通过检索增强生成和多工具反馈改进大语言模型辅助的安全代码生成

VidyutSriram Sawan Pandita Achintya Lakshmanan宾夕法尼亚州立大学
PennsylvaniaStateUniversity Pennsylvania StateUniversity vps5353@psu.edu
sbp5882@psu.edu aql6062@psu.edu

Aneesh Shamraj宾夕法尼亚州立大学
avs8051@psu.edu

Suman Saha
宾夕法尼亚州立大学
sumsaha@psu.edu

arXiv:2601.00509v1 [cs.CR] 1 Jan 2026

摘要

大语言模型（LLM）能够生成代码，但往往会引入安全漏洞、逻辑不一致和编译错误。先前的研究表明，大语言模型能够从结构化反馈、静态分析、检索增强和基于执行的细化中显著获益。我们提出了一种检索增强的多工具修复工作流，其中，单个代码生成大语言模型通过编译器诊断信息、CodeQL 安全扫描和 KLEE 符号执行，迭代地细化其输出。系统使用轻量级嵌入模型检索以往成功的修复案例，提供面向安全的示例以指导代码生成。在由 **DeepSeek-Coder-1.3B** 和 **CodeLlama-7B** 生成的 3,242 个程序组成的组合数据集上进行评估后，系统展现出显著的稳健性提升。对于 DeepSeek，安全漏洞减少了 96%。对于规模更大的 CodeLlama 模型，严重安全缺陷率从 58.55% 降至 22.19%，凸显了工具辅助自我修复的有效性，即使面对“顽固”的模型也是如此。

1 引言

大语言模型（LLM）迅速推动了自动代码生成的发展，但如果缺乏结构化的外部指导，其输出仍然高度不可靠。大量研究表明，大语言模型生成的程序经常遭遇编译失败、逻辑不一致以及严重的安全漏洞。例如，Pearce 等人（2022）表明，GitHub Copilot 在超过 40% 的测试案例中生成了不安全的代码，通常会引入缓冲区溢出、不安全的输入处理或密码学误用。这些发现表明，除非配备明确的安全机制，否则大语言模型往往会复现训练数据中的不安全模式。

Zheng 等人（2025）提供了更为广泛的视角，他们提出了迄今为止最全面的基于大语言模型的自动程序修复（APR）分类体系。对 63 个自动程序修复系统的调查表明，单次大语言模型输出很少足以保证正确性或鲁棒性。相反，最成功的系统会将静态分析器、执行引擎和检索模块等外部工具纳入程序化修复工作流。这一观察结果促使我们采用当前方法：我们不将大语言模型视为独立的生成器，而是将其嵌入多步骤验证与自修复循环中。

基于执行的细化也已证明能够有效提升代码的可靠性。Xu 等人(2023)提出了 Self-Debug 系统，该系统将编译器和运行时错误信息反馈给模型，以迭代修复错误程序。他们的结果表明，通过结构化反馈，程序正确性显著提升，这支持了以下假设：当大语言模型获得可执行的诊断信号时，便能够自我纠正。STaR 等关于迭代推理的补充研究，由 Zelikman 等人 (2022)提出，进一步证明了多步细化能够显著提升模型在不同领域的表现。

受这些发现的启发，我们针对 DeepSeek-Coder-1.3B 和 CodeLlama-7B 这两个特点迥异的大语言模型，设计了一套用于安全代码生成的检索增强、工具引导式自我修复工作流。

我们的方法的特点在于，将三种互补机制紧密整合到单一的推理时循环中：检索增强上下文、多工具验证以及迭代式自我修复。检索提供成功实施安全修复的先前示例，减少重复使用不安全惯用写法的倾向；基于工具的分析则提供具体的诊断信息，揭示语法、语义和安全缺陷；迭代式自我修复使模型能够调整其输出

以响应这些信号，而无需微调或额外的生成式智能体。通过结合这些信号，该工作流能够应对各不相同且无法由任何单一组件充分处理的失效模式，为依赖大量智能体或训练密集型方法提供了一种切实可行的替代方案。

2 文献综述

文献一致强调，大语言模型在代码生成方面存在两个关键失误：(1) 倾向于生成语法错误或无法编译的代码；(2) 经常引入隐蔽或严重的安全漏洞。

Pearce 等人（2022）开展了最早的系统性大语言模型安全评估之一，表明 Copilot 生成的代码通常包含内存安全违规、未检查的输入流、整数溢出及其他关键缺陷。他们提出的漏洞分类体系促使我们集成 CodeQL，以精准检测这些问题类别。

