 作为租户边界。尽管它们在语言和框架方面各不相同，但都遵循相同的功能规范，从而能够在应用语义一致的条件下进行受控的跨语言

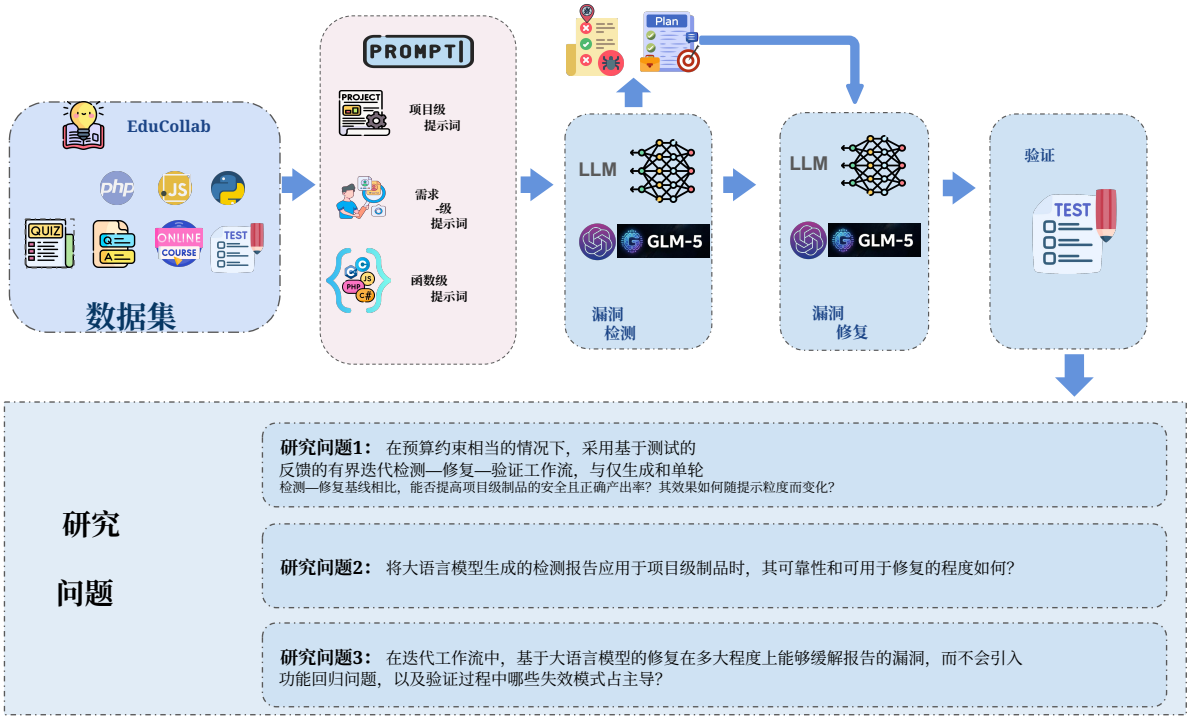


图 1 实验流程概览。以 EduCollab 基准（PHP/JavaScript/Python）为起点，按照三种提示粒度（项目级、需求级和函数级）构建工件，并通过检测–修复–验证工作流进行处理。验证通过测试套件执行，其结果提供有依据的反馈，以推动有限迭代并支持对 RQ1–RQ3 的测量。

比较。数据集及配套资料可在公共代码仓库中获取。[2]

数据集构建分为五个阶段。首先，制定统一的功能规范，以标准化不同实现中的应用范围和预期用户工作流。其次，按照该蓝图实现三个特定语言的项目，并确保角色边界和租户隔离语义保持一致。第三，将支持的用例映射到 OWASP Top 10:2025（R01--R10）风险分类体系，以描述每个功能界面预期的安全暴露面。第四，为每个项目开发两套互补的测试套件。功能测试使用良性输入验证预期工作流，安全测试则探测与十大相关的攻击面，并提供可观察的失败信号以供验证。第五，为大语言模型交互准备三种提示粒度：项目级、需求级和函数级。这样便能在可比预算下评估检测、修复与验证工作流，同时改变提供给模型的上下文和局部性。总体而言，这种分阶段设计使不同语言中的应用语义保持一致，并支持以测试为依据、可复现地评估不同提示粒度下的安全且正确产出率。

3.1.1 阶段 *1*：功能规范

该数据集围绕一套统一的功能规范构建，以支持跨语言的受控比较。我们依据同一蓝图开发了三个 EduCollab 实现，分别使用 PHP、JavaScript 和 Python 编写。因此，检测、修复和验证过程中观察到的差异反映的是工作流行为以及语言或运行时效应，而非应用范围的差异。该规范涵盖了以课程为中心的协作平台中的典型工作流，包括

[2] https://github.com/Hahappyppy2024/EmpricalVDR

表 1 EduCollab 中实例化的功能用例（PHP/JavaScript/Python 通用）。

ID	用例
U1	课程与成员管理（基于角色的访问控制）。
U2	角色更新（成员角色变更）。
U3	安全错误配置检查（Cookie 标志、 `nosniff`、调试信息泄露）。
U4	邀请链接加入（基于令牌的注册）。
U5	帖子/评论与搜索（关键词、排序）。
U6	存储型/反射型渲染面（与跨站脚本攻击相关的输出）。
U7	ZIP/CSV导入与下载（CSV 公式攻击面）。
U8	通过角色变更实现自我提权（助理将自己的角色提升为管理员）。
U9	身份认证与会话生命周期（登录/注销）。
U10	带上传的作业提交（文件处理）。
U11	带路径处理约束的上传/下载（完整性相关攻击面）。
U12	ZIP/CSV导入与下载（完整性检查）。
U13	管理员审计日志视图（`/admin/audit`）；缺少对关键事件的日志记录。
U14	异常条件处理（格式错误的输入；错误泄露面）。

身份认证与会话管理、带有基于角色的访问控制的课程与成员管理、基于令牌的邀请加入、内容发布与搜索、文件提交与资源分发，以及审计、数据导入与导出等管理功能。

表1列出了在各项目中实例化的具体用例。若干用例在设计上即与安全相关，包括渲染与搜索面、ZIP与CSV导入及下载、路径处理、错误配置检查、审计日志记录以及异常条件处理。这些功能提供了Web应用安全评估中常被作为目标的现实攻击面。三种实现均暴露同一组用例，并使用共享的功能测试套件加以验证，以在安全测试以及后续由大语言模型驱动的检测、修复与验证实验之前确立语义等价性。

3.1.2 阶段 *2*：跨语言实现

基于阶段 1 中统一的功能规范，我们使用 PHP、基于 Node.js 和 Express 的 JavaScript，以及基于 Flask 的 Python 实现了三个 EduCollab 变体。每个变体都采用 MVC 风格的组织结构，将路由与控制器、数据访问和服务器渲染视图分离，并使用 SQLite 进行持久化。保持不同语言版本的范围和数据模型一致，能够将检测、修复和验证方面的观察差异归因于语言和框架因素，而非功能缺失或不一致。

通过共享租户机制和授权语义，进一步增强了可比性。课程构成数据隔离的租户边界；对于管理员、教师、助理、学生和未认证角色，针对相同资源类型（包括课程、成员资格、帖子、上传、作业和提交）采用相同的基于角色的访问控制规则。在适用情况下，每个用例均通过 Web用户界面工作流和对应的 /api/* 端点提供。这样，同一套功能检查和安全检查便可在交互式和程序化环境中运行。

3.1.3 阶段 *3*：*OWASP Top 10:2025* 风险映射

本阶段通过将已实现的用例映射到 OWASP Top 10:2025（2025年）风险分类体系（R01--R10），描述数据集的安全暴露面。其目的并非全面覆盖 Web

