arXiv:2611.28584v1 [cs.SE] 21 Nov 2026

正确判定，正确理由？一项以执行为依据的研究：大语言模型漏洞检测器是否理解漏洞

Leizhen Zhang 路易斯安那大学拉法叶分校计算与信息学院，美国路易斯安那州拉法叶

ShengChen[1] 路易斯安那大学拉法叶分校计算与信息学院，美国路易斯安那州拉法叶

摘要

大语言模型（LLM）越来越多地用于检测漏洞，其基准分数也常被视为模型理解所报告漏洞的证据。我们通过程序执行检验这种看法。对于来自 BigVul 和 PrimeVul 的 515 个 C语言和 C语言++ 函数，我们通过概念验证复现了其中的漏洞，并确定故障发生的行以及故障运行所依赖的语句。基于这些事实，我们构建了一个五级测试阶梯：从区分易受攻击函数及其修复版本，到定位故障并对故障移除作出反应，再到判断经执行确认的编辑和预测具体运行的结果。每项测试都配有一个无需理解即可完成的基线，例如通过规则挑选包含高风险调用的行；此外，我们还针对模型训练截止时间之后公开的 195 个漏洞重复了这项研究。我们得到四项发现。第一，区分易受攻击函数及其修复版本并不容易。开放权重模型区分的配对数量不超过机会水平，不过其中五个模型表现出对修复的轻微敏感性；而商业模型仍将许多经过筛选的修复版本标记为易受攻击。第二，商业模型定位故障的能力优于关键词规则，但移除这些证据只会改变最强模型对较新漏洞的判定。第三，在给定程序输入后，商业模型预测运行是否会发生故障的表现优于查找新增检查的规则，而且预测结果会随输入而变化。第四，对于较新的漏洞，这一优势不复存在，最强模型的领先幅度也有所缩小，但定位能力仍然存在。我们公开了该框架、逐函数证据以及一个经执行确认的反事实基准。

2012 年 ACM 学科分类 安全与隐私 →软件安全工程；软件及其工程 → 软件测试与调试；计算方法 → 机器学习

关键词和短语 漏洞检测、大语言模型、动态程序切片、解释评估、反事实评估

数字对象标识符 10.4230/LIPIcs.ECOOP.2027.

1 引言

基于学习的漏洞检测已经从代码小工具和程序图 [31, 62, 8] 发展到微调后的变换器 [19, 43], , 如今通用大语言模型（LLM）不仅会对函数 [28, 49, 44, 16]给出判定，还会提供自然语言解释。开发者利用这些解释来决定需要检查哪些行，以及补丁是否修复了故障。标签噪声和训练数据泄漏会夸大报告中的分数 [14, 16], ，检测器会利用数据收集中的人为痕迹 [25, 6], ，而且难以区分易受攻击函数与其修复版本 [39, 38]。因此，一个正确判定可能建立在

[1] 通讯作者。

© Leizhen Zhang 和 Sheng Chen；根据知识共享许可协议 CC-BY 4.0 授权。莱布尼茨国际信息学论文集 Schloss Dagstuhl – Leibniz-Zentrum für Informatik，Dagstuhl Publishing，德国

XX:2 正确的判定，正确的理由?

