并非任何示例：用于指导大语言模型漏洞修复的安全需求与修复模式

Zichen Liu[1], Liangyi Huang[1], Stanley Nguyen[1], KhaTuNguyen[1], XindaWang[2], 和 Xusheng Xiao[1]
ArizonaStateUniversity *[2]TheUniversityofTexasatDallaszliu396@asu.edu, lhuan139@asu.edu, snguye61@asu.edu knguy170@asu.edu, xinda.wang@utdallas.edu, Xusheng.Xiao@asu.edu*

arXiv:2611.31993v1 [cs.CR] 24 Nov 2026

摘要——**软件漏洞使系统面临攻击风险，而其修复往往会被延迟。大语言模型（LLM）推动了自动化补丁生成的发展，然而，通过功能测试并不意味着补丁能够落实漏洞所违反的安全需求。现有漏洞修复技术依赖覆盖范围有限的精选模式或模板，因此哪些修复策略能够跨漏洞复用，以及如何利用这些策略引导大语言模型，仍不明确。**

我们开展了一项大规模实证研究，将反复出现的修复模式与漏洞的安全影响关联起来，并考察如何利用这些知识指导基于大语言模型的修复。在对安全补丁进行人工研究的基础上，我们对来自 348 个开源代码库的 3,816 个安全补丁进行聚类，并分析其编辑及上下文，从而推导出具体修复模式；这些模式涵盖从局部防护措施到跨多个位置的协调变更。我们依据漏洞所违反的安全需求定义了 12 个安全影响类别，并将其映射到修复类别和修复模式，揭示出各类别中的典型修复、同一类别内的替代策略，以及不同类别之间共享的模式。随后，我们利用这些模式和影响–模式关联关系，为四个大语言模型在不同复杂度的修复任务中选择修复示例。在零样本提示下，经过引导的示例选择将四模型均值 pass@3 从 37.9% 提升至最高 49.2%；相比之下，随机选择的示例仅达到 36.9%。我们的研究将漏洞表征与补丁生成联系起来，并支持围绕安全需求组织修复知识。

1. 引言

软件漏洞允许攻击者扰乱或操纵系统行为，因此及时修补至关重要。然而，运营约束、人员短缺以及相互竞争的优先事项常常导致修复延迟 [1], , 延长系统暴露于攻击之下的时间 [2], [3]。降低开发修复方案所需的工作量，是减轻这一负担的一种方式，也因此推动了通过测试驱动的补丁搜索和学习型变换实现自动化程序修复 [4], [5], [6]。然而，对于安全漏洞而言，仅通过功能测试是不够的：生成的补丁可能过拟合现有测试，却未能解决根本的安全问题 [7], [8]。有效的漏洞修补还必须恢复遭到违反的安全需求，例如输入有效性、内存边界和访问控制，同时保留预期功能。为满足这些需求，特定漏洞修复方法会在精选模式和模板中编码策略，但其覆盖范围限制了对未见漏洞类型和修复上下文的泛化能力 [9], [10]。近期大语言模型（LLM）的进展为突破这些预定义策略、拓展自动化修复提供了机会。大语言模型驱动的补丁生成在功能缺陷基准上的表现显著优于传统技术和早期的基于学习的技术 [11],, 而基于大语言模型的智能体则进一步将修复扩展到现实世界的软件问题 [12], [13]。

尽管大语言模型在功能性缺陷修复方面取得了这些进展——能够根据测试反馈推理功能需求和代码结构——但生成有效的漏洞补丁仍然充满挑战。安全需求还会对对抗性输入和执行条件下的行为施加额外约束，而功能测试通常无法覆盖这些情况 [9]。因此，漏洞修复必须在保留程序预期功能的同时，防止输入触发可利用行为，这要求采用超越满足功能测试的策略。例如，消息解析器可能在格式正确的消息上通过功能测试，却将攻击者控制的字节数复制到目标缓冲区之外。仅检查目标指针是否为非空的补丁，仍会使这一漏洞得以利用。适当的修复必须在复制前验证复制长度是否不超过目标容量，在继续处理有效消息的同时拒绝不安全输入。这一区别说明，仅凭代码生成能力是不够的：有效的漏洞修补还需要了解在目标上下文中，哪种修复策略能够恢复遭到违反的安全需求。将这些知识明确化并提供给大语言模型，可能有助于其将生成能力

将能力转化为更可靠的安全补丁。

我们对各个项目中的漏洞补丁进行了仔细检查，发现了一种组织这些修复知识的基础：具有不同语法结构的补丁往往实现相似的修复策略。例如，提前返回和条件包装器都可以防止复制操作超出其目标容量。这种共享结构促使我们挖掘修复模式，以捕捉代码变更、其所保护的操作以及这些操作适用的条件之间反复出现的关系

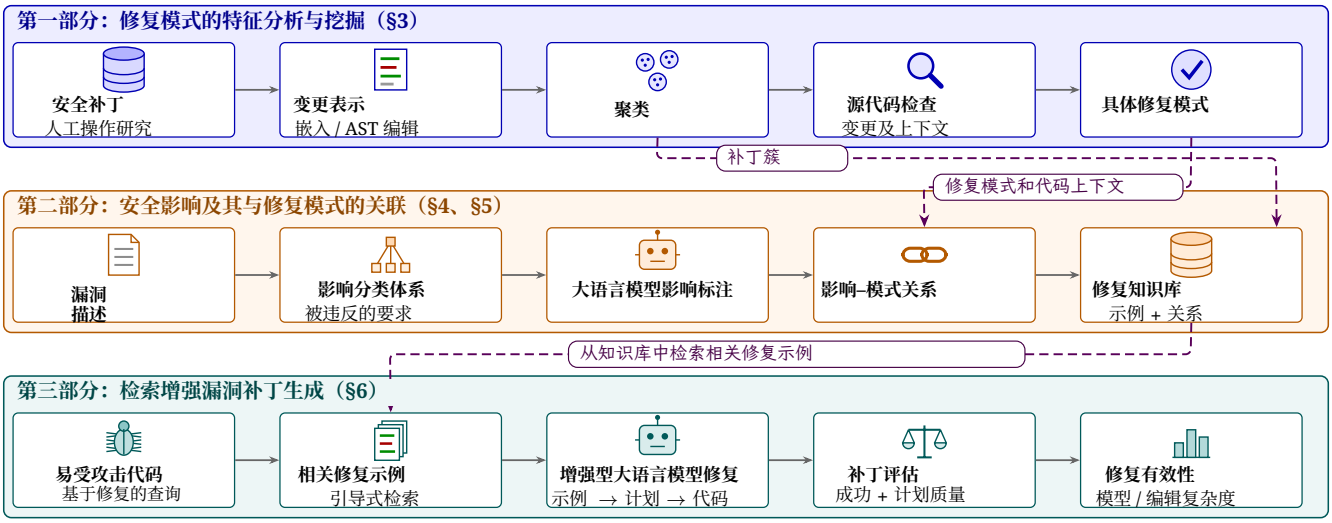


图1。研究概览：对安全补丁进行细致检查和聚类，可以揭示修复模式。安全影响分析将这些模式与被违反的要求相关联，从而形成一个支持通过相关修复示例进行检索增强补丁生成的修复知识库。评估在不同模型和编辑复杂度下，将引导式检索与零样本、随机和预言机条件进行比较。

。我们通过漏洞代码所违反的安全需求来描述其安全影响，例如内存边界或引用的有效性。将影响与模式关联起来，可以把修复必须恢复的内容与代码变更恢复该内容的方式联系起来，而目标的上下文则决定适用哪种策略。利用这些关系来组织修复知识并指导大语言模型生成补丁的潜力，仍未得到充分探索。

尽管现有研究会挖掘修复模式，并将漏洞知识融入补丁生成，但它们侧重于编辑变换、特定漏洞类别或基于 CWE 的组织方式，因此未能充分理解修复策略如何在不同代码上下文中应对被违反的安全要求。模式挖掘工具会从补丁中提取反复出现的变换，包括修复静态分析警告的变换 [14]，[15]，[16]，[17]，[18]。修复方法利用精选模式、模板和学习型变换来生成补丁 [9]，[10]，[19]。具备影响意识的研究会按影响总结安全操作和补丁模式 [20]，，在大语言模型提示中编码针对内存错误类型的预期修复 [21]，，或按 CWE 类型组织修复知识 [22]。这些视角尚未全面、大规模地说明：针对同一种安全影响有哪些替代修复策略、哪些策略可用于多种影响，以及哪些代码条件决定了这些策略的适用性。缺少这样的说明，我们仍不清楚修复模式知识和安全影响知识如何支持检索增强生成，即如何从修复知识库中仅检索与目标漏洞相关的修复示例。因此，我们开展了一项大规模研究，深入了解修复模式、它们与安全影响之间的关联关系，以及这些知识如何改进大语言模型的漏洞补丁生成。该研究旨在为后续探索如何将修复知识融入大语言模型补丁生成奠定实证基础。

挑战。开展这项研究需要应对三项挑战：识别修复模式、将其与安全影响关联起来，以及评估其在大语言模型补丁生成中的应用。❶识别不同代码形式中的修复机制。同一种编辑可能保护不同的操作，而不同编辑或协同变更也可能实现同一种模式。表示和源代码分析必须捕捉修改、其上下文与恢复的要求之间的关系。❷刻画漏洞违反的要求。描述往往会结合实现缺陷、被违反的要求和攻击后果，也可能报告多个连续发生的违规。统一的影响分类体系必须区分这些层次，并保留违规之间的关系。❸将修复知识转化为实用指导。修复示例必须传达适用于目标漏洞的修复关系。评估必须确定相关修复示例是否有益，以及检索能否针对复杂程度各异的修复获取此类指导。

大规模研究。在我们的研究中，我们首先使用 ARVO [23] 进行人工修复动作研究，然后使用 PatchDB、CVEfixes 和 MegaVul [24]，[25]，[26]进行大规模补丁分析。该研究分为三个相互关联的部分（图1）。

第一部分：刻画并挖掘修复模式。我们首先检查了 300 个安全补丁中的修复动作，并发现了反复出现的动作及其组合。这些观察促使我们归纳出广泛的修复类别，并在来自 348 个代码仓库的 3,816 个补丁中搜索更具体的模式。我们将 CodeReviewer 和 Jina 代码嵌入与 AST 编辑表示进行比较，评估簇几何结构以及相关修复类别和同模式示例的集中程度（第 3 节）。通过对簇内源代码进行比较，我们识别出具体模式，包括在导出前初始化输出、限制递归处理的边界、保持字符串终止，以及其他反复出现的修复。这些具体模式为簇中体现的修复

机制提供了可解释的证据，并补充了对补丁分组的定量评估。

第二部分：对安全影响进行分类，并将其与修复模式关联。现有的影响感知研究通常聚焦于特定漏洞类型，因此未能全面刻画更广泛的安全影响范围 [20], [21]。**我们基于遭到违反的安全需求，定义了一个更全面的分类体系，涵盖 12 种安全影响类别；该体系借鉴了并发、算法复杂度、输入解释和证书验证方面的研究，以及密码学规范** [27], [28], [29], [30], [31]。我们针对从 PatchDB、CVEfixes 和 MegaVul 收集的 348 个不同开源项目中的 3,816 个补丁开展了大规模安全影响研究，揭示了内存安全、并发、资源管理、信任决策和数据保护领域中广泛存在的违规现象 [24], [25], [26]。我们将漏洞描述与对修复的基于代码的刻画分开标注（第 4 节）。该分类体系涵盖单项违规和有向违规链。结果还表明，仅凭某种后果或宽泛的 CWE 标签，无法唯一确定需要恢复哪项需求。我们确定了影响类别与宽泛修复类别之间的 22 种典型关联关系，随后考察了按簇归类的源代码示例，以解释具体的修复选择（第 5 节）。对于内存边界问题，代码所支持的输入域和查找逻辑决定了应拒绝请求、增加容量，还是修正索引计算。因此，影响会缩小候选方案的范围，而代码上下文则能说明这些方案在何种情况下适用。

第三部分：检索增强的大语言模型修复与相关修复示例。 我们从修复知识库中检索修复示例，引导四种大语言模型采用相同的先规划后编码工作流程（第 6 节）。聚类引导利用补丁簇查找相似的代码变更，而影响引导则从与目标安全影响相关的修复类别中选择示例。我们将这两种方法与零样本、随机方法以及基于经审查机制匹配的预言机进行比较，并在按编辑复杂度分层的 291 个目标上评估修复成功率和计划质量。