表2 2025年十大Web应用安全风险。

Rank	类别（标识符）	含义
1	失效的访问控制（R01）	授权检查缺失或可被绕过， 因此用户可以访问超出其权限范围的数据或执行相关操作。
2	安全配置错误（R02）	应用程序、服务器、框架或云设置 存在安全隐患，或仍采用不安全的默认设置。
3	软件供应链故障（R03）	风险来自依赖项、构建产物， 或交付流水线，而非您自己的代码。
4	加密失效（R04）	由于缺少、薄弱或误用密码学，数据未得到妥善保护。 缺失、薄弱或使用不当。
5	注入（R05）	不可信输入被解释为代码/命令/查询 由解释器或引擎执行。
6	不安全设计（R06）	系统的设计缺乏必要的安全控制措施 或从一开始就缺乏抗滥用能力。
7	身份验证失效（R07）	身份和会话机制薄弱，从而允许 账户接管或会话滥用。
8	软件或数据完整性故障 (R08)	应用程序信任可能遭到篡改的代码/数据， 导致不安全的更新或流程遭到破坏。
9	安全日志记录 \警报故障 (R09)	未能妥善监控与安全相关的事件， 延迟检测和响应。
10	异常情况处理不当（R10）	错误/异常路径不安全，可能泄露 信息、绕过检查，或以开放失败方式结束。

安全问题，而是使 EduCollab 与广泛使用的风险类别保持一致，从而确保不同语言之间的评估和报告具有可比性。以课程为中心的协作平台提供了合适的场景，因为其中包含常见的安全相关攻击面，例如身份认证与会话管理、基于角色的授权、管理操作、用户生成内容渲染，以及数据和文件导入导出。

表2总结了本研究使用的十大风险类别。每个条目列出类别名称、其标识符（R01--R10），以及为测试设计和结果报告提供依据的操作性解释。为便于阅读，保留了原始OWASP排名，而分类标识符则作为本文其余部分的主要参考。

表3将数据集的功能用例与十大类别相关联。用例按风险标识符分组，并按排名升序排列，以与表2保持一致。多个用例可以映射到同一类别，因为同一风险族可能出现在应用程序的不同部分，例如成员管理和角色更新中的授权检查。该映射涵盖九个类别（R01、R02、R04--R10）。R03涉及软件供应链故障，主要由依赖项选择以及构建或部署流程驱动，因此不在本数据集所包含的应用程序内用例范围内。

3.1.4 阶段 *4*：功能与安全测试套件

此阶段通过为每个实现配备两个互补的测试套件，将数据集转化为可执行基准测试。函数测试套件用于检查功能正确性，而安全测试套件则用于检查高风险面是否符合安全预期。二者共同为评估过程中的正确性与安全性提供基于测试的证据。

功能正确性测试。功能测试会执行以课程为中心的协作系统预期的端到端工作流，包括身份认证与会话处理、课程创建与成员管理、基于邀请的加入、发帖与搜索、带文件上传的作业提交，以及

表3 按排名升序排列、与十大风险列表（R01–R10）相对应的功能特性（用例）。该表涵盖10个风险类别中的9个（R01、R02、R04–R10）；R03（软件供应链故障）被视为流程级风险，因此不作为应用程序内用例呈现。

排名 ID	功能特性（用例）
R01	课程 \成员资格（基于角色的访问控制）
R01	角色更新（成员角色变更）
R02	安全配置错误（Cookie 标志 / nosniff / 调试信息泄露）
R04	通过邀请链接加入（基于令牌的加入）
R05	帖子/评论 \ 搜索（关键词/排序）
R05	存储型/反射型渲染（跨站脚本攻击攻击面）
R05	ZIP/CSV 导入 \下载（CSV 公式）
R06	通过角色变更自我提权（助理可以将自己的角色提升为管理员）
R07	身份验证 \ 会话（登录/注销）
R08	作业提交上传（文件处理）
R08	上传/下载 \ 路径处理（完整性攻击面）
R08	ZIP/CSV 导入 \ 下载（完整性）
R09	管理员审计日志视图（/admin/audit），关键事件缺少日志记录
R10	异常情况（格式错误的输入 / 错误信息泄露）

数据导入与导出。测试在适当的授权上下文中执行，确保授权角色可以成功执行预期操作，而未授权角色会被拒绝执行，如共享的功能蓝图中所规定。

安全测试。 安全测试探测与十大相关的攻击面，旨在产生明确的通过或失败信号。测试套件针对具有代表性的漏洞类别，包括失效的访问控制（如跨租户访问和不安全的直接对象引用（IDOR）行为）、注入类输入处理问题、不安全的文件和压缩包处理、导入与导出流程中的完整性故障，以及异常情况下的信息泄露。与功能测试不同，安全测试具有对抗性。它们使用精心构造的输入和违反边界的请求，然后检查观察到的行为是否符合安全预期。

检测、修复与验证的基线设置。 每个实现都在受控的基线状态下初始化，在此状态下，完整的功能测试套件均通过，从而在进行任何安全加固之前建立语义正确的参照。随后，在同一基线上运行安全测试套件，以识别该语言的一组固定的初始失败的安全测试目标（N）。这些失败目标提供了可复现、可触发的证据，作为检测–修复–验证评估的起点。在实验过程中，检测会为每个目标生成结构化的漏洞报告，修复会依据该报告应用补丁，而验证则会重新运行相应的安全测试目标及完整的功能测试套件，以确认修复效果并检测回归问题。

减少安全测试中的错误通过和错误失败。采用了多项设计选择来提高可复现性。在可能的情况下，每项测试只隔离一个安全属性。判定依据两类证据：HTTP 层信号，例如状态码、重定向、标头和响应正文；以及状态层信号，例如数据库状态、文件系统状态或对象可访问性。对于每个安全类别，测试套件都包含正向控制（已授权操作应成功）和负向控制（未授权操作应失败）。测试运行通过使用包含固定种子数据的新数据库和临时工作目录彼此隔离。泄露检查通过使用稳定的模式集来避免脆弱的字符串匹配

表4 数据集中使用的测试套件结构与判定信号。

套件	覆盖重点	典型判定证据	在D–R–V基线中的角色
功能性	良性工作流下 正确授权 （例如，课程/会员、 邀请加入、发帖/搜索、 提交、导入/导出）	HTTP成功的 允许的操作 + 预期的 UI/JSON 输出 + 预期的数据库状态	所有测试均通过 （语义等价）
安全	对十大的对抗性探测 相关攻击面（例如，访问 控制/不安全的直接对象引用，注入式 输入，文件/归档处理， 导入/导出完整性， 错误信息泄露）	HTTP 拒绝（401/403） 针对边界违规 + 不存在不安全副作 用（数据库/文件） + 泄漏模式检查	所有测试均失败 （可触发漏洞）

表5 按语言划分的数据集测试套件构成。功能测试用于验证基线正确性，而安全测试（目标）则用于收集最初未通过的安全要求，以供检测–修复–验证评估使用。

Lang	功能测试（#）	安全测试 / 目标（#）	总计（#）
Python	15	7	22
JS	8	13	21
PHP	14	6	20
总计	37	26	63

表 6 R01–R10 分类体系下的安全测试覆盖率（OWASP Top 10:2025 映射）。

类别	# 安全测试
R01（访问控制 / 不安全的直接对象引用）	6
R02（安全配置错误 / 会话）	2
R04（加密失效）	4
R05（注入：SQL 注入 / 跨站脚本攻击 / CSV）	6
R08（完整性 / 上传 / 标头）	4
R09（日志记录与监控）	1
R10（异常 / 错误处理）	3
总计	26

。对于基于载荷的测试，首先确认可达性，以确保相关代码路径在应用对抗性输入之前已被执行。

表4和表5总结了本研究所使用的可执行基准测试的结构与规模。表4从覆盖重点、判定证据以及其在检测–修复–验证（D–R–V）基线中的作用等方面，对两个互补的测试套件进行了描述。功能测试套件在正确授权的前提下验证预期工作流，并要求获得成功响应以及一致的系统状态；安全测试套件则针对高风险攻击面发起对抗性探测，并要求获得拒绝响应且不存在不安全副作用。表5进一步量化了各语言中的测试套件构成，报告每种语言的功能测试数量和初始失败的安全测试目标数量，以及总体总计。表6则通过按照R01–R10风险类别（OWASP Top 10:2025映射）细分26个安全目标，对这些总结进行了补充，明确展示了其中涉及的漏洞类别以及覆盖范围在各类别之间的分布。总体而言，这些表格同时阐明了基准测试的定性评估信号，以及其定量规模与覆盖范围。