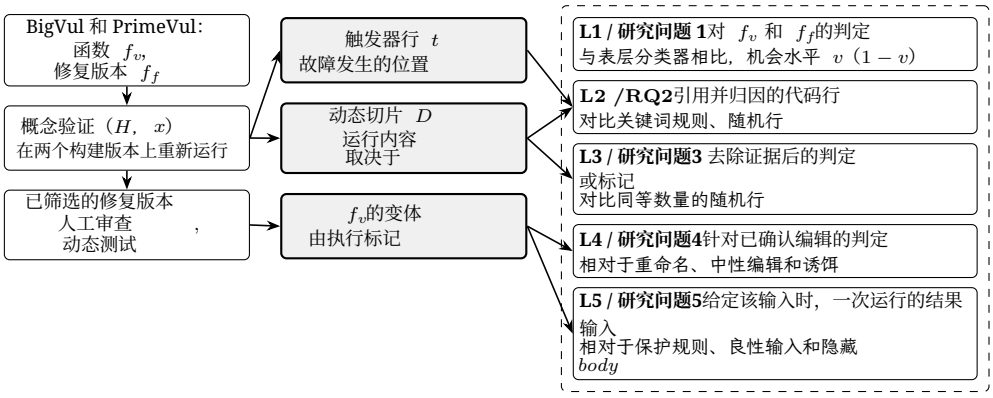


图1 研究概览。每个函数都附带一个重放的概念验证，我们以此获取基于执行的参照（第2节）以及编辑后变体的标签。五项测试分别回答一个研究问题，并与各自的捷径或对照进行比较（第3节）。我们在515个函数上测试了13个模型，并在训练截止时间之后发布的195个通用漏洞披露上重复进行这些测试（第3.3节）。

错误的理由之上。我们要问的是：当大语言模型检测器作出正确判定时，它是否是因为理解了使该函数易受攻击的真正原因?

回答这一问题需要三个要素。

C1. 针对理由的参考基准。现有研究将解释与补丁代码行、静态切片或人工判断进行比较 [20, 23, 11, 49]。补丁代码行描述修复过程，可能遗漏故障位置；静态切片取决于分析和切片准则 [12, 52], ；人工推理依据则成本高且具有主观性。我们为每个函数重放一次概念验证，并提取故障运行的两个事实：故障发生所在的触发器行，以及运行所依赖语句的动态依赖切片 [29, 3]。该切片平均覆盖函数的16.7%，且无需人工标注。

C2. 不理解漏洞的基线。正确判定可能源于记忆中的常见漏洞与披露（CVE）条目 [7, 54], ，也可能源于新增边界检查等表面特征，或偏向于易受攻击；而合理的引用则可能源于风险关键词。因此，每项测试都配有一个无需理解的捷径，或配有一个在不针对证据的情况下实施可比变更的对照。

C3. 经执行确认的干预。模型可以引用正确的行，却不使用这些行 [26, 48]。我们会在提示词中遮蔽或标记证据，并为每个函数构建编辑后的版本，再通过在这些版本上运行概念验证来确定其标签。

研究。这些要素构成了一个五级测试阶梯，每个研究问题对应其中一个层级（图1和表1）。**研究问题 1** 询问模型能否区分易受攻击函数与其修复版本（L1）；**RQ2** 询问模型的证据是否指向故障（L2）；**研究问题3** 询问其判定是否取决于这些证据（L3）；**研究问题4** 询问其判定是否与经执行确认的函数编辑相符（L4）；**研究问题5** 询问其能否预测一次具体运行的结果（L5）。我们评估了九种开放权重模型和四种商业模型，测试对象为来自 BigVul 和 PrimeVul 的 515 个 C语言及 C++ 函数 [18, 16] ，并在 195 个通用漏洞披露上重复实验；这些漏洞均在商业模型的训练截止时间之后公开并修复。我们得出了四项发现。

F1值。区分易受攻击函数与其修复版本并不容易（研究问题 1）。没有开放权重

模型区分出的配对数量超过机会水平。其中五个模型对修复表现出较弱的敏感性，因为在判定不同的配对中，判对的数量多于判反的数量，但它们对大多数配对中的两个版本仍给出相同的判定。商业模型表现更好，但两个 GPT 模型将约一半配对中的两个版本都标记为易受攻击，尽管每个修复版本都通过了我们的动态筛查和人工筛查。

- **F2. 定位超越了捷径，但对定位的依赖并不全面（研究问题2、研究问题3）**。商业模型引用触发器行的频率远高于引用相同行数的关键词规则，而且其引用行比三种静态切片中的两种更符合动态切片。屏蔽动态切片会降低其中三个模型的成对判别能力。Claude Opus 5.5 的判定则保持不变。
- **F3. 判定经得起保留标签的编辑，而具体输入有助于执行推理（研究问题4、研究问题5）**。重命名和中性编辑不会改变判定。在给出驱动程序和输入后，四种商业模型预测运行是否会发生故障的表现均优于这样一种规则：该规则已知易受攻击版本，并接受所有新增检查。这主要是因为它们会将开发者的修复预测为无故障，而且其预测会随输入而变化。
- **F4. 最强结果中的一部分依赖于已知漏洞（第9节）**。对于截止时间之后发布的通用漏洞披露，定位能力依然存在；在 L5 上相较于防护规则的优势消失；Claude Opus 5.5 在 L1 上的领先优势缩小，与截止前集合中最新的通用漏洞披露相比也是如此。