我们的评估结果表明，**选择适当的修复示例能够显著提升大语言模型的补丁生成能力**：聚类引导检索和影响引导检索分别将四个模型的平均 pass@3 从零样本下的 37.9% 提升至 49.2% 和 48.5%，而经审查的模式匹配则达到 56.9%。 **对于开放权重模型 Qwen3.8-27B 和 GLM-4.5-Air，提升尤为显著**。与随机选择相比，聚类引导检索使它们的 pass@3 分别提升 42.2% 和 80.3%，而影响引导检索带来的提升分别为 39.2% 和 87.1%。**相关示例还使一些开放权重模型在简单目标上的观测修复率达到或超过 GPT-6 Luna**：聚类引导检索使 Qwen3.8-27B 和 gpt-oss-120b 的 pass@3 分别达到 74.8% 和 75.7%，而 GPT-6 Luna 为 73.9%。在中等目标上，影响引导检索使 gpt-oss-120b 达到 47.7%，接近 GPT-6 Luna 在相同方法下的 48.6%。这些提升表明，相关示例能够提供漏洞-特定的修复知识，而这些模型无法仅从目标代码中可靠地恢复这些知识，从而帮助它们识别被违反的要求以及适用的修复策略。**这一发现凸显了检索增强生成通过利用相关的安全修复知识补充开放权重模型的代码生成能力，从而增强这些模型的潜力**。然而，并非每个示例都有帮助：在示例数量相同的情况下，随机选择使四个生成器中的三个的观测修复率下降，平均 pass@3 降至 36.9%。 **这些结果表明，有必要检索修复策略适用于目标漏洞的修复示例，使生成过程能够利用修复知识库中的相关知识**。对生成计划的分析进一步表明，相关示例能够改善问题识别和修复计划完整性，这有助于解释它们为何能提升补丁生成效果。使用生成器自身的假设修复作为检索查询仍能保留部分提升，这表明即使开发者修复不可用，这些知识也能发挥作用。然而，在更困难的目标上，提升幅度仍会缩小，此时经审查的模式匹配保持着更大的优势，因此有必要进一步研究如何检索和应用合适的修复策略。通过刻画这些知识并考察其在补丁生成中的使用方式，我们的研究旨在推动进一步探索如何将修复模式和安全影响与大语言模型相结合，以实现更有效的漏洞修复。

2. 关键概念

安全影响与安全要求。安全要求 是操作必须满足的条件，以确保程序保持安全：写入必须限制在其对象范围内，引用必须指向有效目标，访问必须具备所需权限。我们根据易受攻击代码所违反的安全要求来刻画安全影响 [20]。CVE 描述可能会同时说明被违反的要求、根本原因（代码中的缺陷）以及后果（攻击者所能获取的内容）。图 2 在我们的运行示例中标出了这三者。我们标注其中明确指出的违规。根本原因可能已经点明了修复方法（例如，缺少长度检查），因此对其进行标注就会泄露修复方案（第 4.1 节）。相比之下，崩溃等后果可能由许多不同的违规导致。

一个影响类别将相关安全需求的违规行为归为一组，例如内存边界或对象生命周期方面的违规。一个漏洞可能涉及多个类别。图 2 中的示例涉及写入超出栈缓冲区边界的问题，属于内存边界类别。

修复模式。安全要求是反复出现的一种代码变更方式，用于满足安全要求。例如，拒绝超出目标容量的复制（图 2），以及在公开输出结构的内容之前对其进行初始化。模式描述了这些变更如何与周围代码协同工作，因此，不同的编辑和实现都可以实现同一种模式。一个补丁可以实现多个模式，而一个模式可能需要跨函数或文件进行协调变更（第 3.1 节）。

安全影响 仅 CVE 描述

根本原因	“未确保 Tag 11 中的密钥签名长度数据包与密钥签名缓冲区大小兼容”
需求违规	“基于栈的缓冲区溢出”
后果	“导致拒绝服务（系统崩溃），或可能获取特权”
影响类别	内存边界（写入超出密钥签名缓冲区的容量）。

修复模式 仅代码

```
    parse_tag_11_数据包(unsigned char *数据,
            unsigned char *contents,
            大小_t 最大_内容_字节,
            大小_t *tag_11_内容_大小,
    ...
+       if (unlikely((*标记_11_内容_大小)
+               > 最大_内容_字节)) {
+           ...
+           rc = -EINVAL;
+           goto out;
+       }
    ...
        memcpy(内容,          &数据[(*数据包_大小)],
               (*tag_11_内容_大小));
```

模式：拒绝复制超出目标缓冲区容量的数据。
当复制长度超出可用容量。

图 2. Linux eCryptfs 栈缓冲区溢出的安全影响和修复模式。 + 表示新增内容，... 表示省略内容。

其他重要概念。补丁样本是包含可用差异以及恢复出的修复前和修复后代码的数据集样本。在我们的修复模式研究中，每个样本都表示为一个聚类点。一个样本可能包含对多个函数或文件的变更，也可能实现多个修复模式。根据表示方式，我们会对单个函数变更进行编码，或从补丁中组装代码片段。这些输入构造选择并不决定修复模式的边界（第3.2节）。簇将表示相似的补丁样本归为一组，我们会检查这些样本，寻找其中共有的修复模式。修复示例展示修复前代码及其对应补丁，以说明修复过程。修复实验使用函数级示例（第 6.1 节）。CVE 描述是影响标注的单位；即使多个补丁针对同一个 CVE，也只计数一次。补丁级分析将每个补丁与其关联 CVE 的影响类别对应起来；如果其中陈述了不同的违规，则保留多个类别。

3. 第一部分：挖掘修复模式

我们的研究首先考察代码表示是否会依据反复出现的修复机制对安全补丁进行分组，并结合人工修复动作研究、聚类评估和基于源代码的模式分析。

数据集。我们从 PatchDB[24]，CVEfixes [25] 和 MegaVul [26]收集 C/C++漏洞补丁。ARVO [23] 为人工研究（第 3.1 节）提供补丁。每个补丁样本记录其代码仓库和修复提交、关联的 CVE 标识符以及可用的描述、差异以及恢复出的修复前后函数代码。我们使用代码仓库和提交标识符合并同一修复的记录，并将等效的向后移植补丁关联为修复族（附录 A）。

聚类研究使用了来自 348 个代码仓库的 3,816 个补丁样本。我们的安全影响研究和漏洞修复研究会说明如何复用该语料库，以及如何区分各自的分析对象和修复目标群体（第 4 节和第 6 节）。

表 1. 常见修复动作及其近似计数和在所示行中的占比。

修复动作	近似计数	占比（%）
添加前置条件检查	210	15.1
添加或删除前置条件检查	200	14.3
更改前置条件检查	185	13.3
移除前置条件检查	185	13.3
更改方法调用	135	9.7
更改前置条件	135	9.7
添加后置条件检查	115	8.2
更改变量赋值	115	8.2
更改后置条件检查	115	8.2

3.1. 从修复动作到修复模式

对修复动作的人工研究。我们首先考察了安全补丁中反复出现哪些代码变更，以及这些变更如何组合。两位作者检查了 300 个 ARVO 补丁差异 [23]，（按修复提交去重），并为变更行标注其执行的修复动作。他们以一项关于 Python 缺陷修复的研究 [17]，中描述的修复动作为起点，例如更改调用参数的数量或取值；当现有标签不适用时，则添加新标签。最终列表包含 20 个修复动作标签，其中包括同一修复动作的不同变体。表 1 总结了频繁出现的标签，并展示了前置条件检查变更的重要性。大多数接受检查的补丁都获得了多个标签，有些补丁则在非连续位置或跨文件组合了变更。这些观察促使我们进一步寻找通过单个修复动作或多个修复动作组合实现的重复修复模式。

从编辑到修复机制。编辑的目的取决于其上下文：添加条件可能用于保护读取范围、指针有效性或对象状态。先前的工作对反复出现的编辑进行描述 [32]，并将 AST 上下文与编辑动作相结合 [14]。我们研究代码变更及其周边代码所建立的修复关系。修复模式指出受影响的数据或操作、所需条件，以及用于建立该条件的变更，例如在读取之前检查剩余输入是否充足。宽泛类别按照主要方法对修复进行分组，而具体模式则涵盖跨类别协同进行的变更。例如，要保持字符串终止状态，可以同时预留一个缓冲区字节、将输入限制在剩余空间内，并确保预留字节为零。

3.2. 补丁表示与聚类

每个补丁样本对应一个聚类点，可能涉及多个函数或文件，并实现多个模式。固定队列包含 3,816 个样本和 7,534 个恢复的修改函数，这些样本来自一个包含 6,173 个补丁的补丁库，且不限制每个样本包含的函数数量。可用代码不一定能重建每项提交编辑（附录 A）。我们比较冻结的 CodeReviewer (CR) [33] 和 Jina [34]嵌入、Diff-ASTCentrality (Diff-AST) 以及 Edit-Script TF-IDF (TF-IDF)，为每个样本生成一个单位长度向量。

代码输入。两种嵌入方法都会保留变更行及其两侧最多三行对齐的上下文行，并标记省略片段。CR 会分别对每个还原出的已修改函数的变更前后视图进行编码，不包含函数标题。Jina 会对带标记的原始补丁片段进行编码，包括函数外的变更，并保留原始标识符以及带编号的文件头和片段头。对于可解析的代码，Jina 会利用现有源代码行，将片段扩展至其包含语句或控制头。当超出输入预算时，会先移除可选上下文，再移除实质性变更。

CR：对函数内的变更进行编码。我们对 CodeReviewer 编码器针对每个变更前后输入生成的词元状态取均值，同时保留其未变更行标记。这种读取方式以及下文的变更组合均为本研究中的选择。令 u_f 和 v_f 分别为函数 f 变更前后视图的 L2 归一化嵌入，令 $\mathcal{N}$ 表示将向量归一化为单位长度。我们构造

$$\begin{aligned} d_f &= \mathcal{N}(v_f - u_f), \\ z_{\mathrm{CR}} &= \mathcal{N}\Big(\tfrac{1}{m}\textstyle\sum_{f=1}^{m} d_f\Big), \end{aligned} \tag{1}$$

其中，m 是保留的函数变更数量。这种组合方式强调相对于局部上下文的变更，并对各函数赋予相同权重。

Jina：编码联合补丁。对于 Jina CodeEmbeddings 1.5B，我们在标记的差异前加上其代码片段指令，并使用最终词元的隐藏状态，对变更及上下文进行联合编码。

较长的输入会被编码为不重叠分块，并通过长度加权平均进行组合。

AST 编辑基线。两个结构基线都使用配对函数 [35] 之间的 GumTree 编辑操作。Diff-AST Centrality 使用节点中心性特征描述编辑图，采用 TreeCen 的结构摘要方法 [36]，，并使用固定的函数级缩放器对其功能进行标准化。Edit-Script TF-IDF 将固定的 TF-IDF 词表应用于编辑操作词元和 AST 上下文词元。对于每个基线，经过归一化的函数向量会在一个补丁内取平均。

降维与聚类。所有表示均采用平均链接层次聚类，划分为 64 组，并使用欧几里得工作空间距离。CR 使用 UMAP [37]将向量降至 128 维，而 Jina 和两个结构基线则使用 PCA 降至 32 维（表2）。聚类直接使用降维后的向量。

表 2。补丁聚类流程。 K：聚类数量。

表示	降维	聚类	K
代码嵌入			
CR	UMAP，128D	平均连接法	64
Jina	PCA，32D	平均连接法	64
AST 编辑特征			
Diff-AST	PCA，32D	平均连接法	64
TF-IDF	PCA，32D	平均连接法	64

表3。3,816个补丁的簇质量。所有语义得分均为百分比。

方法	轮廓系数 ↑	DBI ↓	C ↑	M ↓	T_3 ↑	$P_\times$ ↑
代码嵌入						
CR 0.307		0.884	13.3	7.1	80.5	12.2
Jina	0.055	2.084	20.0	44.1	84.7	14.0
AST 编辑特征						
Diff-AST 0.124		1.640	16.0	11.9	77.8	8.4
TF-IDF	0.282	1.361	15.0	13.5	78.6	10.3

3.3. 评估补丁聚类

表3报告几何质量和语义质量。轮廓系数 [38] 衡量簇内凝聚度相对于相邻簇的程度（↑），而 Davies–Bouldin 指数（DBI） [39]衡量离散度相对于簇中心间距的程度（↓）。两者均使用工作空间（附录 A）。

广泛修复模式类别。以往研究 [14], [32], [40]和我们的人工研究共同确定了十二种修复类别（表 4）。检查和边界条件还用于区分受保护约束。这些标签用于评估和源代码检查，而不用于编码器训练或聚类。

标注协议。 我们使用大语言模型为补丁分配宽泛的修复模式类别，并对覆盖全部十二个类别的 100 个补丁进行人工审核，以评估类别适用性和标注一致性。代码变更及上下文优先于辅助性的修复相关提交消息或 CVE 段落。影响标签不予提供。引用为 3,748 个补丁分配了主类别。类别评分不包括 68 个未解决案例。