3.1.5 阶段 *5*：提示粒度设计

本阶段定义了大语言模型交互所使用的三种提示粒度，应用于检测、修复与验证工作流。底层应用程序在不同设置下保持不变。变化的仅是提供给模型的上下文范围与局部性。这样便可研究粒度如何影响所检测到的问题、所提出修复方案的可执行性，以及相关产物在验证过程中的表现。

项目级。项目级提示从整体上描述系统，包括租户边界（课程）、用户角色、主要模块以及技术栈。该设置支持跨组件的广泛推理，适合识别由功能交互产生的风险。

需求级。 需求级提示呈现按角色限定的需求与边界约束，例如给定角色允许执行哪些操作，以及必须保护哪些资源。该设置强调边界推理，并将潜在风险与特定工作流和入口面联系起来。

函数级。 函数级提示聚焦于单个用例或端点。它们规定输入与输出、诸如授权等前置条件，以及相关资源和数据对象。其目的是鼓励局部修改、提高修复的可操作性，并减少语义漂移。效果通过修复后的安全重新测试和函数测试回归检查进行评估。

漏洞检测与漏洞修复的完整提示规范涵盖全部三种提示粒度，已在随附代码仓库[3]中公开提供。

3.2 D–R–V 工作流与基于验证的度量

本节将图 1 所示的工作流应用于多语言 EduCollab 基准的评估。首先，定义在预算上限相同的条件下进行比较的工作流条件（W0–W2），其中包括迭代上限为 $K = 2$ 的有界迭代设置。随后，说明用于引出检测和修复行为的三种提示粒度。接着，介绍流水线的三个阶段——检测、修复和验证；验证以可执行的功能与安全测试套件为依据。最后，说明如何将每个目标的测试结果和记录的信号汇总为用于回答 RQ1–RQ3 的度量。结果见第 4 节。

3.2.1 工作流条件与预算上限

在相同的交互预算下比较三种工作流条件。W0（基线）表示未修复项目状态。在此状态下，功能测试套件必须全部通过，以确立正确的参考实现，供后续修复使用；同时，安全测试套件被设计为无法通过，从而得到一组固定的失败安全测试目标，用于评估。

W1（单轮 *D–R–V*）对每个目标执行一轮检测–修复–验证。对于每个未通过的目标，仅发出一次检测提示，以获取修复计划及其中引用的文件/API；随后发出一次修复提示，生成最小补丁。然后，通过重新运行目标安全测试和完整功能测试套件来验证该补丁。

W2（有界迭代式 *D–R–V*）在 W1 的基础上加入基于测试的反馈，迭代次数上限固定为 $K = 2$。每次迭代包含：(i) 根据当前故障证据进行检测和修复；(ii) 使用相同的目标安全测试和完整的功能测试套件进行验证。当目标安全测试通过且未引入功能回归时，流程会提前停止。如果首次修复尝试未成功，则允许进行第二次迭代。在该次迭代中，修复提示会补充模型输出所请求或暗示的额外代码上下文（例如，所引用的

[3] https://github.com/Hahappyppy2024/EmpricalVDR

文件、路由或 API），并以文件级代码片段的形式提供，以支持修订后的修复尝试。工作流会在获得经验证的修复方案或迭代预算耗尽时终止，从而确保 W1 和 W2 在相同的上限下具有可比性。

3.2.2 提示粒度

检测与修复通过三种提示粒度来引导，这些粒度的区别在于向模型提供的系统上下文数量。其目标是将宽泛的架构级推理、角色范围分析以及目标驱动、测试验证的修复区分开来。

项目级提示 提供对 EduCollab 的高层描述，包括其多租户环境、角色、主要模块和入口点。在此范围内，要求模型列举可能的漏洞风险，并使用 R01--R10 分类体系及相应的定位标签（层和攻击面）对其进行总结。在研究问题2中使用这些提示，以刻画模型根据全局系统描述进行推理时检测报告的广度和结构。

需求级提示将分析范围缩小至给定角色视角下的一项特定安全需求（管理员、教师、助理、学生或未认证用户）。每个需求级提示都会要求列出与该角色和需求相关的漏洞可能性，并同样将范围限定在 R01--R10 分类体系及相同的定位标签内。对于每种模型–语言设置，都会收集五份角色范围响应，并在研究问题2中加以汇总，以同时呈现典型报告内容（跨角色均值）以及角色条件化所带来的差异。

函数级提示以目标为导向，并以测试为依据。对于失败的安全测试目标，提示词会提供失败测试和修复所需的最小代码上下文，并要求模型仅针对所引用的文件生成最小补丁。在 W1/W2 中使用函数级提示来执行 D–R–V 流水线，并通过可执行测试产生可验证的结果，以支持 RQ1 和 RQ3。

3.2.3 大语言模型选择

选择 ChatGPT-5 和 GLM-5 作为两种擅长遵循指令的代码相关任务模型，从而在相同基准测试和以测试为依据的 D–R–V 验证协议下进行跨模型比较（图 ??）。ChatGPT-5 是广泛应用于开发者辅助和代码生成的通用基线?。GLM-5 则作为 GLM 系列的代表模型纳入研究，提供了一个不同的提供方／模型系列，以考察观察到的检测和修复行为是否会超出单一生态系统而持续存在??。在相同的工作流条件下评估这两种模型，有助于区分 D–R–V 设计和验证反馈的影响与模型特有的行为差异。

3.2.4 检测阶段

在给定的提示粒度下，检测阶段会针对每种模型–语言设置生成结构化检测报告。为确保不同模型和语言之间的可比性，报告限定使用 R01–R10 分类体系（映射至 OWASP Top 10:2025）及一组通用定位标签。对于报告中的每项风险，输出会记录：(i) R01–R10 风险类别；(ii) 疑似的层 （代码层线索）；以及 (iii) 疑似的攻击面 （入口面线索）。报告还包含简要的支持性理由，以及精简且可执行的修复计划。

在项目级和需求级提示词下，模型需要按照相同的 R01--R10/层/表面格式，在给定范围内列举可能存在的漏洞。需求级检测从五种角色视角（管理员、教师、助理、学生和未认证用户）进行，以捕捉角色条件化如何影响风险覆盖范围和定位。结果通过报告按角色划分的输出，以及报告每种设置下五种角色的平均值来汇总。

在函数级提示词（用于 w0/W1/W2）下，检测由一个失败的安全测试目标驱动。提示词提供失败测试和可用的代码片段，并要求模型解释失败原因、找出最相关的文件和 API/路由，以及为后续修复阶段提出具体的修复计划。这些函数级报告可作为即时修复指导，之后还会根据修复可操作性和基于测试的可靠性进行评估。

3.2.5 修复阶段

修复阶段会将前述检测输出转化为针对每个失败安全测试目标的具体补丁。修复提示词包含失败测试以及从检测报告中选取的元素，包括报告中的 R01--R10 类别、层/表面标签，以及任何被提及的文件、路由或 API。明确保留这些信息，可确保报告内容与实际修改之间的可追溯性，从而支持后续对修复充分性和位置正确性的检查。

修复遵循最小改动约束。模型只能编辑提示词中提供的文件名和代码片段，不得引入新的依赖项，并且需要以满足目标安全属性为前提，尽可能减少改动。如果解决某个目标需要编辑多个文件，则改动仅限于必要的最小文件集合。

修复尝试次数遵循工作流条件。W1 根据初始检测报告生成单个补丁。若第一个补丁未通过验证，W2 允许额外尝试一次。在第二次尝试中，如果第一轮缺少文件级上下文，且模型输出要求或暗示需要此类上下文（例如特定的路由处理程序、中间件或控制器），则会补充提供。第二次尝试遵循相同的补丁约束，并以第一次尝试的验证结果为依据。

4 结果

4.1 RQ1（流水线级有效性）。

4.1.1 指标。

RQ1 比较在交互预算匹配的情况下，单轮检测–修复–验证（W1）与有界迭代检测–修复–验证（W2，上限为 $K = 2$）的基于测试的结果，并以未经修复的项目状态作为基线（W0）。对于每个模型–语言设置，评估均使用固定的功能测试套件和一组固定的初始失败安全测试目标。表 8 报告了各工作流下的测试套件规模（#功能测试和#安全测试/目标）以及通过的测试数量（功能测试通过数和安全测试通过数）。