贡献。 (1) 一种基于执行的漏洞解释参照基准，由重放的概念验证构建，并与补丁代码行及三个静态切片器进行比较（第 2 节）。(2) 一个五级测试阶梯，每一级都包含一个捷径或一个对照，以及一个经执行确认、包含 315 个变体的反事实基准（第 3 节和第 7 节）。(3) 对十三个模型在其训练截止时间之前和之后的评估，分别考察定位、对证据的依赖以及执行推理（第 4 节至第 9 节）。我们发布该框架、逐函数证据和基准。

2 基于执行的参照

要判断检测器是否以正确的理由得出正确结论，需要对该理由作出一种独立于检测器、补丁以及任何特定分析工具的说明。本节基于一次经确认的故障执行定义这种说明，介绍我们如何获取它，在两个函数上进行演示，并将其与先前工作的参照基准进行比较。

2.1 定义和范围

▶ **定义 1**（实例）。一个实例是一个元组（f_v, f_f, H, x）。 f_v 是一个标记为易受攻击的函数， f_f 是其修复提交后的版本， H 是一个围绕函数 f 的某个版本构建可执行程序 $H[f]$ 的测试框架，而 x 是一个输入。当运行 $H[f_v](x)$ 表现出预言机检测到的故障，且运行 $H[f_f](x)$ 不包含该故障时，该实例得到确认。

预言机是消毒器报告 [42, 45], 、断言失败，或退出码中的行为差异。我们用行(f_v)表示 f_v 中非空的源代码行。下文定义的所有参照基准都是行(f_v)的子集，因此可以将它们与检测器针对 f_v报告的行号进行比较。

XX：4 正确的判定，正确的理由?

表1 五项测试。每个层级回答一个研究问题。捷径无需理解漏洞，而对照则是在不针对证据的情况下进行可比的更改。L4 和 L5 的标签通过执行得到确认。

研究问题和层级	Test	与（捷径或对照）相比
研究问题 1 L1 区分	关于 f_v 以及关于 f_f的判定，每个单独展示	捷径：基于表面特征（如长度）的分类器；对照：一个忽略其输入的检测器，达到机会水平 $v(1-v)$
RQ2 L2 定位	被引用的行或排名靠前的行相对于触发器行以及切片	捷径：一种引用包含高风险调用的行和随机行的规则
研究问题3 L3 证据敏感性	证据后的判定在提示词中被置空或标记提示词	对照：置空的随机行数量相同或被标记
研究问题4 L4 反事实	对.....的判定执行确认的编辑函数的	对照：重命名和中性编辑，它们保留标签；捷径：接受每一项新增检查，其中诱饵击败
研究问题5 L5 执行	结果和故障行在给定驱动程序的某次运行中和输入	捷径：寻找边界的规则检查；对照：同一个驱动程序，使用一个良性输入，且函数体隐藏

▶ **定义 2**（触发器行）。设 ρ 为故障运行 $H[f_v](x)$。触发器行 $t \in$ 行（f_v）是故障报告中最内层且位于 f_v 内的栈帧对应的行。

当故障由 f_v 自身的语句引发时，t 就是故障语句；当故障在被调用函数内部引发时（例如 `memcpy`），t 就是 f_v 中当时正在执行的调用。有些已确认的实例在 f_v中没有报告帧，因为两次运行的差异仅在于退出状态，例如拒绝服务或缺少授权检查；也可能是因为报告帧位于 f_v之外。对于这些情况，我们会定位修复决定结果的位置。

▶ **定义 3**（基于补丁得出的触发器行）。**</style}令 行**（f_v）为修复所作用的行：修复替换或删除的每一行，以及对于每处插入，插入点之后的 f_v 的第一条语句。基于补丁的触发语句是 C 中在 ρ里执行的第一行，其中条件优先于赋值和调用。