基于修复模式的聚类评估。T_3 衡量每个簇中占比最大的三个类别所覆盖的已标注补丁比例。 $P_\times$ 衡量簇内跨源组配对的主标签一致性，使用 22 个用于区分受保护约束的标签。伙伴捕获率 C和干扰项混合率 M 分别衡量与测试补丁归为一组的已知同模式伙伴和经审查的同源组干扰项所占比例，并对有引用的补丁进行等权平均。 T_3、 $P_\times$ 和 C越高、M越低，效果越好。源组规则、引用数量和公式见附录 A。

两种嵌入方法在 T_3 和 $P_\times$ 上均优于 AST 编辑基线。CR 还提升了轮廓系数、DBI 和干扰项混合率，而 Jina 捕获了更多同模式伙伴。我们保留这两种方法用于源代码分析：CR 分组更具选择性，而 Jina 收集了更多已知实现。

表 4. 广泛修复模式类别及其占比，以全部 3,816 个补丁的百分比表示。

类别	典型变更	Shar
检查并拒绝	在所需条件失败时拒绝操作。	53.9%
边界处理	限制长度、计数、数值或工作负载。	5.2%
提供容量	为……分配足够的空间。	2.8%
更改数值表示	所需数据。校正值的类型、位宽或有符号性。	2.3%
校正大小或位置	校正长度、偏移量或索引计算。	4.7%
初始化或清除状态	在使用内存或状态之前，或在输出之前定义它们	7.8%
释放与清理	输出。修正缺失或重复的资源释放。	6.1%
保护对象生命周期	在对象被使用。	7.9%
传播故障	返回此前被丢失。	0.6%
协调并发访问	同步访问共享状态。	3.4%
分离数据与指令	将输入作为命令中的数据，查询或格式。	0.8%
限制权限或功能	移除不必要的权限或功能。	2.6%
未确定	无法确定修复类别。	1.8%

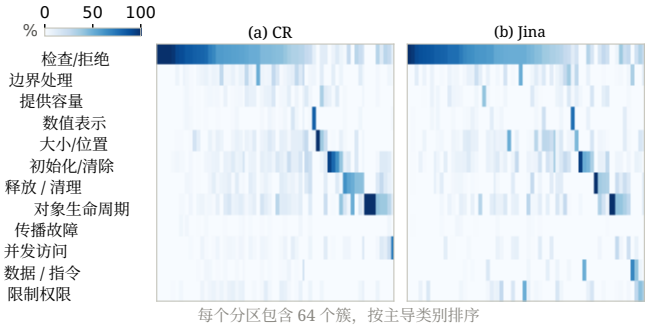


图3. CR 和 Jina 簇中的大类分布。每列代表一个按主导类别排序的簇，颜色越深表示簇内占比越大。

3.4. 补丁簇中的修复模式

为了识别修复模式，我们检查 CR 和 Jina 簇中的代码变更及其周围上下文。我们利用图 3 中的类别分布，考察由初始化或检查主导的簇，以及包含多个修复类别的簇。根据这番检查，我们提出候选模式，随后使用大语言模型智能体根据源代码变更完善其定义，并检查每个簇成员是否提供支持。

发现1：补丁簇反映了不同代码形式之间共有的修复机制。源代码比较揭示了 CR 和 Jina 簇中通过不同代码形式实现的共同防护措施（图4）。

（*1*）在导出输出结构之前，先将其初始化。补丁会在字段赋值前清除整个输出结构-

表5. 补丁簇中识别出的其他修复模式。

修复模式	核心修复关系
检查指针链	验证到达一个所需的每个指针 a 嵌套字段所需的每个指针在读取之前均为非空。
验证索引和查找结果	拒绝越界索引或失败的查找在访问集合之前读取结果。
将访问限制在剩余的空间	使用以下方式检查或限制读取或复制长度其起始偏移量之后可用的字节数。
将长度限制在存储容量	拒绝超大长度，或将其限制为使用前相关的固定容量。
检查乘法内存分配前	检查元素数量乘以元素大小是否溢出，然后再分配或初始化内存。
读取前检查长度	在处理固定结构或编码字段之前，确保输入字节数足够。
读取字段	结构或编码字段之前对其进行解读。
发生后续失败	释放先前的内存分配，并对部分已构造的对象，若后续操作失败。

(a) 导出前初始化输出：Linux cxgb3。

```
 struct ch_reg e数据;

+内存设置(&edata,  0,  sizeof(struct ch_reg));
+
  edata.cmd  =  CHELSIO_获取_QSET_NUM;
  edata.val  =  pi->nqsets;
  if (copy_to_user(useraddr, &edata, sizeof(edata)))
```

(b) 有界递归解码：Wireshark。

```
+深度++;
+if (深度  >  MP4_BOX_MAX_REC_LVL) {
   . . .
+return -1;
+}
   . . .
-ret  =  dissect_mp4_box(box_类型,
-       tvb, 偏移量, 数据包信息,  box_tree);
+ret  =  dissect_mp4_box(box_type, 深度,
+tvb, 偏移量, 数据包信息,  box_tree);
```

（c）保留字符串终止符：ssdp-responder。

```
-char 缓冲区 [MAX_PKT_大小];
+char 缓冲区 [MAX_PKT_大小  +  1];
  内存设置(缓冲区,  0, sizeof(buf));
-len  =  接收数据(sd  ,  buf,  sizeof(缓冲区),
- MSG_DONTWAIT,&sa, &salen);
+len  =接收数据(sd, 缓冲区, sizeof(缓冲区)        -  1,
+       MSG_DONTWAIT,  &sa,  &salen);
  if (len  >  0) {
-缓冲区 [长度]  =  0;
-
```

图4. 三种代表性修复模式的源代码摘录。省略的行以……标记。

例如 Linux cxgb3（图4a）中的相关操作，或者初始化保留字段和填充字段，以防未定义字节暴露。

(2) 传递并限制递归深度或开销。Wireshark 会在解码调用间传递深度（图4b），而 Libexif 会累加每个层级条目的开销。当超出限制时，两者都会停止处理。

(3) 限制字符串输入并保留终止符。ssdp- responder 会在有界输入之后，于零初始化的缓冲区中预留一个字节（图4c），而 GPAC 会显式将预留位置清零。两者都会保留字符串终止。

表5总结了在这些簇中发现的其他常见模式。

簇中体现的修复机制。一个 Jina

簇组归纳了不同的资源生命周期修复：TigerVNC 删除自有的子对象，GPAC 清除已释放缓冲区记录的大小，而 Linux 故障循环则结合了对象销毁与指针清除。该簇涵盖了相关的防护问题，但并不意味着它们遵循同一种模式。

具体模式也会跨越宽泛类别：ROS、GPAC 和 ssdp-responder 分别通过边界限制、初始化和容量提供来维护字符串终止。反之，一个 FreeRDP 容量提供补丁也会执行故障清理。宽泛类别概括补丁的主要方法，而具体模式则揭示共享的保护机制和协同编辑。

4. 第一部分：安全影响分类体系

我们的安全影响研究基于既有研究和规范提炼出12个类别，这一过程独立于补丁簇和修复类别，并为第5节的关联分析标注 CVE 描述。

4.1. 分类单位与证据

我们的安全影响研究复用了修复模式研究（第3节）中的3,816个补丁语料库。我们为每个不同的关联 CVE 描述仅标注一次，并将其 CVE 的类别并集分配给每个补丁。关联分析保留同时具有修复类别和影响标签的补丁。

证据分离。修复类别使用代码变更及上下文，并辅以修复说明（第3.3节）。相反，影响标注使用经过清理的 CVE 描述，不使用补丁、提交消息或 CWE标签。我们移除了补救措施、来源和严重性陈述，并且两种过程均不使用对方的标签（附录 B）。

每个明确陈述的违规都记录为一条规则：一个操作、其对象、被违反的要求以及支持性引文（图 2）。不同的违规（包括链中的连续违规）分别归入不同类别。访问方向和资源类型仍作为属性保留。因此，多线程服务器中的越界访问属于内存边界违规。只有在有协调失败证据时，才会添加并发标签。由于 CVE描述可能省略详细信息 [41]，我们将证据不足与分类体系之外的明确陈述的违规区分开来。

4.2. 分类体系构建

推导分类体系。 我们根据被违反的安全需求定义安全影响，并将其与代码缺陷和攻击后果区分开来 [20]。以往对安全规则违反的分类为我们整理内存、对象状态、引用和权限要求提供了依据 [20]。其他来源则提供了协调 [27], 工作预算 [28], 输入识别与解释 [29], 协议转换 [42],证书验证 [30], 随机数唯一性 [31], 及侧信道要求 [43]。CWE定义还提供了受限解析和敏感数据擦除等需求 [44]。

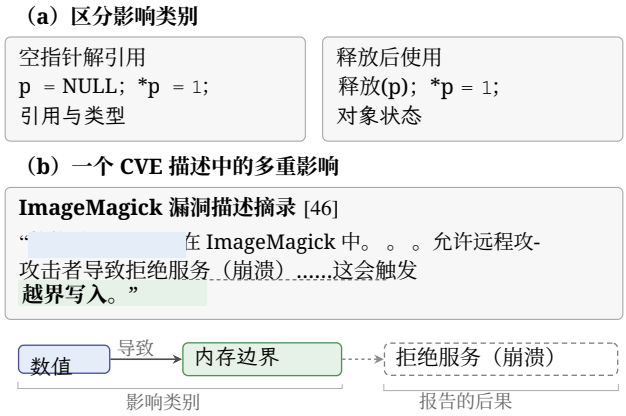


图5. 影响类别边界与多个标签。

我们为每项需求明确其操作、对象、条件和边界示例，合并同义词，并将36项需求归为12类（表6）。分组方式反映了描述的具体程度和预期样本量：并发类合并了原子性、顺序和同步，因为竞态报告可能无法区分这些问题。在关联分析之前，分类体系即已确定。

边界与多个标签。空指针解引用违反引用与类型要求，而释放后使用违反对象状态要求（图5a）。遵循 NIST 漏洞框架的链式视角 [45], , 不同的违规保留各自有证据支持的标签，例如图5b 中数值溢出导致内存边界违规的情况。对同一事件的不同描述不会产生额外的违规。

4.3. 标注与验证

流程。 首先，我们提取被违反的要求、支持这些要求的文本，以及相邻违规之间明确陈述的任何因果联系。其次，我们将每项要求与固定的类别定义进行比较，仅在其与单一类别匹配时才分配标签。我们使用 GPT-6 Luna 对每条清理后的 CVE 描述应用这一两阶段流程。当两个存在因果联系的违规映射到不同类别时，我们计为一条有向链接。

案例 1。 ImageMagick 的描述（图 5b）获得两个标签：*数值* 用于整数溢出，*内存边界* 用于由此产生的越界写入。通过崩溃导致的拒绝服务是报告的后果。

标注结果。 在 3,888 条不同且可用的 CVE 描述中，有 1,002 条（25.8%）未获得任何影响类别。大多数描述没有明确说明被违反的要求：例如，仅报告发生崩溃，并不能解释究竟出了什么问题。其他违规不属于该分类体系，或无法明确映射。随后，我们为每个补丁分配其关联描述中出现的类别。这样一来，在 3,816 个补丁中，有 2,838 个（74.4%）至少获得一个类别。

标签验证。 作者随机审核了100条抽样描述，建立了一个引用基准；与之相比，自动标签实现了94.0%的微平均精确率、86.7%的微平均召回率和0.902的微平均 F1 值（附录B）。

表6. 3,816个补丁中的安全影响类别及其占比。

影响类别	安全要求	示例性违规	补丁， n (%)
内存边界	内存操作遵守对象边界、对齐以及重叠规则。	将 100 字节复制到 64 字节的缓冲区中。	975（25.6）
对象状态	使用和释放遵守对象的当前状态。	使用已释放的缓冲区。	564（14.8）
引用与类型	引用具有有效的目标，并且使用时采用兼容的类型。	解引用失败操作返回的空指针查找。	387 (10.1)
数值	算术运算和转换满足范围、精度和定义域需求。	分配大小乘法发生回绕。	291 (7.6)
资源持有	资源会在适当时释放，并保持在容量限制内。	错误返回会导致文件描述符保持打开状态。	235 (6.2)
进展	工作持续推进、保持在预算范围内，并按要求终止。	解析器循环运行，却没有推进其输入位置。	160（4.2）
并发	线程、回调和信号处理程序按需协调。	未同步的计数器更新会丢失一次递增。	130（3.4）
输入接受	接受的输入和步骤遵循格式或协议，并保持一致的解释。	协议接受了一条跳过了必需握手步骤的消息。	34（0.9）
不可信解析	外部输入仅被视为允许的数据并解析为允许的目标。	提供的路径通过以下方式逃逸出其允许的目录 `../`.	67 (1.8)
信任决策	信任与访问决策满足身份、证明、权限和新鲜性需求。	接受了主机名错误的证书。	152 (4.0)
密码学材料	密码学构造、参数和材料满足其使用需求。	GCM 加密重复使用随机数，且使用相同的 key.	14（0.4）
敏感数据	敏感数据在输出中仍受到保护，可观测行为、存储和处置。	已释放的缓冲区在可复用的内存。	44（1.2）