安全且正确产出率（SCY）按目标 级别衡量，即在给定工作流下，初始失败的安全测试目标中转为通过且未 引入任何功能回归的比例。在表 8 中，#SCY 报告了 W0/W1/W2 下安全且正确的目标数。为了量化在预算可比的情况下有限迭代的效果，表 7 报告了 W1 和 W2 的 SCY 比率，以及在相同上限 $K = 2$ 下的改进 ΔSCY $=$ SCY(W2)$-$SCY(W1)。

4.1.2 研究发现。

在匹配预算（$K = 2$）下，与未修复基线（W0）和单轮检测–修复基线（W1）相比，有界迭代检测–修复–验证（W2）始终能提高流水线级安全且正确产出率。表 8 显示，按照设计，在 W0 下，所有模型和语言的安全测试目标均未通过（安全测试通过数$=0$ 和 #安全且正确产出率$=0$），这为比较确立了共同的起点。从 W0 转向 W1 在若干设置下带来了显著提升，但也暴露出单轮流程无法取得任何安全且正确结果的情况（例如，ChatGPT-5–Python 在 W1 下的 #安全且正确产出率仍为 $=0$ ）。

表 7 RQ1 摘要：在匹配预算下的安全且正确产出率（SCY），比较单轮检测–修复（W1）和有界的迭代检测–修复–验证（W2）。

LLM	Lang	粒度	预算上限	安全且正确产出率 (W1)	安全且正确产出率 (W2)	Δ安全且正确产出率 (W2–W1)
ChatGPT-5	Python	函数	$K=2$	0.00	0.57	+0.57
	JS	函数	$K=2$	0.54	0.77	+0.23
	PHP	函数	$K=2$	0.17	0.50	+0.33
GLM-5	Python	函数	$K=2$	0.43	0.86	+0.43
	JS	函数	$K=2$	0.23	0.69	+0.46
	PHP	函数	$K=2$	0.33	0.83	+0.50

相比之下，W2 在每种模型–语言设置下都能产出更多安全且正确的目标，这表明迭代循环中的基于测试的反馈能够实现单轮无法达到的进展。

总体效果汇总于表 7。对于 ChatGPT-5，Python/JavaScript/PHP 的安全且正确产出率（SCY）从 W1 中的 0.00/0.54/0.17 提升至 W2 中的 0.57/0.77/0.50，分别提高了 $+0.57$、$+0.23$ 和 $+0.33$。对于 GLM-5，SCY 从 W1 中的 0.43/0.23/0.33 提升至 W2 中的 0.86/0.69/0.83，分别提高了 $+0.43$、$+0.46$ 和 $+0.50$。这些提升并非由评估范围变化所致：每种语言的功能测试和安全测试套件规模均保持不变（表 8），且所有模型的 W2 均采用相同的迭代预算上限（$K=2$）。

表 8 进一步说明了 SCY 提升的原因。主要贡献来自 W2 下安全问题解决率的提高（安全测试通过数更多），而功能回归仍然有限。例如，ChatGPT-5–Python 从 W1 到 W2 时，通过的安全目标数从 0 增至 4，使 #SCY$=4$ 提升，且未观察到功能损失（全程功能测试通过数$=15$）。对于 GLM-5–Python，安全测试通过数 从 3 增至 6，使 #SCY$=6$提升。在 JavaScript 中，两种模型在 W2 下的安全表现均大幅提升（ChatGPT-5：$7\rightarrow11$；GLM-5：$3\rightarrow10$），但两者在 W2 下也都出现了一项功能回归（功能测试通过数从 8 降至 7），因此 #SCY$=10$，分别为 9。PHP 呈现出类似模式：ChatGPT-5 在 W2 下的安全测试通过数从 1 增至 4，但出现一项功能回归（$14\rightarrow13$），使 #SCY$=3$；GLM-5 则达到 #SCY$=5$，且没有功能损失。总体而言，迭代工作流带来了 SCY 的净提升，因为新增的安全修复弥补了少量功能回归造成的影响。

W2 带来的收益幅度因语言和模型而异，这表明在 W1 难以通过单次尝试定位或修复漏洞的情况下，迭代最有价值。最大收益出现在 ChatGPT-5–Python（$+0.57$）和 GLM-5–PHP（$+0.50$）中，而最小收益出现在 ChatGPT-5–JavaScript（$+0.23$）中，此时 W1 已实现相对较高的 SCY（0.54）。这一规律符合“边际收益递减”效应：当 W1 的表现已经较好时，在相同预算上限下，迭代带来的增量改进较小。

关于提示粒度，当前表格报告的是函数级条件。可以在其他提示范围下重复相同的预算匹配比较（W2 与 W1，$K=2$），并按照表 7 的相同格式报告 SCY 和 ΔSCY，以评估粒度的影响。这种设计能够将提示范围的作用与迭代预算区分开来，从而直接检验更宽或更窄的提示范围是否会增强或削弱以测试为依据的迭代所带来的收益。

RQ1 要点。在相同的预算上限（$K=2$）下，有界迭代式 D–R–V（W2）相较单轮 D–R（W1）持续提升安全且正确产出率，在各模型–语言设置中的增益范围为 $+0.23$ 至 $+0.57$（表 7）。

表 8 RQ1 函数级结果：不同工作流中基于测试的结果。W0 表示初始（未经修复的）项目状态。W1 执行一轮检测–修复–验证，W2 执行有界迭代检测–修复–验证。各行按语言分组，以便在同一语言内进行模型比较。

Lang	LLM	工作流	#函数测试	#Sec测试	Func通过	Sec通过	#SCY
Python	ChatGPT-5	W0 基线	15	7	15	0	0
		W1 单次处理	15	7	15	0	0
		W2 迭代	15	7	15	4	4
	GLM-5	W0 基线	15	7	15	0	0
		W1 单次处理	15	7	15	3	3
		W2 迭代	15	7	15	6	6
JS	ChatGPT-5	W0 基线	8	13	8	0	0
		W1 单次处理	8	13	8	7	7
		W2 迭代	8	13	7	11	10
	GLM-5	W0 基线	8	13	8	0	0
		W1 单次处理	8	13	8	3	3
		W2 迭代	8	13	7	10	9
PHP	ChatGPT-5	W0 基线	14	6	14	0	0
		W1 单次处理	14	6	14	1	1
		W2 迭代	14	6	13	4	3
	GLM-5	W0 基线	14	6	14	0	0
		W1 单次处理	14	6	14	2	2
		W2 迭代	14	6	14	5	5

RQ1：在相同的预算上限（$K = 2$）下，有界迭代式 D–R–V（W2）相较单轮 D–R（W1）持续提升安全且正确产出率，在各模型–语言设置中的增益范围为 $+0.23$ 至 $+0.57$）。

4.2 研究问题2（检测可靠性作为修复指导）。

4.2.1 指标。

研究问题2从两个角度评估检测报告：修复可操作性，即报告是否提供清晰且结构化的修复指导；以及基于测试的可靠性，即报告中指出的风险是否有基准测试中可观察到的证据支持。

可用于修复（报告实用性）。检测报告限定为 R01--R10 分类体系（映射至 OWASP Top 10:2025）和一组固定的定位标签。实用性通过表 13–16 中报告的三个指标进行概括：漏洞类型 统计提及的不同 R01--R10 类别数量；漏洞位置层 统计不同代码层级线索的数量；漏洞位置表面 统计不同入口表面线索的数量。项目级结果按模型–语言设置报告（表 13）。需求级结果以五个按角色限定的提示（管理员、教师、助理、学生和未认证用户）的平均值报告（表 14），并通过按角色细分结果展示设置内的差异（表 15–16）。

在函数级，报告结构还通过 $PlanRate$、平均文件数、平均 *API* 数 和 缺失率 进行概括（表 17）。这些指标反映报告是否提供了可执行的计划，以及是否列明修复所需的文件和 API。