除了安全缺陷之外，多项研究表明，大语言模型难以理解更深层次的程序语义。Xu 等人（2023）证明，在迭代使用符号执行和编译器反馈时，代码正确性会得到显著提升。他们关于“Self-Debug”的研究结果指导了我们对 KLEE 的集成；KLEE 能够发现通过简单测试用例评估无法触及的缺陷。

Zelikman 等人（2022）提出了 STaR，这种方法表明，与单次推理相比，让大语言模型置于迭代优化循环中能够显著提升其表现。这支持了我们的设计理念：安全代码生成需要多步推理，而不是单次生成输出。

最后，Zheng 等人 (2025提供的全面分类体系对 63 个自动程序修复系统进行了分类，并得出结论：最成功的系统是工具增强型的，而非纯粹由提示驱动的系统。他们认为，检索增强修复和静态分析引导的工作流是尤其有前景的方向。他们的研究结果直接支持我们将检索增强生成（RAG）与 CodeQL 和 KLEE 相结合，突出了可靠的代码生成需要多工具编排，而不是纯粹的大语言模型采样。

3 方法

我们 的系统将编译器诊断信息、符号执行、静态分析和检索式指导整合到一个统一的自修复框架中，该框架围绕单个代码生成大语言模型构建。轻量级句子嵌入模型（all-MiniLM-L6-v2）对包含以往成功修复案例的数据库执行语义检索，并返回面向安全的示例，将其注入提示中。检索模型不会生成或修改代码；它仅用于选择相关的上下文示例。

活跃的大语言模型（DeepSeek-Coder-1.3B 或 CodeLlama-7B，具体取决于实验）根据任务描述和检索到的示例生成候选代码。生成的程序会接受编译器诊断（GCC）、符号执行（KLEE）和静态分析（CodeQL）。如果检测到任何错误——包括编译失败、未通过的 KLEE 断言或 CodeQL 安全警报——诊断消息会附加到提示中，并在迭代式自我修复循环中反馈给同一个大语言模型。该过程针对每个提示最多重复三次。

当程序通过所有检查后，最终代码及其相关上下文会被嵌入并存回 RAG 数据库。在后续生成过程中，系统会检索语义相似且修复成功的案例，用来引导大语言模型从一开始就采用更安全、更稳健的模式。

图 1总结了这一工作流，展示了检索、编译、符号执行和安全分析如何围绕单个大语言模型形成闭环反馈回路。

4 错误分类与数据集

我们在总计 3,242 个生成的程序上评估了该工作流（其中 1,522 个由 DeepSeek 生成，1,720 个由 CodeLlama 生成）。每个样本均从三个错误维度进行标注：

- **编译错误：** 程序因语法或类型问题而无法编译。
- **安全错误：** CodeQL 报告至少一项严重安全问题（例如 cpp/unbounded-write、未检查的输入流或整数溢出）。
- **语义错误：** 程序可以编译，但未能通过所提供测试框架中的基于 KLEE 的语义检查。

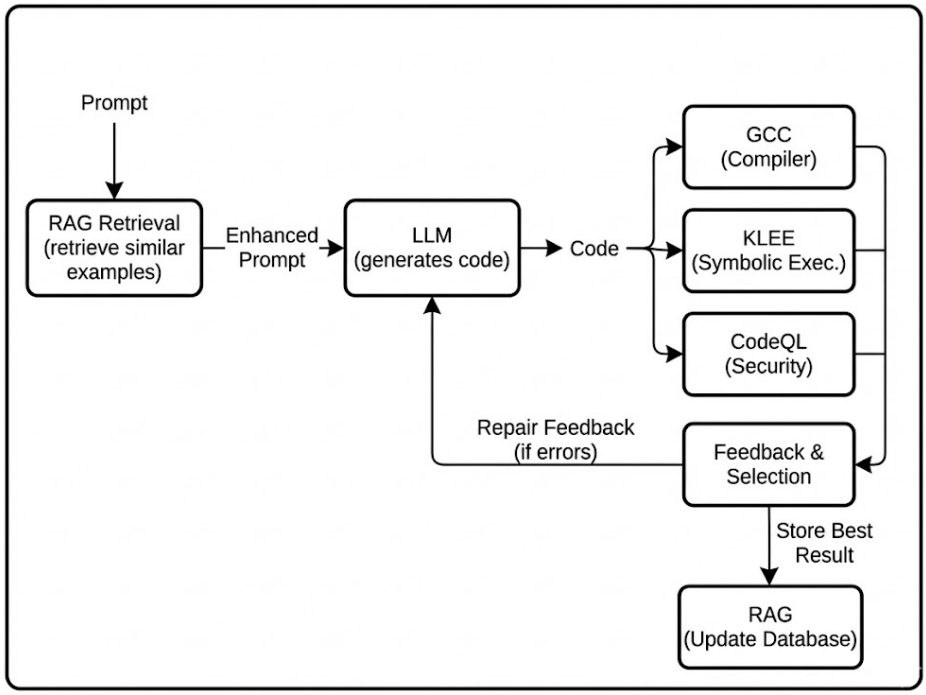


图 1：所提出的多工具反馈系统框图。轻量级嵌入模型从 RAG记忆库中检索面向安全的示例，以补充提示。单个代码生成大语言模型生成候选代码，然后通过编译器诊断（GCC）、符号执行（KLEE）和静态分析（CodeQL）对其进行分析。诊断反馈会迭代地反馈给同一个大语言模型，成功修复的代码则会存回 RAG 数据库。