4.4. 安全影响与违规链的分布

表6按每个补丁所分配的类别分别计数。在全部3,816个补丁中，内存边界是最常见的类别（975个，占25.6%），其次是对象状态（564个，占14.8%）和引用与类型（387个，占10.1%）。

发现 2：当一条描述报告多个影响类别时，一个违规往往会导致另一个违规。在被分配了多个类别的 198 条 CVE 描述中，有 134 条（67.7%）根据第 4.3 节的规则包含至少一个有向链接。最常见的链接是从数值违规指向内存边界违规，例如整数溢出导致越界写入（图 5b）。其次常见的是从并发指向对象状态，例如竞争条件导致释放后使用（附录B）。

ImageMagick 示例还说明了仅按后果对漏洞进行分组会遗漏什么：拒绝服务既无法识别溢出，也无法识别越界写入。在整个语料库中，提及拒绝服务的描述合计涵盖全部 12 个影响类别。唯一 CWE 标签为 CWE-20（不当输入验证）的描述合计涵盖 11 个类别。识别出遭到违反的具体需求，为我们提出“哪些代码变更能够恢复该需求”这一问题提供了依据（第 5 节）。

5. 第二部分：将修复模式与安全影响关联起来

我们分析了数千个安全补丁，以确定相较于不具备某种安全影响的补丁，哪些修复类别在具有该影响的补丁中出现得更频繁。这些关联关系指向了针对某项违反的安全要求的候选修复模式。随后，我们检查源代码，以解释为什么解决相同影响的补丁可能需要不同的修复。

5.1. 关联分析

分析采用了 2,791/3,816 个补丁，这些补丁均具有一个主要修复类别和至少一个独立分配的影响类别（第 3.3 节和第 4.3 节）。我们保留所有关联 CVE 的影响类别，并在十二个修复类别中进一步根据受保护条件区分检查并拒绝 。

对于类别 p 和影响类别 i，我们比较该影响类别内外的修复发生率：

$$\Delta_{p,i} = P(p \mid i) - P(p \mid \neg i).$$

每个补丁在每次比较中计数一次。支持度和代码仓库覆盖情况共纳入 198 个配对。仓库分块自助法区间 [47], 分块置换检验和错误发现校正 [48]识别出符合区间要求及附录 C 中分层检查要求的正向主链接。

5.2. 关联背后的修复选择

发现 3：安全影响指导修复选择，而代码上下文决定适用的模式。图 6 展示了 22 条主链接，这些链接均通过全局错误发现校正，同时也展示了其他共现情况。指针有效性检查出现在 55.0% 的引用与类型补丁中，而在不属于该类别的补丁中仅占 2.9%。释放与清理 出现在 67.8% 的持有资源的补丁中，而在其他补丁中仅占 2.2%。

按代码变更对补丁进行分组，还能将整个项目中具有相同安全影响的补丁归到一起

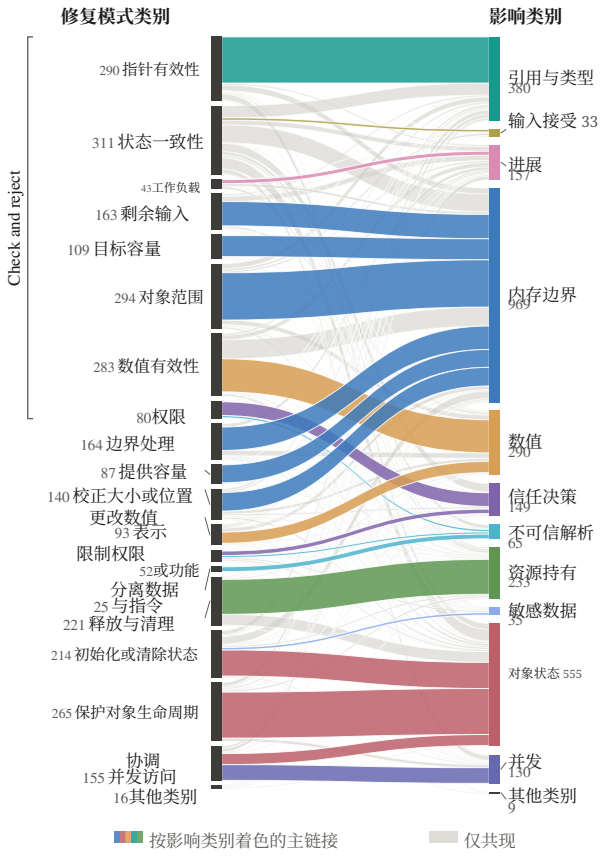


图 6. 修复类别与安全影响。带宽表示共现次数，彩色带表示主要关联关系。

(a) 拒绝不受支持的索引：Linux sock diag。

```
+if (req->sdiag_family >= AF_MAX) + return -EINVAL; hndl =
sock_diag_lock_handler(req->sdiag_family); (b) 通配 一个支持
的索引：tpm2-tss。 -} layer_handler (TPM2_ERROR_TSS2_RC_
LAYER_COUNT); {-} layer_handler(TPM2_ERROR_TSS2_RC
for (i = 0x0F; i > -1; i--) - if (wd_times [i] >t)+ if (wd_times [i
] >= t) break; wd_margin = i; 图 7。三种内存边界修复：拒绝请
```

谱系。在来自不同谱系的同簇代码对中，Jina 中有 41.4% 至少共享一个影响类别，而在每个代码仓库内打乱影响标签后，这一比例平均为 28.3%。

内存边界说明了为什么上下文决定修复方式（图7）。Linux sock_diag 会拒绝不受支持的族编号，而 tpm2-tss 则将其表从255项扩大到256项，以支持层0xFF。Linux ib700 则将 > 更改为 >=，使其表查找匹配有效的30秒上限，而不是生成索引 −1。这些修复分别用于拒绝无效请求、容纳有效请求或纠正其查找方式。

表7. 具有不同修复作用的相关边界编辑。条件为缩写形式。

修复后的条件	相等时的操作及含义
FreeRDP：索引 >= number	返回错误：拒绝超出 最后一个条目的索引。
radare2：sym_num >= nsymtab	跳过条目：排除超出 最后一个元素的索引。
ib700：wd_times [i] >= t	停止搜索：选择 与请求的超时时间匹配的条目。

看门狗的 CR簇包含来自七个代码仓库的11个补丁，其中所有十个带标签的成员都属于内存边界。相同的 > 到 >=编辑在 FreeRDP 和radare2中拒绝越过末端一位的索引，但在看门狗中选择有效的边界条目（表7）。因此，相同的编辑实现了不同的修复作用。

CR 簇 52 中的两个 Linux 补丁属于同一类释放与清理，且进行了相反的修改（图 8）。在 mwifiex_tm_cmd 中，三个错误退出路径都会导致已分配的缓冲区未被释放（资源持有），因此补丁在每次返回前释放该缓冲区。在 path_openat 中，do_tmpfile() 已经执行了清理，因此补丁跳过了第二次释放，以避免报告中涉及的释放后使用问题（对象状态）。

(a) 在 mwifiex_tm_cmd 中补上遗漏的释放操作（资源持有）。

```
 hostcmd =kzalloc(sizeof(*hostcmd),          GFP_KERNEL);
 . . .
 if (mwifiex_send_cmd(priv,      0, 0, 0, hostcmd,      true)) {
         . . .
+        kfree(hostcmd);
         return -EFAULT;
 }
```

(b) 路径_openat 中跳过重复清理（对象状态）。

```
 错误 = do_临时文件(dfd, 路径名, nd, 标志, 操作,
                    文件, &opened);
-goto out;
+goto out2;
 . . .
 out:
 路径_cleanup(nd);
+out2:
```

图 8。CR 簇 52 中的两个补丁添加缺失的释放操作 (a)，或跳过重复的释放操作 (b)。

对于递归限制而言，受保护的操作说明了该模式所应对的影响。open62541 在索引 ctx->c ommaNeeded 之前限制递归深度，以保护内存边界；而 Linux Netlink 限制深度，以通过策略反向引用阻止无限递归，从而保障进展。影响范围缩小了修复选择，簇用于定位相关示例，而源代码上下文决定了适用性。

6. 第三部分：检索增强修复

我们研究修复示例的选择如何影响大语言模型识别不安全操作并修复漏洞的能力。对于检索增强生成，我们使用补丁簇和影响–模式关联关系，从修复知识库中检索相关示例。我们比较示例选择方法，考察由此产生的计划和

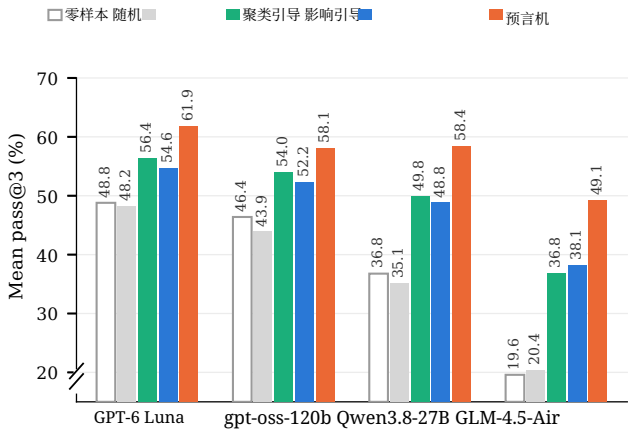


图9。不同难度的 291 个目标上的 pass@3。纵轴从 15% 开始。

修复结果，并测试查询质量、项目谱系和代码表示所发挥的作用。

6.1. 实验设计

目标与示例。 我们的漏洞修复研究评估了一个单独留出的集合，其中包含来自 62 个项目谱系、75 个代码仓库的 291 个 C/C++ repair 目标。测试补丁及其修复族中的所有其他成员均被排除在聚类之外。我们根据开发者补丁复杂度，将目标分为 111 个简单、107 个中等和 73 个困难目标：简单修复只需进行简短的局部更改，而困难修复则涉及多个函数、更大范围或分散的编辑，或多个新引入的标识符。附录 D 详细介绍了基准构建和分层规则。在检索过程中，我们复用了修复模式研究中的补丁簇，以及安全影响研究中的关联关系，并将知识库池扩展为 5,642 个安全补丁。对于超出 3,816 个聚类队列的补丁，我们将其分配到现有聚类中，而不是对整个池重新进行聚类。我们从其他修复族中抽取示例，其中也包括目标所属项目谱系中的修复族。

检索方法和比较条件。在这些实验中，一个修复示例包含某个函数的修复前代码和差异。每个少样本提示包含三个修复示例。零样本 不提供任何示例，而 随机 则从符合条件的修复族中随机抽取示例。聚类引导 根据示例与查询修复的相似度来选择示例。它会先搜索相应的补丁簇，再根据代码变更之间的相似度对示例进行排序。影响引导 利用影响–模式关联关系，在修复知识库中找出对于目标的安全影响而言最常见的三个修复类别。它从这些类别中检索修复示例，优先在查询修复所属的簇中查找，并根据代码变更相似度对示例进行排序。在主要比较中，这两种方法都使用 CR 簇和开发者修复作为查询。第 6.3 节还评估了模型生成的查询。预言机从广泛的候选集中挑选经过仔细筛选的修复示例。它优先选择实现目标

表 8. 按编辑复杂度层划分的 pass@3（%），以及同一层内与随机的差异（ΔR，百分点）。粗体标出每个生成器在各层中的最高比率，单元格底色按 pass@3 着色。