可靠（基于测试的正确性）。可靠性是在相同的模型–语言设置下，依据可执行的安全测试进行评估的。当在相应设置中观察到至少一项与该类别一致的失败安全测试时，所报告的 R01–R10 风险类别即被视为基于测试的。下游验证为实际可靠性提供了额外信号：当报告能够支持一个补丁，使失败的安全测试通过，且在相同工作流约束下不引起功能回归时，这些报告被视为更可靠（表 17 与 RQ3 结果）。

4.2.2 研究发现。

研究问题2从两个角度加以考察：检测报告是否可用于修复 （即提供清晰、可定位且可遵循的指导），以及在基于测试的证据下它们是否可靠 。本节中的表格从提示范围（项目与需求）、角色条件化（五个角色提示词）以及检测–修复流程中的函数级行为等方面提供了互补视角。

可用于修复（报告的实用性）。表 9–10 聚焦于报告中呈现的、可用于修复的内容。在项目级（表 9）下，两种模型针对 R01--R10 均报告了广泛的风险覆盖范围：ChatGPT-5 在不同语言中识别出 9–10 个类别，而 GLM-5 识别出 8–9 个。此范围内也提供了定位线索（5–7 个层和 5–6 个攻击面），但具体程度因模型–语言设置而异。例如，在 JavaScript 中，GLM-5 的项目级报告涵盖的风险类别和层级线索较少（8 个类别、5 个层），而 ChatGPT-5 的报告则涵盖 10 个类别和 6 个层；相比之下，Python 和 PHP 中的差距较小。这表明，项目级提示词通常会引出广泛的枚举，而定位提示的充分程度则更依赖具体设置。

需求级提示词（表 10）更清楚地体现了实用性差异。对五个按角色限定的提示词取平均后，ChatGPT-5 在三种语言中始终能提供丰富的信息（8.8–9.2 个类别；6.0–6.8 个层；5.0–5.4 个攻击面）。相比之下，GLM-5 报告的类别数量和定位线索均较少（5.2–5.8 个类别；4.6–5.2 个层；3.2–3.4 个攻击面）。总体而言，与项目级提示相比，按角色限定的提示能展现出更大的模型差距，尤其是在直接用于指导后续修复的定位线索方面。

放大视图说明了有用性如何因角色而异。在 Python（表11）中，ChatGPT-5 在各角色间相对稳定（通常为 8–9 个家族、6 层和 4–6 个攻击面），而 GLM-5 变化更大，并为未认证角色提供最弱的攻击面线索（2 个攻击面）。这种角色敏感性对修复指导很重要，因为面向边界的角色往往正是必须精确识别入口攻击面和执行点的地方。

可靠（基于测试的正确性）。 基于测试的可靠性通过将报告指导与可执行基准测试中的可观测故障相关联来评估。在此设置中，当报告的指导与同一模型–语言设置下失败的安全测试目标一致并支持经验证的修复时，该报告被视为更可靠。表 13 通过修复成功率提供基于结果的信号。若干设置说明了结构与可靠性之间的差距：尽管 PlanRate$=1.00$，单轮成功仍可能很低（例如，ChatGPT-5–Python 在 W1 中为 0.00），表明报告可遵循，但在定位或根因归因上可能不正确。

有限迭代（W2）在所有设置下都缩小了这一差距。对于两个模型和所有语言，修复成功率均从 W1 提升至 W2（例如，ChatGPT-5–Python：$0.00 \rightarrow 0.57$；GLM-5–Python：$0.43 \rightarrow 0.86$；GLM-5–JavaScript：$0.23 \rightarrow 0.77$；ChatGPT-5–PHP：$0.17 \rightarrow 0.67$）。总体而言，表9–11展示了报告包含哪些内容，以及这些内容如何随提示范围和角色而变化；表 13 则展示了在基于测试的验证下，这些报告能否作为可靠的修复指导。

表 9 在 `R01--R10` 分类法下的项目级检测摘要（OWASP Top 10:2025），按模型–语言设置报告。

LLM	Lang	粒度	漏洞类型（唯一 R01–R10）	Vul. 位置图层	Vul. 位置表面
ChatGPT-5	Python	项目	10	7	6
GLM-5	Python	项目	9	6	6
ChatGPT-5	JS	项目	10	6	6
GLM-5	JS	项目	8	5	6
ChatGPT-5	PHP	项目	9	7	6
GLM-5	PHP	项目	9	6	5

表10 在 `R01--R10` 分类法下的需求级检测摘要（OWASP Top 10:2025）。报告值为五个特定角色提示（管理员、教师、助理、学生和未认证用户）的平均值。

LLM	Lang	粒度	漏洞类型（唯一的 R01–R10）	Vul. 位置图层	Vul. 位置表面
ChatGPT-5	Python	需求	8.8	6.0	5.4
GLM-5	Python	需求	5.2	4.6	3.2
ChatGPT-5	JS	需求	9.0	6.4	5.0
GLM-5	JS	需求	5.8	4.8	3.4
ChatGPT-5	PHP	需求	9.2	6.8	5.2
GLM-5	PHP	需求	5.8	5.2	3.4

表11 Python 的需求级检测按角色细分。每一行对应一个特定角色提示响应。

LLM	Lang	粒度	Role	漏洞类型（唯一的 R01–R10）	Vul. 位置图层	Vul. 位置表面
ChatGPT-5	Python	需求	管理员	9	6	5
			教师	9	6	6
			助理	9	6	6
			学生	9	6	6
			未授权	8	6	4
GLM-5	Python	需求	管理员	5	6	3
			教师	5	4	4
			助理	6	4	4
			学生	5	5	3
			未授权	5	4	2

研究问题2： 检测报告通常可用于修复（PlanRate 较高且 MissingRate 接近于零），但其可靠性有所差异，这体现在结构化指导与基于测试的修复成功之间的差距上。

表 12 检测至修复结果与失效模式分布。对于每种模型–语言–工作流设置，Plan- Rate / 平均文件数 / 平均 API 数 / 缺失率概述了针对失败的安全测试目标的检测报告结构（N）。补丁率和修复成功率在修复–验证步骤之后测量。失效模式列根据 *within unsuccessful targets* (N_{unsucc}) 计算。预算超限仅适用于有界迭代工作流（W2）；W0 条目不适用。

Lang	LLM	工作流	# 安全 Test	Plan Rate	平均# 文件	平均值# APIs	缺失 Rate	补丁 Rate	修复- 成功 Rate
Python	ChatGPT-5	W0 基线	7	–	–	–	–	–	–
		W1 单次遍历 DRV	7	1.00	0.29	1.14	*0.00*	1.00	*0.00*
		W2 迭代 DRV	7	1.00	0.29	1.14	*0.00*	1.00	*0.57*
	GLM-5	W0 基线	7	–	–	–	–	–	–
		W1 单次遍历 DRV	7	1.00	1.43	1.14	*0.00*	1.00	*0.43*
		W2 迭代 DRV	7	1.00	1.43	1.14	*0.00*	1.00	*0.86*
JS	ChatGPT-5	W0 基线	13	–	–	–	–	–	–
		W1 单次遍历 DRV	13	1.00	1.38	1.62	0.00	1.00	*0.54*
		W2 迭代 DRV	13	1.00	1.38	1.62	0.00	1.00	*0.85*
	GLM-5	W0 基线	13	–	–	–	–	–	–
		W1 单次遍历 DRV	13	1.00	2.00	0.92	0.00	1.00	*0.23*
		W2 迭代 DRV	13	1.00	2.00	0.92	0.00	1.00	*0.77*
PHP	ChatGPT-5	W0 基线	6	–	–	–	–	–	–
		W1 单次遍历 DRV	6	1.00	0.00	1.50	0.00	1.00	*0.17*
		W2 迭代 DRV	6	1.00	0.00	1.50	0.00	1.00	*0.67*
	GLM-5	W0 基线	6	–	–	–	–	–	–
		W1 单次遍历 DRV	6	1.00	0.00	0.67	0.00	1.00	*0.33*
		W2 迭代 DRV	6	1.00	0.00	0.67	0.00	1.00	*0.83*