指标	基线	修复循环	改进
编译	39.79%	20.43%	19.36% Red.
安全	36.35%	1.45%	34.90% 降低。
语义	60.09%	5.72%	54.37% 降低。

表 1：DeepSeek-1.3B 性能对比。

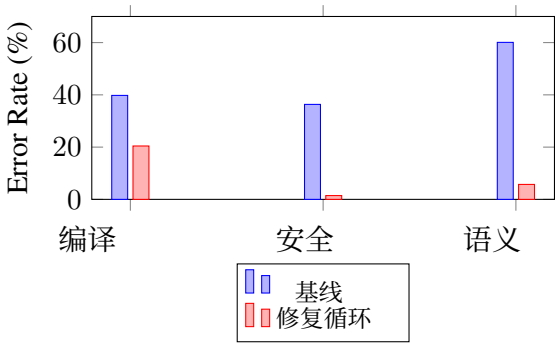


图2：DeepSeek 在各类别中的错误减少情况。

5 结果

5.1 DeepSeek-Coder-1.3B 的性能

启用 DeepSeek 模型的完整工作流后（1,522 个程序），我们观察到了显著改进。编译成功率提高了 19.4 个百分点。最值得注意的是，系统达到了**98.6% 的安全清洁率**（基线约为 64%），几乎消除了所有漏洞。

5.2 CodeLlama-7B 的性能

我们将评估扩展到更大的 CodeLlama-7B 模型（1,720 条提示词）。如表2所示，反馈回路带来了显著改进，不过与 DeepSeek 相比，更大的模型表现出不同的行为。

分析：RAG系统有效缓解了 CodeLlama 编写无法编译的代码的倾向，使失败率几乎降低了一半。在安全性方面，基线 CodeLlama 模型被认定为 “ 逻辑正确，但存在严重安全隐患”（缺陷率为 58.55%）。反馈回路成功将这一比例降至 22.19%。

5.3 对比分析：“顽固模型”现象

对这两个模型进行比较，揭示了一项关键发现。DeepSeek（1.3B）最初生成的代码较为简单，但安全性较低——

指标	交付物 1	交付物 2	改进
编译	49.71%	28.72%	20.99% 降低。
安全	58.55%	22.19%	36.36% 降低。
语义	19.98%	12.18%	降低 7.80%。

表 2：CodeLlama-7B 性能对比。

最终错误率	DeepSeek（1.3B）	CodeLlama（7B）
编译	20.43%	28.72%
安全	**1.45%**	**22.19%**
语义	5.72%	12.18%

表 3：最终系统性能比较（交付成果 2）。

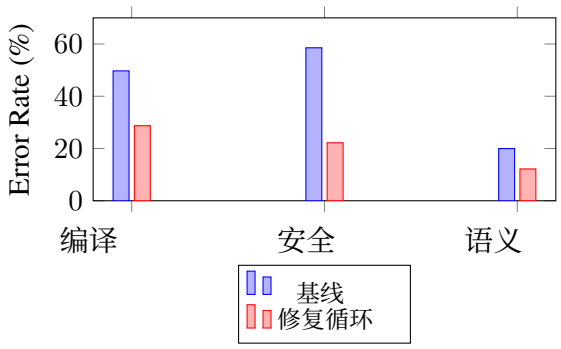


图 3：CodeLlama-7B 在各类别中的错误减少情况。

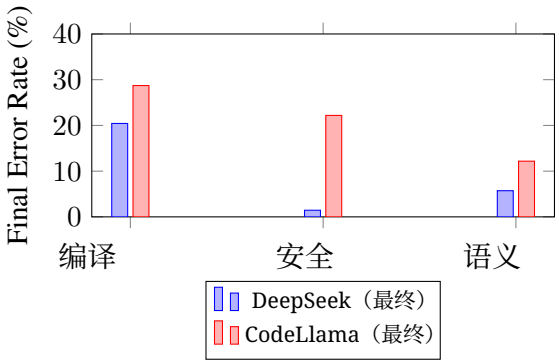


图 4：最终系统的性能比较。

因此在工具反馈下具有很强的可调整性，最终实现了近乎完美的安全性（错误率为 1.45%）。相比之下，CodeLlama（7B）生成的代码模式更加复杂且极不安全，即使经过修复，安全错误率仍高达 22.19%。