生成器	方法	简单（111）		中等（107）		困难（73）	
		pass@3	ΔR	pass@3	ΔR	pass@3	ΔR
GPT-6 Luna	零样本	65.8	+2.4	39.3	−2.8	37.0	+2.7
	随机	63.4	–	42.1	–	34.2	–
	聚类引导	73.9	+10.5	53.3	+11.2	34.2	0.0
	影响引导	72.1	+8.7	48.6	+6.5	37.0	+2.7
	预言机	**76.6**	+13.2	**57.9**	+15.9	**45.2**	+11.0
gpt-oss-120b	零样本	57.7	0.0	43.9	+4.7	32.9	+3.2
	随机	57.7	–	39.3	–	29.7	–
	聚类引导	**75.7**	+18.0	47.7	+8.4	30.1	+0.5
	影响引导	69.4	+11.7	47.7	+8.4	32.9	+3.2
	预言机	72.1	+14.4	**59.8**	+20.6	**34.2**	+4.6
Qwen3.8-27B	零样本	49.5	0.0	34.6	+6.9	20.5	−3.2
	随机	49.5	–	27.7	–	23.7	–
	聚类引导	**74.8**	+25.2	38.3	+10.6	28.8	+5.0
	影响引导	73.0	+23.4	38.3	+10.6	27.4	+3.7
	预言机	**74.8**	+25.2	**56.1**	+28.3	**37.0**	+13.2
GLM-4.5-Air	零样本	27.9	+1.8	17.8	+0.6	9.6	−6.8
	随机	26.1	–	17.1	–	16.4	–
	聚类引导	52.3	+26.1	28.0	+10.9	26.0	+9.6
	影响引导	47.7	+21.6	36.4	+19.3	26.0	+9.6
	预言机	**63.1**	+36.9	**48.6**	+31.5	**28.8**	+12.3
四者平均值	零样本	50.2	+1.1	33.9	+2.3	25.0	−1.0
	随机	49.2	–	31.5	–	26.0	–
	聚类引导	69.1	+20.0	41.8	+10.3	29.8	+3.8
	影响引导	65.5	+16.4	42.8	+11.2	30.8	+4.8
	预言机	**71.6**	+22.4	**55.6**	+24.1	**36.3**	+10.3

修复机制的示例，并用共享必要修复要素的相关修复示例填满所有剩余名额。

生成与评估。我们评估了四个大语言模型。GPT- 6 Luna [49]是一款商业前沿模型，gpt-oss- 120b [50]、Qwen3.8-27B 和 GLM-4.5-Air [51]则是不同大小的开放权重模型。每个模型都会先生成修复计划，再在同一对话中生成修正后的函数。修复提示提供易受攻击的函数和选定示例，但不包含目标漏洞描述，也不提供开发者修复中的位置提示。如果候选方案能够阻断与开发者修复相同的危险路径，并对控制流产生等效影响，我们的评估协议便会判定其通过。对于 GPT-6 Luna，我们在隐藏生成器和检索方法的情况下，将此标准应用于源代码变更。每组示例生成三个候选方案；只要其中任一候选方案通过，该组便计为成功（pass@3）。

6.2. 修复有效性

发现 4：聚类引导和影响引导检索甚至能提升前沿模型的修复效果，且基线修复率较低的模型获益最大。对于全部四种生成器，这两种方法的 pass@3 均高于随机方法（图 9），平均分别提升 12.3 和 11.6 个百分点。对于 GPT-6 Luna，提升幅度分别为 8.1 和 6.4 个百分点。对于 Qwen3.8-27B，pass@3 从 35.1% 提升至

随机方法下的 49.8% 和 48.8%；对于 GLM-4.5-Air，则从 20.4% 提升至 36.8% 和 38.1%。这些提升源于改变生成器接收的三个示例，而无需对其进行微调或增加生成步骤。

表 8 展示了这一优势如何随编辑复杂度而变化。在四个模型中，随机方法在简单目标上的表现为 49.2%，在中等目标上为 31.5%，在困难目标上为 26.0%。在每个难度级别上，聚类引导、影响引导和预言机的四模型均值都高于随机方法。聚类引导和影响引导带来的提升从简单目标上的 20.0 和 16.4 个百分点，降至中等目标上的 10.3 和 11.2 个百分点，以及困难目标上的 3.8 和 4.8 个百分点。预言机的提升在中等目标上最高（24.1 个百分点），在困难目标上则达到 10.3 个百分点。

相关示例帮助一些开放权重模型的修复率接近前沿模型。在简单目标上，聚类引导检索使 Qwen3.8-27B 的 pass@3 达到 74.8%，使 gpt-oss-120b 达到 75.7%；同样采用该方法时，GPT-6 Luna 的表现为 73.9%。在中等目标上，影响引导检索使 gpt-oss-120b 达到 47.7%，与 GPT-6 Luna 的 48.6% 相差仅 0.9 个百分点。这些观测到的相似表现表明，相关修复知识可以帮助开放权重模型在特定难度级别缩小性能差距，不过 GLM 仍落后于 GPT-6 Luna，且在困难目标上的差距仍然较大。

仅添加示例并未比零样本带来整体提升，而随机方法在四种生成器中的三种上观测到的修复率更低。因此，只有当示例内容能够引导目标修复时，修复示例才有用。无关示例可能无法解决相关缺陷，或将修改引向其他问题。我们的聚类和影响—模式映射支持从修复知识库中检索相关示例：簇对示例池进行组织，而影响关联则优先选出与目标安全要求相关的修复类别。检索到的示例展示了如何通过检查、初始化或资源管理来满足该要求，并提供生成器可根据目标上下文进行调整的实现。预言机为经过仔细筛选的示例提供了乐观的参照，达到 56.9%。

为了解这些提升，我们将每次尝试的修复计划与其生成代码分开评估（表 9）。问题识别检查计划是否识别出不安全操作和缺失的保护，而完整计划检查计划是否包含所有必需的修复步骤。与随机方法相比，聚类引导方法将这两项得分分别从 33.4% 提高到 47.0%，从 21.6% 提高到 31.8%。影响引导方法分别达到 45.9% 和 31.2%，而预言机分别达到 57.0% 和 39.3%。两种引导方法在问题识别方面的提升都高于计划完整性，这表明相关示例有助于模型识别需要修复的问题。要规划完整的修复，可能仍需了解调用方或项目特定行为，而所提供的函数并未包含这些信息。

我们进一步审查了 60 个被计为成功的候选项，并判定其中 91.7%（55/60）能够阻断目标危险路径（附录 D）。接下来，我们将说明检索到的修复示例如何为修复计划提供信息，并引导生成补丁。

表 9. 每次尝试的计划质量（%），按目标和生成器取平均。

方法	已识别的问题	完整计划
零样本	33.2	22.5
随机	33.4	21.6
聚类引导	47.0	31.8
影响引导	45.9	31.2
预言机	**57.0**	**39.3**

案例 2。一种通用的初始化模式可以迁移到不同的输出接口。在 Linux EQL 中，eql_g_master_cfg 填充栈对象的两个字段，并将整个对象复制到用户空间，从而暴露其未初始化的 16 字节名称字段。在一次记录在案的随机试验中，GLM 添加了目标指针检查，但对象仍未初始化。聚类引导检索到了整对象初始化的示例，其中包括通过套接字发送 LibVNCServer 消息的示例（图 10a）。GLM 的计划明确地将这些示例与未初始化数据暴露联系起来，并要求在赋值各字段前清零整个对象。生成的 memset 实现了开发者的修复；排除同谱系示例后，GLM 也以相同方式修复目标。
案例 3。影响关联将示例选择引向所需的内存生命周期修复。在 Wireshark 的 PCP 解析器中，通过 g_malloc 分配的会话状态未在文件的内存管理作用域内释放。使用随机或聚类引导的示例时，Qwen3.8 转而添加数据包长度检查，却未改动这项分配。目标的资源持有影响更倾向于清理相关类别。因此，影响引导从相邻簇中检索到一个 TN3270 修复示例，该示例将地址复制所需的分配置于 wmem_file_scope 下（图 10b）。其上下文也在该作用域中分配会话状态。Qwen3.8 在计划中引用了这一模式，并生成相应的作用域管理分配，与开发者的修复一致。

6.3. 消融研究

我们考察查询质量、项目谱系和代码表示如何影响示例引导的修复（图 11）。附录 D 报告了四个模型之间的配对差异。
对查询修复的敏感性。我们要求每个模型生成一个初步修复，并将其用作检索查询。在四模型均值上，两种检索方法相较于零样本和随机仍保持优势。在四个生成器中，将查询切换为其他查询时，影响引导的 pass@3 降幅均小于聚类引导（图 11a）。更改查询可能会改变起始簇以及示例的排序。对于影响引导，即使初步修复指向补丁集合中的不同部分，目标的安全影响仍能识别出相同的候选修复类别。
共享项目谱系的影响。我们比较了检索是否包含来自目标项目谱系（包括相关代码仓库）的示例。纳入这些示例后，聚类引导的平均 pass@3 提高了 3.5 个百分点，影响引导提高了 3.0 个百分点，预言机提高了 3.4 个百分点（图 11b）。随机

(a) 在导出前初始化整个对象。

检索到的修复示例 mple：LibVNCServer

```
 rfbServerCutTextMsg sct;
+内存设置((char *)&sct,      0,  sizeof(sct));
 . . .
 sct.type  =  rfbServerCutText;
 sct.长度  =  Swap32IfLE(len);
 . . .
 if (rfbWriteExact(cl,          (char *)&sct,
                          sz_rfbServerCutTextMsg)  <  0) {
```

生成的目标修复：Linux EQL

```
 master_config_t mc;
+内存设置(&mc,      0,  大小(mc));
 ... mc.max_slaves  =EQL->max_slaves; mc.min_
 slaves  =EQL->min_slaves; if (copy_to_user(mcp, &
 mc, sizeof(master_config_t))) return -EFAULT;
```

（b）将内存分配与其清理作用域关联起来。

检索到的修复示例 mple：Wireshark TN3270

```
 tn3270_信息  =wmem_新建(wmem_文件_作用域(),
                                tn3270_conv_info_t);
 . . .
-copy_address(...);
+复制_地址_wmem(wmem_文件_作用域(),        ...);
 . . .
 conversation_add_proto_数据(...);
```

生成的目标修复：Wireshark PCP

```
-pcp_conv_info  =  (pcp_conv_info_t*)g   _malloc(...);
+pcp_conv_info  =  wmem_new(wmem_文件_scope(),
+                                pcp_conv_info_t);
 conversation_add_proto_数据(...);
```

图10. 检索到的修复示例与生成的修复：（a）GLM 初始化一个导出对象；（b）Qwen3.8 将内存分配绑定到其清理作用域。省略号表示省略的代码。

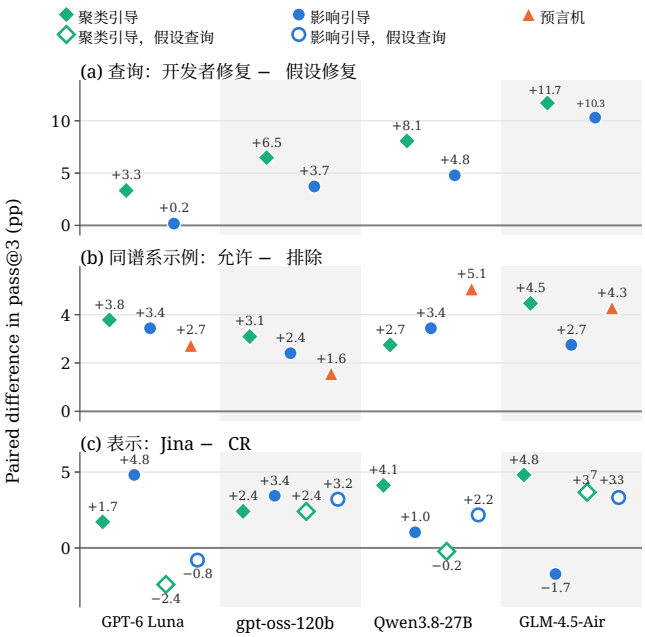


图11. 查询质量、同谱系示例可用性和补丁表示对 pass@3 的影响。各面板使用不同的刻度。

仅变化 −0.3 个百分点。排除同谱系示例后，使用开发者查询时，影响引导仍比随机高出 +8.3 个百分点；使用假设查询时则高出 +4.4 个百分点（表15）。共享谱系提供了有用的上下文，但保留下来的增益表明，修复示例也能为跨谱系修复提供指导。

更改表示。 将 CR 替换为第3.2节中的 Jina 表示和聚类流程后，在开发者查询和假设查询两种情况下，对于四模型均值，两种引导方法相较于零样本和随机方法的优势仍得以保留（图11c）。在 Jina 表示下，影响先验也仍然有用：在假设查询下，为聚类引导方法增加了 +2.8 个百分点。两种表示的一致收益表明，可以通过修复模式及其安全影响关联来组织安全补丁，从而将其关联关系作为从知识库中检索相关修复示例、开展检索增强生成的基础。

7. 讨论

我们的发现表明，安全需求可以作为组织修复知识并将其应用于新漏洞的共同基础。

安全影响引导修复选择。通过被违反的要求来表达安全影响，可以为修复设定具体目标。影响—模式图将这一目标与候选修复方法联系起来，而目标代码上下文则决定这些方法是否适用。因此，安全影响引导着修复生成中的一项关键决策：哪些修复知识与目标相关。

修复模式解释了聚类捕捉到的内容。要组织这些知识，需要采用能够体现修复原理的补丁相似性。簇中识别出的模式通过其所保护的操作和所恢复的安全要求，将不同的代码形式联系起来。通过明确共同的修复机制，这些模式解释了补丁相似性如何支持搜索相关修复示例。