4.3 RQ3（修复可信度与主要失败模式）。

4.3.1 指标。

RQ3 评估基于测试的验证下的修复可信度。分析单位是一个失败的安全测试目标。对于每种模型–语言–工作流设置， N 表示最初失败的安全测试（目标）数量。W0 是未修复基线，W1 是单轮检测–修复–验证，W2 则是在相同预算上限（$K = 2$）下进行的有界迭代检测–修复–验证。

修复效果通过已修复进行概括，即经过修复–验证步骤后转为通过的目标数量；还通过修复率 $=$ 已修复/N进行概括。修复可信度还会考量对功能正确性的副作用。功能回归 统计补丁引入的功能回归问题，衡量方式是补丁应用并验证后，先前通过的功能测试中失败的数量。

我们在未成功目标中分析主要失败模式，其中 $N_{\text{unsucc}} = N -$已修复。在此设置下，如果目标在修复–验证步骤后仍然失败，则记为未修复 。在这些未成功目标中，错误定位 表示记录的定位判定指出，修复尝试未针对正确位置（location right=false）。错误定位以错误定位数 （计数）和错误定位率 $=$ 错误定位数/N_{unsucc}汇总。语义漂移或新引入的安全问题等更细粒度的失败类别，需要当前记录的信号之外的额外证据，因此本研究未对其进行量化。

表 13 基于测试的验证下的修复结果与副作用。对于每种模型–语言–工作流设置，N 表示最初未通过的安全测试（目标）数量。Fixed 和 FixRate 分别概述修复–验证步骤后已修复的目标数量和比例，而 FuncReg 则统计补丁引入的功能性回归问题。

Lang	LLM	工作流	# 安全 Test	固定	修复率	FuncReg
Python	ChatGPT-5	W0 基线	7	0	0.00	–
		W1 单次遍历 DRV	7	0	0.00	0
		W2 迭代 DRV	7	4	0.57	0
	GLM-5	W0 基线	7	0	0.00	–
		W1 单次遍历 DRV	7	3	0.43	0
		W2 迭代 DRV	7	6	0.86	0
JS	ChatGPT-5	W0 基线	13	0	0.00	–
		W1 单次遍历 DRV	13	7	0.54	0
		W2 迭代 DRV	13	11	0.85	1
	GLM-5	W0 基线	13	0	0.00	–
		W1 单次遍历 DRV	13	3	0.23	0
		W2 迭代 DRV	13	10	0.77	1
PHP	ChatGPT-5	W0 基线	6	0	0.00	–
		W1 单次遍历 DRV	6	1	0.17	0
		W2 迭代 DRV	6	4	0.67	1
	GLM-5	W0 基线	6	0	0.00	–
		W1 单次遍历 DRV	6	2	0.33	0
		W2 迭代 DRV	6	5	0.83	0

4.3.2 研究发现。

表 13 汇总了基于测试的验证下的修复结果。未修复基线（W0）在任何设置下都未能解决任何失败的安全测试目标（FixRate$= 0.00$）。

在不同语言和模型中，有界迭代式 D–R–V（W2，$K = 2$）始终比单轮工作流（W1）取得更高的修复率。对于 ChatGPT-5，Python 中的修复率从 0.00 提升至 0.57，JavaScript 中则从 0.54 提升至 0.85；对于 GLM-5，Python 中的修复率从 0.43 提升至 0.86，JavaScript 中则从 0.23 提升至 0.77。PHP 中也观察到类似的提升。这些结果表明，在相同预算上限下，有界迭代中的验证反馈能够将更多原本失败的目标转变为通过目标。

功能性副作用并不常见，但也并非不存在。对于两种模型，Python 中均未观察到功能回归。相比之下，两种模型在 JavaScript 中均观察到 W2 导致的一处回归问题；ChatGPT-5 在 PHP 中也观察到 W2 导致的一处回归问题。总体而言，在该基准测试中，迭代修复提升了经验证的安全修复效果，同时基本控制住了功能回归。

主要失败模式是在未成功目标（N_{unsucc}）中评估的，即验证后安全测试仍未通过（未修复）。表 14 显示，剩余的大多数失败都与修复目标定位错误有关（错误定位，location right=false）。在各种设置下，错误定位率都很高，在 N_{unsucc}中通常超过 0.75，这表明修复未成功更常受定位不准确的限制，而非缺少补丁。若要对定位之外的失败进行更细致的分类（例如语义漂移或新引入的安全问题），则需要额外证据；本文未对此进行量化。

表 14 验证下的主导失败模式。对于每种模型–语言–工作流设置，N_{unsucc} 表示经过修复–验证步骤后仍未通过的安全测试目标数量。WrongLoc 统计修复定位错误（`location right`=false）的目标数量（在 N_{unsucc}中），WrongLocRate 计算为 WrongLoc/N_{unsucc}。

Lang	LLM	工作流	N_{unsucc}	错误定位数	错误定位率
Python	ChatGPT-5	W1 单次遍历 DRV	7	6	0.86
		W2 迭代 DRV	3	2	0.67
	GLM-5	W1 单次遍历 DRV	4	3	0.75
		W2 迭代 DRV	1	0	0.00
JS	ChatGPT-5	W1 单次遍历 DRV	6	6	1.00
		W2 迭代 DRV	2	2	1.00
	GLM-5	W1 单次遍历 DRV	10	9	0.90
		W2 迭代 DRV	3	3	1.00
PHP	ChatGPT-5	W1 单次遍历 DRV	5	5	1.00
		W2 迭代 DRV	2	2	1.00
	GLM-5	W1 单次遍历 DRV	4	3	0.75
		W2 迭代 DRV	1	0	0.00

RQ3： 迭代式 D–R–V（W2，$K = 2$）显著增加了经验证的修复，且功能回归很少；而在验证下，剩余的主要失败模式是错误定位——大多数未修复目标都与修复目标定位错误有关（错误定位率通常为 0.67–1.00，范围在 N_{unsucc} 内）。

5 讨论

本节将 RQ1–RQ3 的结果联系起来，并探讨这些结果对大语言模型辅助漏洞管理的意义。

5.1 迭代过程中的流水线级行为

迭代将修复从一次性补丁转变为一系列小幅修改，并对每次修改进行测试。在实践中，只有验证信号清晰易读时，这个循环才有用：安全测试表明目标利用路径是否已无法触达，功能测试则表明预期行为是否仍然成立。有界预算使这一循环切实可行：它避免无休止地调整，同时允许在后续编辑不再改善经验证的结果时提前停止。

在不同场景下，迭代能否奏效主要取决于两个因素：检测指引是否明确，以及由此引发的编辑规模。当报告指出具体位置，且修复局限于局部时，反复迭代往往能取得可衡量的进展，并维持功能稳定。当报告含糊不清时，修复往往会扩散到多个文件和接口；随后几轮迭代便会耗费精力去消除副作用或追查新的症状（回归问题、契约漂移），而非消除原始原因。因此，不能孤立地讨论迭代：验证、提示粒度与编辑局部性相互关联。

5.2 检测作为修复指导

检测是从“某处出错”到“修改这段代码”的交接。只有当报告能够被直接付诸行动时，它才是有用的修复指导。仅仅指出类别是不够的；报告需要（i）

问题发生的位置，（ii）如何触发该问题，以及（iii）安全行为应当是什么样。当这些要素齐备时——例如，具体的端点、参数以及访问边界——修复步骤更有可能触及正确的组件，并产生测试能够确认的变更。

无用的报告以熟悉的方式失败。有些停留在警告级别，从不落实到某条代码路径。另一些则把正确的标签与错误的叙述混在一起，描述的场景与已实现的工作流并不相符。在这两种情况下，修复都变成了猜测：改动落在错误的位置，或者“安全加固”扩散过广并破坏无关行为。一个实用的结论是，应将检测输出视为假设，而非决策。最可靠的报告看起来像是小型、可测试的主张，包含位置、触发条件和预期结果。

5.3 验证下的修复可信度

验证是信任一次修复的主要依据。安全检查提供证据，表明在所测试的威胁模型下，所报告的故障不再可复现；功能测试则提供证据，表明预期的工作流仍然可用。两者都是必要的，因为许多安全修复会触及控制流、验证或授权，而这些恰恰是可能悄然改变行为的领域。