6 局限性与未来工作

尽管在量化指标上取得了显著提升，我们的研究仍存在若干局限。首先，评估仅针对一种问题分布：来自竞赛编程风格任务的简短 C/C++ 程序。现实世界的软件系统具有更丰富的结构、更庞大的代码库以及长期存在的有状态交互，这些特征可能会暴露出不同类型的安全漏洞。因此，在大型工业级代码库中，报告的错误率降幅可能高估了实际性能。

其次，我们的工作流聚焦于三个分析渠道——编译、CodeQL 和 KLEE——其覆盖范围本质上并不完整。CodeQL 规则针对有限的一组漏洞模式，而 KLEE 只能探索在给定测试框架和资源预算下可达的路径。未检测到的缺陷、侧信道泄漏或并发问题仍不在研究范围内。同样，我们将修复循环限制为三次迭代；增加迭代次数或许能进一步提升鲁棒性，但代价是更高的延迟和计算开销。

第三，我们的 RAG 组件使用相对简单的机制对过去的修复方案进行检索。检索通过轻量级嵌入模型（all-MiniLM-L6-v2）执行，该模型能够选择语义相似的成功修复方案，但不会生成或修改代码。我们尚未学习哪些修复方案具有最佳的泛化能力，也不会根据漏洞类型、程序结构或工具反馈调整检索策略。此外，我们仅在静态基准测试上进行离线评估，未衡量我们的流水线对人类开发者的影响，例如信任、可用性或调试时间。

面向自适应修复策略的强化学习。 一个有前景的未来方向是引入强化学习，利用流水线已生成的工具反馈信号来优化修复策略。编译成功（GCC）、语义正确性（KLEE）和安全性洁净度（CodeQL）自然构成稀疏但有意义的奖励信号，而修复迭代次数则直接反映延迟和计算惩罚。强化学习无需对底层模型权重进行微调，而是可以应用于编排层，以学习自适应修复策略，例如优先处理哪些诊断信号、采取多大力度修改代码（如选择最小补丁还是结构性修改），以及何时提前终止。这种方法将使系统能够在正确性、安全性和效率之间取得平衡，同时以客观的、基于工具的验证为依据。

智能体扩展与多模型扩展。 在扩展到大型、真实世界的软件项目时，智能体或额外的模型可能会变得很有价值

这类项目通常包含许多文件、复杂的依赖关系以及长期交互。在此类场景中，智能体组件可以协助进行代码仓库导航、依赖关系分析、工具编排和变更规划（例如，选择要修改哪些文件以及修改顺序），同时确保改动规模小且易于验证。不过，为了维护安全性并避免不必要的重构，任何基于智能体的扩展都应严格受同一套验证工具（构建/测试、CodeQL 以及符号检查或动态检查）的约束，并应侧重于协调与规划，而不是不受约束的代码生成。

未来的工作将通过以下方式解决这些局限：（i）将评估扩展到多文件、多语言代码库；（ii）纳入动态模糊测试和污点追踪等更多分析方法；（iii）学习自适应修复策略，根据错误类型选择工具和提示词；（iv）开展用户研究，将系统置于真实的开发工作流中。

7 结论

我们的结果表明，检索增强与多工具反馈显著提升了不同规模模型的基于大语言模型的代码生成能力。该工作流将 DeepSeek 的安全漏洞减少了 96%，将 CodeLlama 的安全漏洞减少了 36% 以上。这些发现证实了自动程序修复和大语言模型调试领域的既有研究：与单独用作生成器相比，将模型嵌入由工具治理、反馈丰富的流水线中，才能使其发挥最佳性能。未来的工作将探索更大的模型、检索与静态分析之间更紧密的集成，以及直接优化安全性和语义正确性的强化学习目标。

参考文献

Henry Pearce、Ben Ahmad 等人。2022 年。键盘前沉睡？评估 GitHub Copilot 代码贡献的安全性。载于*IEEE*安全与隐私研讨会。

FrankXu、Xinyun Chen 等人。2023 年。Self-Debug：通过执行反馈实现自动化大语言模型调试。*arXiv* 预印本。

Eric Zelikman、Yuhuai Wu 和 Noah Goodman。2022 年。Star：通过推理引导推理能力的自举。载于神经信息处理系统进展（*NeurIPS*）。

Siyu Zheng、Yiming Wang 和其他 1 位作者。2025 年。大语言模型自动程序修复综述：分类体系、设计范式与应用。*arXiv* 预印本。