相关示例有助于模型诊断缺陷。即使初步修复使检索转向不同的簇，目标的安全影响仍会将搜索范围缩小到相同的候选修复类别。这有助于解释为什么在所有四个模型中，与聚类引导检索相比，从开发者修复切换为使用变更位置提示生成的初步修复，对影响引导检索的 pass@3 降低幅度较小（图 11a）。与随机方法相比，两种引导式方法在识别不安全操作及其缺失的保护方面，带来的提升均大于其对计划完整性的提升（表 9）。观测到的修复效果提升体现了这样一个知识库的实际价值：它围绕修复模式和安全影响进行组织，以支持检索增强补丁生成。

未来方向。能够更精确地捕捉修复机制的表示，可以在不同代码形式和项目中保留补丁、其保护的操作及其必要上下文之间的关系。与预言机之间的差距

促使我们依据修复示例对目标代码中缺失的防护措施的阐释效果进行排序，并结合互补示例来协调修复。未来的工作可以测试，在后训练期间将易受攻击代码与此类示例配对，是否有助于模型识别其他地方何时需要相同的防护措施。这些方向共同指向一个更宏大的目标：建立一套修复工作流，使安全需求贯穿诊断、相关修复示例的检索、生成和验证。应用并检查候选修复所获得的反馈，可以改进诊断并指导后续从修复知识库中进行检索，从而将影响–模式关联扩展到迭代式修复流程中。

8. 相关工作

挖掘修复模式。模式挖掘有助于构建修复规则和转换模板。FixMiner [14] 迭代地对原子变更的 AST 级编辑脚本进行聚类，并将这些模式整合到修复原型中。Getafix [15] 学习用于静态分析器警告的分层模式，并根据上下文对修复方案排序。Liu 等人 [16] 对违规及其修复中学到的功能进行聚类。Revisar [18] 从修订历史中挖掘反复出现的编辑，SPINFER [52] 则为大规模 Linux 变更推断语义补丁规则。VuRLE [10] 从易受攻击和已修复的代码中学习编辑与上下文模板，用于漏洞检测与修复。Pan 等人 [32]，、Zhong 和 Su [53]，以及 Sobreira 等人 [40] 对缺陷修复及其反复出现的模式进行了分析。PPD [54] 和 Coming [55] 可检测补丁和提交中的预定义模式。Yang 等人 [17] 提供了一个 Python 修复目录，为我们的人工修复操作标签提供了参考。本研究分析了补丁簇所体现的安全修复机制，并探讨这些机制与所恢复安全需求之间的适用关系。

补丁表示和分析单元。 TreeCen [36]和 DSFM [56]使用结构特征进行克隆检测，而 Feng 等人 [57] 研究基于词元的比较。GraphCodeBERT [58]学习感知数据流的代码表示。对于代码变更，CC2Vec [59] 对添加和删除的代码进行建模，CCRep [60] 将变更词元与上下文关联起来，而 GraphSPD [61] 结合补丁前后的代码属性图进行安全补丁检测。CodeReviewer [33] 针对代码变更和审查任务进行预训练，Jina [34]则用于代码检索。我们将由这些冻结编码器构建的补丁表示与 AST 编辑摘要进行比较，以组织修复机制。Li 和 Paxson [3]记录了安全补丁的局部性，而 Fixseeker [62] 考察了代码块之间的相关性。Herbold 等人 [63] 研究了修复缺陷的提交中的混合变更，Atomizer [64] 则分离提交意图。Sejfia 等人 [65]考察了检测跨越多个代码单元的漏洞所面临的挑战。我们的分析将每个补丁在各个函数和文件中的可用变更汇总为一个聚类点，从而支持对涉及多个位置的修复进行源代码分析。

安全影响表征。 SID [20] 使用安全规则比较和补丁模型来识别其影响这些影响由 Linux 内核补丁解决。DUALLM [21]结合提交消息与差异分析，并利用切片代码对内存错误进行分类。SPAIN [66]根据二进制变更推导漏洞模式和补丁模式，用于安全补丁分析。PatchDB [24] 表征安全补丁变更，并将修复模式作为其数据集的一种潜在用途加以展示。我们分别从漏洞描述中表征被违反的安全要求，并从基于代码的修复类别中进行区分，然后系统地衡量两者之间的关联关系。聚类后的示例说明，代码上下文如何区分针对同一影响的具体修复选择，从而将影响表征与修复知识的适用性联系起来。

知识引导的漏洞修复。Pearce 等人 [67]评估了大语言模型进行零样本漏洞修复的能力，而 CEDAR [68] 检索与代码相关的少样本任务（包括程序修复）的示例。给定 CWE 类型和易受攻击位置，APPATCH [69] 根据推断出的根本原因选择示例。VulMaster [70] 将 CWE 知识和示例融入训练好的修复模型中。VulKey [22] 按照 CWE 类型、句法操作和语义关键要素组织领域特定的修复知识。VulMatch [71] 将修复表示聚类为原型，以指导微调后的修复模型。CWEFT [72] 报告了在 Java 修复中标注 CWE 类别所带来的收益，而 VulnRepairEval [73]发现，在基于漏洞利用的验证下，增强提示带来的收益有限。我们的研究探讨了将安全需求作为基础，从修复知识库中检索相关修复示例，以支持检索增强生成。我们将补丁簇与测得的影响–模式关联联系起来，并评估由此得到的修复示例如何帮助大语言模型理解和修复新漏洞。

9. 结论

我们的大规模研究发现了语法不同的补丁中反复出现的修复模式，并将其与 12 类安全影响关联起来。由此形成的映射将被违反的要求与候选修复策略联系起来，而代码上下文则决定这些策略是否适用。

从该知识库中检索相关示例，可提升四种大语言模型的补丁生成效果，其中 Qwen 和 GLM 的提升尤为显著。在采用相同检索方法的情况下，Qwen 和 gpt-oss 在简单目标上的修复率与 GPT-6Luna 相近，而 gpt-oss 在中等目标上的表现也接近 Luna。相比之下，随机示例会降低三种模型的修复率，这凸显了相关性的重要性。对于更难修复任务，模型仍存在差距，这促使我们进一步研究如何检索和应用修复模式及安全影响知识，从而在保留预期功能的同时防止漏洞利用。

引用

[1] N. Dissanayake、A. Jayatilaka、M. Zahedi 和 M. A. Babar，“软件安全补丁管理——一项关于挑战、方法、工具与实践的系统性文献综述”，《信息与软件技术》，第 144 卷，页码 106771，2022 年。

[2] Z. Durumeric、F. Li、J. Kasten、J. Amann、J. Beekman、M. Payer、N. Weaver、D. Adrian、V. Paxson、M. Bailey 等，“Heartbleed 问题”，载于 *2014* 年互联网测量会议论文集，2014 年，页码 475–488。

[3] F. Li and V. Paxson，《安全补丁的大规模实证研究》，载于*2017* 年 *ACM SIGSAC* 计算机与通信安全会议论文集，2017 年，第 2201–2215 页。

[4] W. Weimer、T. Nguyen、C. Le Goues 和 S. Forrest，《使用遗传编程自动查找补丁》，载于*2009* 年 *IEEE* 第 *31* 届国际软件工程会议。IEEE，2009 年，第 364–374 页。

[5] C. Le Goues、T. Nguyen、S. Forrest 和 W. Weimer，《GenProg：一种用于自动软件修复的通用方法》，*IEEE*软件工程汇刊，第 38 卷，第 1 期，第 54–72 页，2012 年。

[6] F. Long 和 M. Rinard，《通过学习正确代码自动生成补丁》，载于*ACM SIGPLAN* 编程语言原理研讨会论文集（*POPL*），2016 年。

[7] Z. Qi、F. Long、S. Achour 和 M. Rinard，“生成并验证的补丁生成系统中的补丁合理性与正确性分析”，载于*2015*年国际软件测试与分析研讨会论文集，2015年，第24–36页。

[8] E. K. Smith、E. T. Barr、C. Le Goues 和 Y. Brun，“治疗是否比疾病更糟？自动化程序修复中的过拟合”，载于*2015*年第*10*届软件工程基础联合会议论文集，2015年，第532–543页。

[9] P. Fang、P. Gao、Y. Peng、Q. Zhang、T. Xie、D. Song、P. Mittal、S. Kulkarni、Z. Liu 和 X. Xiao，“Vfix：通过自动修复漏洞促进智能合约的软件维护”，载于*2024*年 *IEEE*软件维护与演化国际会议（*ICSME*）。IEEE，2024年，第13–24页。

[10] S. Ma、F. Thung、D. Lo、C. Sun 和 R. H. Deng，“VuRLE：通过从示例中学习实现自动化漏洞检测与修复”，载于 欧洲计算机安全研究研讨会。Springer，2017 年，第 229–246 页。

[11] C. S. Xia、Y. Wei 和 L. Zhang，“大型预训练语言模型时代的自动化程序修复”，载于 *2023 IEEE/ACM* 第*45*届国际软件工程会议（*ICSE*）。IEEE，2023 年，第 1482–1494 页。

[12]H. Li、Y. Tang、S. Wang 和 W. Guo，“PatchPilot：一种具有形式化验证早期尝试的高成本效益软件工程智能体”，载于 第*42*届国际机器学习会议论文集，机器学习研究论文集，第 267 卷。PMLR，2025年，第 35 922–35 941 页。 [在线]。可访问：https://proceedings.mlr.press/v267/li25cf.html

[13] C. S. Xia、Y. Deng、S.Dunn 和 L. Zhang，“揭秘基于大语言模型的软件工程智能体”，*ACM* 软件工程汇刊，第 2 卷，FSE 期，第 801–824 页，2025年。

[14] A. Koyuncu、K. Liu、T. F. Bissyand´e、D. Kim、J. Klein、M. Monperrus 和 Y. Le Traon，“FixMiner：挖掘与自动化程序修复相关的修复模式”，*Empirical Software Engineering*，第 25 卷，第 3 期，第 1980–2024 页，2020。

[15] J. Bader、A. Scott、M. Pradel 和 S. Chandra，“Getafix：学习自动修复漏洞”，*ACM* 编程语言汇刊，第 3 卷，OOPSLA 期，第 1–27 页，2019年。

[16] K. Liu, D. Kim, T. F. Bissyand´e, S. Yoo, 和 Y. Le Traon，《挖掘 FindBugs违规修复模式》，*IEEE*软件工程汇刊，第 47 卷，第 1 期，第 165–188 页，2021 年。

[17] Y. Yang, T. He, Y. Feng, S. Liu, 和 B. Xu，《通过分析细粒度源代码变更挖掘 Python修复模式》，*Empirical SoftwareEngineering*，第 27 卷，第 2 期，第 48 页，2022 年。

[18] R. Sousa、G. Soares、R. Gheyi、T. Barik 和 L. D’Antoni，《从代码仓库中学习快速修复》，载于第*35*届巴西软件工程研讨会论文集，2021年，第74–83页。

[19] J. Harer、O. Ozdemir、T. Lazovich、C. Reale、R. Russell、L. Kim等，《利用生成对抗网络学习修复软件漏洞》，发表于神经信息处理系统进展，第31卷，2018年。[20] Q. Wu、Y. He、S. McCamant 和 K. Lu，《通过符号规则比较，精准刻画大量补丁的安全影响》，载于*2020*年网络与分布式系统安全年度研讨会（*NDSS’20*），2020年。[21] X.Li、J.Pu、Y.Wu、X.Zou、S.Zhu、Q.Wu、Z.Zhang、J.Hsu、Y.Dong、Z. Qian等，《它们修复了什么？借助大语言模型对严重内存缺陷的安全补丁进行分类》，*arXiv* 预印本 *arXiv:2509 22796*，2025。。

[22] J. Li, Z. Chen, Y. Su, 和 M. R. Lyu，《Vulkey：由特定领域修复模式引导的自动化漏洞修复》，*ACM* 软件工程汇刊，第 3 卷，2487, 2026 2465 年，第 FSE 期，第 – 页。

[23] X. Mei、J. Del Castillo、P. S. Singaria、H. Xi、A. Benchikh、T. Bao、R. Wang、Y. Shoshitaishvili、A. Doup´e、H. Pearce 等，“Arvo：开源软件可复现漏洞图谱”，载于 *2026*年 *IEEE* 第 *11* 届欧洲安全与隐私研讨会（*EuroS&P*）。IEEE，2026年，第 860–873 页。

[24] X. Wang, S. Wang, P. Feng, K. Sun, 和 S. Jajodia，《Patchdb：大型安全补丁数据集》，载于 *2021* 年第 *51* 届 *IEEE/IFIP* 可信系统与网络国际会议（*DSN*）。IEEE，2021 年，第 149–160 页。