修复之后，结果往往聚成三类。最好的情况是干净的成功：安全故障消失，功能保持完好。第二种情况改善了安全信号，却破坏了功能，通常是因为修复过度限制了访问、把输入检查收紧到超出合法请求所需，或者更改了在别处使用的接口。第三种情况在代码中看起来合理，但并未改变已验证的结果：易受攻击路径仍然可达，修复不完整，或者故障只是被转移而非解决。换言之，在测试同时确认缓解措施与稳定性之前，补丁应被视为临时性的。

5.4 对实践和基准测试的启示

对于从业者而言，结果指向一条简单明了的工作流规则：让验证驱动循环。在每次修复尝试后运行安全测试和功能测试，可以让迭代成为基于证据的安全加固，而不是猜测性的编辑。提示粒度也会影响日常可靠性。项目级上下文有助于推理跨模块工作流和访问边界；函数级上下文则自然限制了编辑的影响范围。需求级提示通常介于两者之间：它们以角色和受保护资源为修复提供依据，但在定位方面仍留有歧义空间。

对于基准测试，本研究支持采用项目级、以测试为依据的评估。可运行的项目配合可执行的功能与安全测试，能让安全且正确的结果变得可观察，也更容易将失败归因于具体行为。在不同实现中保持共享的功能规范，也有助于在跨语言环境下维持语义可比性。扩展基准测试主要需要拓宽覆盖范围：增加更丰富的功能、涵盖更多漏洞类别，并配备具有稳定判定预言机的匹配测试。

6 效度威胁

6.1 内部效度

内部效度关注的是，观察到的差异是否源于工作流变体，而非实验设置。

基于测试的结果与预言机充分性。安全且正确产出率通过可执行的判定预言机进行评估：功能测试用于正确性，针对性安全测试以及修复后再检测用于安全。因此，结果取决于测试套件所覆盖的内容。功能测试可能会遗漏边界情况

情况或隐式需求，而安全测试仅反映其所编码的威胁模式。为使比较保持公平，在所有变体中以及每次修复迭代之后，均使用相同的项目、测试套件和验证流程。

归因新引入的安全失败。当基线安全通过率接近零时，很难量化新引入的安全失败，因为修复后可用于检测进一步退化的信号有限。在这种情况下，分析侧重于修复成功（相较基线的改进）以及通过功能测试捕获的功能回归。这样可以避免对信息不足的基线过度解读，但也限制了我们对修复引入新安全问题的频率所能作出的判断，超出重新检测和观察到的测试失败所能表明的范围。

非确定性与过程效应。大语言模型的输出和自动检测可能会因随机性以及看似微小的提示词或执行差异而有所不同。为减少不同运行之间的偶然影响，所有变体均在预算限制、有限迭代和相同停止规则下进行评估，同时保持测试套件不变。仍存在一定的实例级方差，我们将其视为所测量工作流行为的一部分。

6.2 外部效度

外部效度关注研究发现能在多大程度上推广到所研究的项目、提示词和测试套件之外。

基准项目的范围。该基准测试包含可运行的Web应用项目，并配有功能性测试与安全测试，从而支持项目级验证。这些研究发现不应被假定可推广到其他领域（例如系统软件、移动/嵌入式应用或智能合约）。同样，安全结论与当前安全测试所覆盖的漏洞模式以及用于重新检测的检测器密切相关。

提示粒度。 提示在项目级、需求级和函数级上有所变化，反映了从全项目生成到功能实现和局部修补的常见使用模式。其他交互方式——例如长时间的多轮协同设计、工具增强规划或大规模面向重构的提示——未被涵盖，并可能导致不同的动态。

语言、框架与生态系统。该基准测试涵盖多种语言和Web技术栈，降低了对单一生态系统的依赖。尽管如此，在常见漏洞形式和验证预言机有所不同的其他框架、依赖格局或安全关键运行时下，结果仍可能发生变化。

功能和测试覆盖的演进。功能性和安全测试捕捉当前项目快照的预期行为与威胁模式。随着功能不断增加或新的漏洞类别被纳入，这两套测试套件都必须不断演进，以维持覆盖范围。通过更丰富的功能、更广泛的漏洞类别以及相匹配的测试来扩展基准测试，有望提升通用性，并支持更细粒度的分析。

7 结论

本工作将大语言模型辅助编码置于漏洞管理框架中，并将安全加固具体化为一种以测试为基础的迭代式检测–修复–验证工作流。本文引入了一个用于端到端评估的项目级基准测试数据集，其中包含可运行的 Web 应用项目，以及与之配套的可执行功能测试和针对性安全测试，并支持三种提示粒度（项目级、需求级和函数级）。利用该基准测试，研究评估了在可比预算下流水线级的安全且正确的产出率，考察了检测报告的可靠性与修复可操作性，并刻画了修复后的输出在验证过程中的可信度，包括主要

失效模式。未来工作将通过增加更丰富的功能、扩大对其他漏洞类别的覆盖范围，并添加相应的测试来扩展该基准测试，从而实现更全面、更细粒度的评估。

声明

资金支持

本研究未获得任何外部资金支持。

伦理审批

不适用。

知情同意

不适用。

作者贡献

Cheng Cheng：构思、方法论、数据收集、撰写—初稿、审阅与编辑。

数据可用性声明

T复制包可在此处获取：https://github.com/Hahappyppy2024/EmpricalVD R.

利益冲突

作者声明，他们没有任何竞争性利益。

临床试验编号

临床试验编号：不适用。

参考文献

Cao S, Zhang Z, 等（2024）Coca：改进并解释基于图神经网络的漏洞检测系统。载于：第46届国际软件工程大会（ICSE）论文集，ACM。Cao X, Wang J, Wu P（2025）通过提取和匹配修复模式增强漏洞修复。《系统与软件杂志》230:112528，DOI 10.1016/j.jss.2025.112528。Cheng C, Yang J（2024）基于大语言模型的代码生成安全性基准测试。Cheng C, Yang J（2025）Cfceval：评估大语言模型生成代码的安全性。载于：2025年第2届IEEE/ACM人工智能驱动软件国际会议（AIware），IEEE，第01–10页。

Fakih 等（2025）Llm4cve：借助大语言模型实现迭代式自动化漏洞修复。载于：IEEE/ACM国际软件工程大会论文集。de Fitero-Dominguez D, Garcia-Lopez E, Garcia-Cabot A, Martinez-Herraiz JJ（2024）利用大语言模型增强自动化代码漏洞修复。DOI 10.48550/arXiv.2401.03741，网址 https://arxiv.org/abs/2401.03741，期刊信息：《人工智能工程应用》，138（A辑），109291（2024），2401.03741。Fu M, Tantithamthavorn C, Lo D（2022）Vulrepair：一种基于T5的自动化软件漏洞修复方法。载于：第37届IEEE/ACM国际自动化软件工程会议（ASE）论文集。He J, Vero M, Krasnopolska G, Vechev M（2024）面向安全代码生成的指令微调。载于：第41届国际机器学习会议（ICML）论文集，DOI 10.5555/3692070. 3692793。Hu W, Shu Y, Yu Z, Wu Z, Lin X, Dai Z, Ng SK, Low BKH（2024）局部零阶提示优化。载于：神经信息处理系统进展（NeurIPS）。Hu Y, Li Z, Shu K, Guan S, Zou D, Xu S, Yuan B, Jin H（2025）SoK：自动化漏洞修复：方法、工具与评估。载于：Bauer L, Pellegrino G（编），第34届USENIX安全研讨会（USENIX Security 2025），美国华盛顿州西雅图，2025年8月13–15日，第4421–4440页，网址 https://www.usenix.org/conference/usenixsecurity25/presentation/hu-yiwei。Khattab O, Singhvi A, Maheshwari P, Zhang Z, Santhanam K, Vardhamanan S, Haq S, Sharma A, Joshi TT, Moazam H, Miller H, Zaharia M, Potts C（2024）DSPy：将声明式语言模型调用编译为自我改进流水线。载于：国际学习表征会议（ICLR）。Kim Y, 等（2025）输入日志，输出补丁：通过思维树大语言模型分析实现自动化漏洞修复。载于：第34届USENIX安全研讨会论文集（USENIX Security 25），网址 https://www.usenix.org/conference/usenixsecurity25/presentation/kim-youngjoon。Kojima T, Gu SS, Reid M, Matsuo Y, Iwasawa Y（2022）大语言模型是零样本推理者。载于：神经信息处理系统进展（NeurIPS）。Kulsum U, Zhu H, Xu B, d’Amorim M（2024）大语言模型用于自动化漏洞修复的案例研究：评估推理和补丁验证反馈的影响。DOI 10.48550/arXiv.2405.15690，网址 https://arxiv.org/abs/2405.15690，2405.15690。Li W, Zhou Y, 等（2022a）MVD：基于流敏感图神经网络的内存相关漏洞检测。载于：第44届国际软件工程大会（ICSE）论文集，ACM。Li Z, Zou D, Xu S, Chen Z, Zhu Y, Jin H（2022b）VulDeeLocator：一种基于深度学习的细粒度漏洞检测器。《IEEE可信与安全计算汇刊》19(4):2821–2837，DOI 10.1109/TDSC.2021.3076142。Liu C, Chen X, Li X, Xue Y（2024）让漏洞预测更实用：预测、分类与定位。《信息与软件技术》171:107458，DOI 10.1016/j.infsof.2024.107458。Liu P, Yuan W, Fu J, Jiang Z, Hayashi H, Neubig G（2023）预训练、提示与预测：自然语言处理中提示方法的系统综述。《ACM计算调查》，DOI 10.1145/3560815。OWASP基金会（2025）OWASP Top 10:2025。https://owasp.org/www-project-top-ten/，访问日期：2026-02-28。Pearce H, Tan B, Ahmad B, Karri R, Dolan-Gavitt B（2021）探究使用大语言模型进行零样本漏洞修复。网址 https://arxiv.org/abs/2112.02125，2112.02125。Pearce H, Ahmad B, Tan B, Dolan-Gavitt B, Karri R（2022）键盘前的沉睡者？评估GitHub Copilot代码贡献的安全性。载于：2022年IEEE安全与隐私研讨会（SP），DOI 10.1109/SP46214.2022.9833571。Peng J, Cui L, Huang K, Yang J, Ray B（2025）CWEval：对大语言模型代码生成功能性与安全性的结果导向评估。载于：2025年IEEE/ACM大语言模型国际研讨会。

代码模型（LLM4Code），DOI 10.1109/LLM4Code66737.2025.00009 Perry N, Srivastava M, Kumar D, Boneh D（2023）使用人工智能助手的用户会编写更多不安全的代码吗？载于：2023年ACM SIGSAC计算机与通信安全会议论文集（CCS '23），DOI 10.1145/3576915.3623157 Qu M, Liu J, Kang L, Ling S, Wang S, Ye D, Huang T（2025）Scodegen：用于基于大语言模型进行安全代码生成的实时可信约束解码框架。载于：2025年 IEEE 第24届计算、信任、安全与隐私国际会议（TrustCom），第492–503页，DOI 10.1109/Trustcom66490.2025.00061 Siddiq ML, Santos JCS（2022）SecurityEval数据集：挖掘漏洞示例以评估基于机器学习的代码生成技术。载于：第1届软件仓库挖掘隐私与安全应用国际研讨会论文集（MSR4P&S '22），第29–33页，DOI 10.1145/3549035.3561184 Siddiq ML, Santos JCS, Devareddy S, Muller A（2024）Sallm：生成代码的安全评估。载于：第39届 IEEE/ACM 自动化软件工程国际会议研讨会论文集（ASEW '24），DOI 10.1145/3691621.3694934 Singla S, Wang Z, Liu T, Ashfaq A, Hu Z, Xing EP（2024）动态奖励结合提示优化，使语言模型无需微调即可实现自对齐。载于：2024年自然语言处理实证方法会议论文集（EMNLP），第21889–21909页 Tian W, Lin Y, Gao X, Sun H（2025）增强漏洞定位：协调任务专用微调与通用大语言模型提示。载于：第41届 IEEE 软件维护与演化国际会议论文集（ICSME），DOI 10.1109/ICSME64153.2025.00007，已录用/会议论文 Ullah S, Han M, Pujar S, Pearce H, Coskun A, Stringhini G（2024）大语言模型尚不能可靠地识别安全漏洞并进行推理（目前如此？）：全面的评估、框架与基准测试。载于：IEEE安全与隐私研讨会论文集（S&P），第862–880页，DOI 10.1109/SP54263.2024.00210，网址 https://hdl.handle.net/2144/49910 Vero M, M¨undler N, Chibotaru V, Raychev V, Baader M, Jovanovi´c N, He J, Vechev M（2025）Baxbench：大语言模型能否生成正确且安全的后端？载于：第42届国际机器学习会议论文集（ICML）Wan X, Sun R, Nakhost H, Arik SO（2024）教得更好还是展示得更巧？论自动提示优化中的指令与示例。载于：神经信息处理系统进展（NeurIPS）Wang P, Liu X, Xiao C（2025）CVE-Bench：基准测试基于大语言模型的软件工程智能体修复真实世界CVE漏洞的能力。载于：2025年美洲计算语言学协会北美分会会议：语言技术论文集（NAACL 2025），第1卷：长篇论文，第4207–4224页，网址 https://aclanthology.org/2025.naacl-long.212/ Wang X, Wei J, Schuurmans D, Le QV, Chi EH, Narang S, Chowdhery A, Zhou D（2023a）自一致性提升了语言模型中的思维链推理。载于：国际学习表征会议（ICLR）Wang Z, Chen K, 等（2023b）Defiwarder：保护去中心化金融应用免受代币级漏洞侵害。载于：第38届 IEEE/ACM 自动化软件工程国际会议论文集（ASE），IEEE/ACM Wei J, Wang X, Schuurmans D, Bosma M, Ichter B, Xia F, Chi EH, Le QV, Zhou D（2022）思维链提示能够引导大语言模型进行推理。载于：神经信息处理系统进展，第35卷 Wei Z, Zeng J, Wen M, Yu Z, Cheng K, Zhu Y, Guo J, Zhou S, Yin L, Su X, Ma Z（2025）Patcheval：用于评估大语言模型修复现实世界漏洞能力的新基准测试。arXiv:2511.11019，网址 https://arxiv.org/abs/2511.11019 Yang L, Yu Z, Zhang T, Cao S, Xu M, Zhang W, Gonzalez JE, Cui B（2024）思想缓冲区：利用大语言模型进行思维增强推理。载于：神经信息处理系统进展

（NeurIPS）Yao S, Yu D, Zhao J, Shafran I, Griffiths T, Cao Y, Narasimhan K（2023）思维树：利用大语言模型进行审慎的问题求解。载于：神经信息处理系统进展（NeurIPS）Zhang G, Yao T, Wang H, Qin J, Li Y, Ma Q, Sun D（2025）基于细粒度线索优化与局部敏感微调的自动程序漏洞修复。SSRN，DOI 10.2139/ssrn.5090066，发布于：2025年1月9日 Zhang J, Liu S, Wang X, Li T, Liu Y（2023）学习定位和描述漏洞。载于：第38届 IEEE/ACM 自动化软件工程国际会议论文集（ASE），第1234–1245页，DOI 10.1109/ASE56229.2023.00045 Zhang L, Zou Q, Singhal A, Sun X, Liu P（2024a）评估大语言模型修复 C/C++ 代码中的现实世界漏洞的能力。技术报告，美国国家标准与技术研究院（NIST），网址 https://www.nist.gov/publications/evaluating-large-language-models-real-world-vulnerability-repair-cc-code Zhang R, Liu Y, 等（2024b）CFExplainer：针对基于图神经网络的漏洞检测的反事实解释。载于：ACM计算机与通信安全会议论文集（CCS），ACM Zhang Y, Li M, 等（2024c）STASE：用于检测配置错误和漏洞的静态污点分析框架。载于：第39届 IEEE/ACM 自动化软件工程国际会议论文集（ASE），IEEE/ACM Zhou D, Sch¨arli N, Hou L, Wei J, Scales N, Wang X, Schuurmans D, Cui C, Bousquet O, Le Q, Chi EH（2023）由简到繁提示使大语言模型能够进行复杂推理。载于：国际学习表征会议（ICLR）