[25] G. Bhandari、A. Naseer 和 L. Moonen，“Cvefixes：从开源软件中自动收集漏洞及其修复方案”，载于第 *17* 届软件工程预测模型与数据分析国际会议论文集，2021 年，第 30–39 页。

[26] C. Wang, Y. Ni, Z. Shen, A. Yang, Y. Li, S. Chen, 和 Y. Zhou，《Megavul：一个基于代码表示的 Ac/c++漏洞数据集》，载于第 *21* 届软件仓库挖掘国际会议论文集，2024年，第 738–742 页。

[27] S. Lu, S. Park, E. Seo, 和 Y. Zhou，《从错误中学习：现实世界并发错误特征的全面研究》，载于 第 *13* 届编程语言与操作系统架构支持国际会议论文集，2008 年，第 329–339 页。

[28] S. A. Crosby 和 D. S. Wallach，《通过算法复杂度攻击实施拒绝服务》，载于 第 *12* 届 *USENIX* 安全研讨会（*USENIX Security 03*）。USENIX 协会，2003 年，第 29–44 页。

[29] L. Lucasto, D. Wagner, 和 C. Bratus，“基于Patterson-Sassaman-Morris 形式语言理论的安全应用”，*IEEESystems Journal*，第 7 卷，第 3 期，第 489–500 页，2013 年。

[30] M. Georgiev、S. Iyengar、S. Jana、R. Anubhai、D. Boneh 和 V. Shmatikov，“世界上最危险的代码：在非浏览器软件中验证 SSL 证书”，载于*2012* 年 *ACM* 计算机与通信安全会议论文集，2012 年，第 38–49 页。

[31]D. McGrew，“认证加密的接口和算法”，RFC 编辑器，RFC 5116，2008 年。 [在线]。地址：https://www.rfc-editor.org/rfc/rfc5116

[32] K. Pan, S. Kim, 和 E. J. Whitehead Jr.，《迈向对缺陷修复模式的理解》，*Empirical SoftwareEngineering*，第 14 卷，第 3 期，第 286–315 页，2009 年。

[33] Z. Li、S. Lu、D. Guo、N. Duan、S. Jannu、G. Jenks、D. Majumder、J. Green、A. Svyatkovskiy、S. Fu 等，《通过大规模预训练自动化代码审查活动》，载于第 *30* 届 *ACM* 欧洲软件工程会议与软件工程基础研讨会论文集，2022 年，第 1035–1047 页。

[34] D. Kryvosheieva、S. Sturua、M. G¨unther 和 H. Xiao，《来自代码生成模型的高效代码嵌入》，*arXiv* 预印本 *arXiv:2508.21290*，2025年。

[35] J.-R. Falleri、F. Morandat、X. Blanc、M. Martinez 和 M. Monperrus，《细粒度且准确的源代码差分》，载于第 *29* 届 *ACM/IEEE* 国际自动化软件工程会议论文集，2014 年，第 313–324 页。

[36] Y. Hu、D. Zou、J. Peng、Y. Wu、J. Shan 和 H. Jin，“Treecen：构建树图以实现可扩展的语义代码克隆检测”，载于第 *37* 届 *IEEE/ACM* 自动化软件工程国际会议论文集，2022 年，第 1–12 页。

[37] L. McInnes、J. Healy 和 J. Melville，“Umap：用于降维的统一流形近似与投影”，*arXiv* 预印本 *arXiv:1802.03426*，2018 年。

[38] P. J. Rousseeuw，“轮廓系数：一种用于聚类分析解释和验证的图形方法”，计算与应用数学杂志，第 20 卷，第 53–65 页，1987 年。

[39] D. L. Davies 和 D. W. Bouldin，“一种聚类分离度量”，*IEEE* 模式分析与机器智能汇刊，第 2 期，第 224–227 页，1979 年。

[40] V. Sobreira、T. Durieux、F. Madeiral、M. Monperrus 和 M. de Almeida Maia，“缺陷数据集剖析：来自 Defects4J 的 395 个补丁的解剖”，载于 *2018 IEEE* 第*25*届软件分析、演化与再工程国际会议（*SANER*）。IEEE，2018 年，第 130–140 页。

[41] H. Sun，“检测并补充漏洞描述中缺失的关键方面”，*ACM*软件工程与方法学汇刊（*TOSEM*），第 31 卷，第 3 期，第 1–27 页，2022 年。

[42] B. Beurdouche、K. Bhargavan、A. Delignat-Lavaud、C. Fournet、M. Kohlweiss、A. Pironti、P.-Y. Strub 和 J. K. Zinzindohoue，“混乱的联合状态：驯服传输层安全协议的复合状态机”，载于 *2015*年*IEEE*安全与隐私研讨会。IEEE，2015 年，第535–552页。

[43] Y. Yarom、D. Genkin 和 N. Heninger，“Cachebleed：针对 OpenSSL 常数时间 RSA 的定时攻击”，《密码工程杂志》，第7卷第2期，第99–112页，2017年。

[44] MITRE公司，“通用弱点枚举（CWE）”，https://cwe.mitre.org/，2026年，访问日期：2026-10-13。

[45] I. Bojanova，“漏洞框架（BF）：形式化网络安全弱点和漏洞”，美国国家标准与技术研究院，NIST特别出版物800-231，2024年。[在线]。可访问：https://doi.org/10.6028/NIST.SP.800-231

[46] CVE计划，“通用漏洞与暴露（CVE）”，https://www.cve.org/，访问日期：2026年11月16日。

[47] B. Efron 和 R. J. Tibshirani，《自助法导论》，统计与应用概率专著丛书。纽约：Chapman and Hall，1993年，第57卷。

[48] Y. Benjamini 和 D. Yekutieli，“依赖条件下多重检验中假发现率的控制”，《统计年刊》，第29卷，第4期，第1165–1188页，2001年。

[49] OpenAI，“GPT-6-luna模型”，OpenAI API文档，https://developers.openai.com/api/docs/models/gpt-6-luna，2026年，访问于 2026-10-16。

[50] S. Agarwal, L. Ahmad, J. Ai, S. Altman, A. Applebaum, E. Arbus, R. K. Arora, Y. Bai, B. Baker, H. Bao 等，“gpt-oss-120b 与 gpt- oss-20b 模型卡”，*arXiv* 预印本 *arXiv:2508.10925*，2025年。

[51] A. Zeng, X. Lv, Q. Zheng, Z. Hou, B. Chen, C. Xie, C. Wang, D. Yin, H. Zeng, J. Zhang 等，“Glm-4.5：智能体、推理与编程（ARC）基础模型”，*arXiv* 预印本 *arXiv:2508.06471*，2025年。

[52] L. Serrano、V.-A. Nguyen、F. Thung、L. Jiang、D. Lo、J. Lawall 和 G. Muller，《SPINFER：为 Linux 内核推断语义补丁》，载于*2020* 年 *USENIX* 年度技术会议（*USENIX ATC 20*）。USENIX 协会，2020 年，第 235–248 页。 [在线]。链接：https://www.usenix.org/conference/atc20/presentation/serrano

[53] H. Zhong 和 Z. Su，《真实缺陷修复的实证研究》，载于第*37*届 *IEEE/ACM* 国际软件工程大会（*ICSE*）论文集。IEEE，2015 年，第 913–923 页。[54] F. Madeiral、T. Durieux、V. Sobreira 和 M. Maia，《迈向缺陷修复模式检测的自动化方法》，arXiv:1807 11286，2018 年。[55] M. Martinez 和 M. Monperrus，《Co ming：一种从 Git 提交中挖掘变更模式实例的工具》，载于第*41*届 *IEEE/ACM* 国际软件工程大会：伴随会议论文集（*ICSE-Co mpanion*）。IEEE，2019 年，第 79–82 页。[56] Z. Xu、S. Qiang、D. Song、M. Zhou、H. Wan、X. Zhao、P. Luo 和 H. Zhang，《DSFM：通过深层子树交互增强功能性代码克隆检测》，载于第*46*届 *IEEE/ACM* 国际软件工程会议（*ICSE*）论文集。ACM，2024年，第 1–12 页。[57] S. Feng、W. Suo、Y. Wu、D. Zou、Y. Liu 和 H. Jin，《机器学习就够了：一种简单的基于令牌的方法，实现高效的代码克隆检测》，载于第*46*届 *IEEE/ACM* 国际软件工程会议（*ICSE*）论文集。ACM，2024年，第 1–13 页。

[58] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu, M. Tufano, S. K. Deng, C. Clement, D. Drain, N. Sundaresan, J. Yin, D. Jiang, and M. Zhou，“GraphCodeBERT：基于数据流预训练代码表示”，载于第九届国际学习表征会议（*ICLR*）论文集，2021。

[59] T. Hoang, H. J. Kang, D. Lo, and J. Lawall, “CC2Vec：代码变更的分布式表征，” in *Learning Distributed Representationsof codechanges,”in* 第*42*届 *ACM/IEEE* 国际软件工程会议论文集（*ICSE*）. ACM, 2020, pp. 518–529.

[60] Z. Liu, Z. Tang, X. Xia, and X. Yang，“CCRep：预训练代码模型和查询回溯学习代码变更表示”，载于第*45*届*IEEE/ACM*国际软件工程会议论文集（*ICSE*）。IEEE，2023，第17–29页。

[61] S. Wang, X. Wang, K. Sun, S. Jajodia, H. Wang, Q. Wang 和 C. Wang，“GraphSPD：基于图的补丁检测与丰富语义”，载于*2023*年*IEEE*安全与隐私研讨会论文集（*SP*）。IEEE，2023年，第2409–2426页。

[62] Y. Chen, D. Fang, Z. Shi 和 K. L. Sun，“Fixseeker：一种用于检测开源软件中静默漏洞修复的经验驱动型基于图的方法”，*ACM* 软件工程与方法学汇刊，2026年。 [在线]。可访问：https://doi.org/10.1145/ 3803410

[63] S. Herbold、A. Trautsch、B. Ledel、A. Aghamohammadi、T. A. Ghaleb、K. K. Chahal、T. Bossenmaier、B. Nagaria、P. Makedonski 等，“缺陷修复提交中纠缠现象的细粒度数据集与分析”，*Empirical Software Engineering*，第 27 卷，第 6 期，第 125 页，2022 年。

[64] K. Zhu、Z. Tian、S. Wang、M. Leng 和 X. Mao，“Atomizer：一种用于意图驱动的提交解纠缠的基于大语言模型的协作式多智能体框架”，载于第 *48* 届 *IEEE/ACM* 国际软件工程会议（*ICSE*）论文集。ACM，2026年，第 1419–1431 页。

[65] A. Sejfia、S. Das、S. Shafiq 和 N. Medvidovi´c，《迈向改进的基于深度学习的漏洞检测》，载于 *IEEE/ACM* 第*46*届国际软件工程会议（*ICSE*）论文集。ACM，2024年，第1–12页。

[66] Z. Xu、B. Chen、M. Chandramohan、Y. Liu 和 F. Song，《SPAIN：面向二进制文件的安全补丁分析，以理解痛点与药方》，载于 第*39*届 *IEEE/ACM* 国际软 件工程会议（*ICSE*）论文集。IEEE，2017年，第462–472页。

[67] H. Pearce、B. Tan、B. Ahmad、R.Karri 和 B. Dolan-Gavitt，《使用大语言模型检验零样本漏洞修复》，载于 *2023 IEEE*安全与隐私研讨会（*SP*）。IEEE，2023年，第2339–2356页。

[68] N. Nashid、M. Sintaha 和 A. Mesbah，《基于检索的代码相关少样本学习提示选择》，载于 第*45*届*IEEE/ACM*国际软件工程会议论文集（*ICSE*）。IEEE，2023年，第2450–2462页。

[69] Y. Cai, {APPATCH}: 自适应 {真实世界} 软件漏洞修补的大语言模型自动化自适应提示”，载于第*34*届 *USENIX* 安全研讨会（*USENIX Security 25*），2025年，第 4481–4500 页。

[70] X. Zhou、K. Kim、B. Xu、D. Han 和 D. Lo，“眼不见，心不烦：通过扩大输入范围和源代码改进自动漏洞修复”，载于 第*46*届 *IEEE/ACM* 国际软件工程会议（*ICSE*）论文集。ACM，2024年，第1–13页。

[71] X. Cao, W. Wu, “通过提取和匹配修复模式增强漏洞修复”，《系统与软件杂志》，第230卷，第112528页，2025年。

[72] L. A. Carbonell 和 A. Koyuncu，“CWEFT：面向 CWE 的自由文本提示与类型化提示评估”，载于 第*20*届实证软件工程与度量国际研讨会（*ESEM 2026*），《莱布尼茨国际信息学论文集》，R. Feldt、M. Paasivaara、D. Mendez、S. Wagner 和 M. M. Bar´on 编，第394卷。Schloss Dagstuhl – Leibniz-Zentrum f¨ur Informatik，2026年，第78:1–78:14页。

[73] W. Wang、W. Ma、Q. Hu、Y. Zhang、J. Sun、B. Wu、Y. Liu、G. Xu 和 L. Jiang，“VulnRepairEval：用于评估大语言模型漏洞修复能力的基于漏洞利用的评估框架”，arXiv 预印本 arXiv:2509.03331，2025年。 [在线]。获取地址：https://arxiv.org/abs/2509.03331[74] A. Agresti 和 B. Caffo，“通过加上两次成功和两次失败，得出比例及比例差的简便有效置信区间”，《美国统计学家》，第 54 卷，第 4 期，第 280–288 页，2000年。

附录 A. 补丁聚类与评估

队列与表示。 共享代码仓库和修复提交的记录会被合并，相当的反向移植补丁则归入修复族。由 3,816 个补丁组成的队列包括 1,633 个部分恢复样本。源组通过已记录的分叉、补丁依赖和相同的归一化差异，将代码仓库关联起来。这些关联定义了语义评估中使用的溯源组。CR 和 Jina 分别采用 512 个和 8,192 个令牌的编码器调用。长输入的读取结果通过不重叠分块计算加权平均，权重依据代码令牌数确定。几何分数使用各流程的工作空间，而语义得分使用相同的固定参考。

语义评估。 T_3 表示按簇中数量最多的三种修复类别计算的按补丁加权覆盖率。 $P_\times$ 衡量跨源组配对中主标签的一致性，使用 22 个经条件细化的标签，并排除未确定标签。对于查询 q，令 $G(q)$ 表示其所属的簇， P_q 表示已知的同模式伙伴， D_q 表示经过审查的同源组干扰项：

$$C_q = \frac{|P_q \cap G(q)|}{|P_q|}, \qquad M_q = \frac{|D_q \cap G(q)|}{|D_q|}.$$

C 对 143 个查询和 1,686 个有向伙伴对取平均值。 M 对 84 个查询和 447 个干扰项对取平均值。未判定关系并非负例。干扰项拒绝指定的模式，而非所有可能的共享模式。

标注与人工检查。 Qwen3.8 对仅含代码的视图和包含代码及修复上下文的视图进行标注，GPT-6 Luna 则对存在分歧和置信度较低的情况进行裁定。表10总结了本研究中的人工检查。每项审计均由一名软件安全标注员负责。类别审计包括 30 个按类别覆盖范围和边界案例选取的补丁，以及 70 个随机选取的修复族。影响审计会随机抽取描述，并纳入空类别集合。修复审计则对每种生成器–方法组合抽取三个已接受的候选方案。

表 10. 人工审计结果。

审计	样本	结果
修复类别	100 个补丁	主标签一致性：79/100
影响类别	100 条描述	类别集合完全一致率：84.0%，微平均 F1 值：0.902
修复评估	60 个已接受的候选项	目标路径阻断：55/60，在保留合法行为的同时进行阻断得到保留：38/60

修复判断包括38项完全确认、6项部分确认、2项错误判断和14个未解决案例。在未解决案例中，有13项会阻断目标路径，但需要更多代码上下文才能判断是否保留了合法行为。

附录B. 安全影响标注与验证

证据与类别边界。影响提取基于清理后的 CVE 描述，并记录每项操作、对象、被违反的要求及支持性引文。映射时对这些记录应用固定的代码本，不参考补丁或目标 CWE标签。此过程保留有证据支持的不同违规，并将证据不足、含义模糊或未涵盖的陈述留作未分类。死锁归入进展；如果陈述中还提到协调违规，则另行标注。进展还包括超出预期隔离边界的终止或异常逃逸。

统计违规链。 如果某项描述包含任何跨类别因果链接，则该描述在描述计数中计为一次。在方向分析中，我们统计附加到标记为存在因果关联的连续违规上的不同类别之间的每个有序对。在这类 138 个链接中，有 56 个从数值指向内存边界，23 个从并发指向对象-状态。总计 138 个链接中的 111 个（80.4%）从数值、并发或输入接受指向内存边界、对象-状态或引用与类型；只有一个链接在这些组之间沿相反方向运行。

验证。表10中的100条描述审计使用了清理后的描述和类别定义。类别集合完全一致率包括未分配任何类别的描述，而微平均 F1 值衡量的是单类别分配的一致性。

表 11. 主要修复—影响关联：支持度和流行率差异。

影响类别	修复类别／条件	n_{pi}/n_i	Δ [95% 区间] (pp)
内存边界	检查剩余输入	110/969	+8.8 [4.1, 13.8]
	检查目标容量	95/969	+9.5 [6.2, 12.5]
	检查对象范围	214/969	+18.8 [13.0, 24.6]
	边界处理	106/969	+8.4 [5.3, 11.8]
	提供容量	81/969	+8.1 [4.6, 12.2]
	校正大小或位置	84/969	+6.2 [3.3, 8.9]
对象状态	初始化或清除状态	118/555	+17.2 [5.1, 27.1]
	保护对象生命周期	208/555	+35.6 [24.2, 50.0]
	协调并发访问	50/555	+6.0 [0.9, 10.4]
引用与类型	检查指针有效性	209/380	+52.1 [43.5, 62.5]
数值	检查数值有效性	150/290	+47.8 [36.8, 59.5]
	更改数字表示	51/290	+16.5 [11.3, 21.9]
资源持有	释放与清理	158/233	+65.6 [53.8, 73.4]
进展	检查工作负载	18/157	+10.6 [2.9, 19.2]
并发	协调并发访问	71/130	+52.9 [14.2, 81.5]
输入接受	检查状态一致性	11/33	+23.1 [5.4, 42.6]
不可信解析	检查权限	10/65	+13.0 [6.6, 21.4]
	分离数据与指令	22/65	+33.7 [21.9, 47.0]
	限制权限或功能	9/65	+12.3 [5.5, 21.2]
信任决策	检查权限	62/149	+41.1 [15.8, 63.9]
	限制权限或功能	20/149	+12.2 [6.5, 19.8]
敏感数据	初始化或清除状态	11/35	+24.3 [8.5, 39.2]

附录 C. 修复—影响关联

这2,791个双重标注补丁构成283个代码仓库区块，其中Linux主线代码仓库与五个开发树归为一组。每个补丁贡献其主要修复类别以及每个已分配的影响类别。符合条件的修复标签必须来自五个区块中的20个补丁。每个经过测试的类别对还必须在影响类别内外各包含20个补丁，且类别内支持度覆盖五个区块。这些规则允许测试198个类别对，但对传播失败和密码学材料的测试支持度不足。

调整后的流行率估计 [74]，仓库分块自助法区间和区块置换支持推断。Benjamini–Yekutieli 校正 [48] 在每个影响类别内以及全局范围内应用于 $q \le 0.10$。主链接要求差异为正、区间支持，并且在 Linux、数据源和编辑规模分层中保持稳定。全部 22 项也通过了全局校正。表 11 报告了完整的主链接集合。

补丁簇内的影响一致性。 针对已分配影响标签的 2,838 个补丁，我们统计每个分区的簇中来自不同项目谱系的所有无序配对。如果一对中的成员至少共享一个影响类别，则视为一致。引用值为 300 次置换的均值；置换时固定簇分配，并在同一代码仓库的补丁之间打乱完整的影响标签集。

表 12. 同一簇中共享一个影响类别的跨谱系对。

分区	配对	观测值 (%)	打乱值（%）
CR	109,192	23.4	20.7
Jina	90,690	41.4	28.3
Diff-AST	282,553	24.6	21.0
TF-IDF	197,558	22.7	20.7

附录D
修复条件与鲁棒性

目标与编辑复杂度。 该基准包含 291 个留出的 C/C++修复目标，提供的修复前代码不超过 10,000 个字符，且样本池中至少有一个经过审查的同机制示例。代码长度标准适用于已完成的运行，不对示例重新采样。表13定义了表 8 中使用的三个编辑复杂度层级。

表13. 编辑复杂度分层。

层级	标准
Easy	一个函数、至多一个代码块和六个变更行，且没有新的标识符。
中等	处于简单层级和困难层级之外的目标。
Hard	至少两个函数、超过 15 个变更行，或至少两个新的标识符，或至少四个代码块。

新增行中出现了修复前函数中不存在的新标识符，但不包括关键字。

检索。该池提供来自 5,642 个补丁的函数级示例。每个查询都会排除其目标实现、关联的 CVE、修复提交和修复系列。同谱系示例在主要比较中仍然符合条件。冻结的约简器和最近质心会分配

附加补丁到现有聚类中。对于查询函数变更 d_f 和示例向量 V_u，排序使用$S(u) = \max_f \cos(d_f, V_u)$。选择过程会采用各不相同的修复系列，优先选择不同的代码库族，并从邻近聚类以及全局池中补足。影响引导会根据目标影响保留排名前三的修复类别，使用按系列计数的条件频率，并结合全局频率平滑，同时对多重影响进行等权处理。其类别过滤器在回退期间仍保持启用。缺少影响标签或符合条件的示例少于三个时，将使用零样本。

每个生成器提供三个假设补丁，每个补丁定义一个查询组，并附带开发者更改行或插入位置提示，但不包含替换代码。不可用的查询组使用零样本。预言机采用开发者信息辅助审查，优先选择相同机制示例，并用包含必要修复要素的示例填充剩余位置。表14报告了开发者查询和模型生成查询的 pass@3。

表 14. 所有 291 个目标中按查询类型统计的平均 pass@3（%），包括零样本回退。

方法	开发者修复	模型生成
聚类引导	49.2	41.6
影响引导	48.5	43.2

零样本和随机方法的平均值分别为 37.9% 和 36.9%。模型生成的查询会收到变更位置提示。

评估与聚合。第一轮生成要求提供修复计划，第二轮则要求提供修正后的函数。未作修改、无法解析、失败或被截断的输出均视为失败。精确规范化开发者差异匹配会被接受。对其他候选方案，使用易受攻击代码、参考差异和候选差异进行源代码级评估，同时隐去生成器、选择方法和示例。只有在阻止参考中的危险路径且产生相同控制流效果（包括以等效方式实现该效果）时，候选方案才会被接受。候选方案仅根据源代码进行评估，不进行编译或执行。在对 60 个已接受候选方案进行的人工审查中，55 个阻止了目标路径，其中 38 个还保留了合法行为（表10）。

对于每个目标，如果某组的三个候选项中有任意一个通过，该组得 1 分，否则得 0 分。对于随机条件和使用模型生成的查询进行检索的条件，我们对三组得分取平均。其他条件下，每个目标只有一组。报告的 pass@3 是这些目标得分的平均值，其中包括回退方案，且每个目标的权重相同。谱系消融会排除该目标记录的谱系；Jina消融则会同时替换候选池和查询表示。表15报告了这些比较的配对差异。

表 15. 四个模型的平均配对消融差异（百分点）。

对比	目标	Δ (pp)
开发者查询减去模型生成的查询		
聚类引导	285	+7.4
影响引导	285	+4.8
影响引导减去聚类引导		
开发者查询	291	−0.8
模型生成的查询	291	+1.7
差距差异（开发者减去模型查询）	291	−2.4
同谱系示例：允许减去排除		
随机	291	−0.3
聚类引导	291	+3.5
影响引导	291	+3.0
影响引导（模型查询）	291	+1.7
预言机	257	+3.4
跨谱系：影响引导减去随机		
增益，开发者查询	291	+8.3
相对于共享谱系池的变化	291	−3.3
增益，模型生成的查询	291	+4.4
变更与共享谱系池对比	291	−1.9
Jina 减去 *CR*		
聚类引导	291	+3.3
影响引导	291	+1.9
聚类引导（模型查询）	291	+0.9
影响引导（模型查询）	291	+2.0
Jina 的影响过滤器		
模型生成的查询	291	+2.8

前两行排除了不可用的模型查询组，并对四个模型均有配对得分的目标取平均值。其他比较包含零样本回退。所有差值均使用未舍入的得分计算。

对评估器的敏感性。在评估结果重叠的 218 个目标上，要求 gpt-oss-120b 和 Qwen3.8-27B 均接受某个候选方案后，在使用开发者查询的聚类引导检索和使用模型生成的查询的影响引导检索中，四种生成器相较随机条件的优势均得以保留。对于 GPT-6 Luna 生成的修复方案，在双模型规则下，这两项优势分别为 9.8 和 7.8 个百分点；而按 Luna 自身的判断，分别为 9.5 和 7.2 个百分点。分别评分时，两种开放权重评估器也都发现了这两项优势。

修复计划。 计划评估使用针对不安全操作、违反的条件和所需修复效果的固定引用。计划评估时不提供生成器身份、选择方法和最终候选项。问题识别要求诊断匹配，而完整性要求涵盖所有修复要素。得分先对各目标内的尝试取平均，再对目标和生成器取平均。