基于补丁的触发语句依据开发者的修复来确定，并标记两个版本在 x上开始出现分歧的位置；该位置未必是故障显现之处，因此依赖于它的结果会单独报告。在本研究的 515 个函数中，363 个的触发器行取自故障报告，152 个则是基于补丁得出的；补充材料的 S14 节给出了详细规则。触发器行指明一条语句。故障原因还涉及产生问题值并决定故障语句得以执行的语句，而动态切片 [29, 3] 会针对一次运行捕获这些语句。

▶ **定义 4**（动态依赖）。设运行 ρ 为已执行指令实例的序列。当某个实例 j 直接依赖于较早的某个实例 i 时，满足以下条件之一：j 使用了由 i 定义的寄存器值（数据）；j 通过由 i 计算出的地址访问内存（地址）；j 读取了一个字节，而该字节最近一次写入由 i 完成（内存）；j 的执行由分支 i决定，即从后支配节点的意义上说，j 所在的代码块控制依赖于 i所在的代码块，并且 i 是此类分支最近一次执行（控制）；或者

i 和 j 分别是一次调用及其被调用函数对参数的使用，或一次返回及调用方对返回结果的使用（过程间）。

▶**定义5**（动态切片）。设 s_t 为触发器行上故障访问的实例；对于基于补丁的触发语句，则设其为在 ρ 中的最后一次执行（当 ρ 不终止时，为其第一次执行）。动态切片 $D \subseteq$行（f_v）是 f_v 中满足以下条件的行集合：其中包含一条指令实例，该实例可通过直接依赖到达 s_t，并且还包括 t。

依赖链可能通过调用离开 f_v，再返回到其中；切片保留此类依赖链上 f_v 的行，因为检测器只能看到该函数。 D的每一行都在 ρ 中执行过，并且位于一条最终到达故障访问的依赖链上，因此不可能仅通过列举许多行来实现与 D 的一致。

▶**命题 1**。设 S 是由切片器从触发器行计算得到的 f_v 的静态后向切片，该切片器对所有执行中的五种依赖进行过近似。那么 $D \subseteq S$。

该命题成立，因为每个直接动态依赖都是 ρ 中静态依赖的一次出现，而此类切片器会纳入该静态依赖（证明见补充材料第 S14 节）。因此，动态切片中存在而静态切片中缺失的行，表明静态切片器漏掉了某个依赖，例如它未建模的别名所导致的依赖。

范围。参考基准描述了故障出现的位置，以及观测到的运行通过哪些值和决策到达该故障。通过分配状态作用于故障的根因（如示例 2 所示），或由函数 [60], 中缺失的代码构成的根因，均在其范围之外。因此，与参考基准的重合度是衡量定位效果的保守指标；此外，还通过对程序进行干预的测试来补充定位测试（第3节）。

2.2 构造与选择

我们在 LLVM [30]上实现了参考基准。动态依赖的概念和切片算法采用的是既有方法 [3, 59]，具体形式为 Giri [41]所实现的方法。我们的流水线增加了相关机制，使这些方法能够大规模应用于真实程序中由消毒器检测到的故障：从故障报告推导出判据、与消毒器共存且能在故障发生时程序中止后继续工作的跟踪机制、对未插桩库代码写入的内存进行建模，以及为每个切片明确给出完整性判定。

概念验证。BigVul 和 PrimeVul 包含每个函数的文本及其修复提交，但两者都不包含触发输入或测试框架。VulValidate [57], 为每个实例生成了测试框架 H 和输入 x；在此过程中，Claude Opus 5.0 会对修复前后的版本调用动态分析工具。我们按照 ARVO 的方法 [33] 重新构建这两个版本。模型只提出驱动程序和输入，其是否被接受则通过机械方式判定。当同一输入在两次运行中都能使易受攻击的构建版本发生故障、该故障可通过其行范围归因于已标注函数，且修复后的构建版本运行正常时，我们便将该实例计为已确认；在实例进入研究之前，我们的切片流水线还会在源代码树的新副本中再次重放每个已确认实例。每个已确认实例都附带一个自包含的证据文件夹，其中包含构建脚本、驱动程序、函数的两个版本、保持原始字节形式的输入，以及两次运行的日志。

切片与验证。该流水线会在源代码树的私有副本中重新执行概念验证，并根据恢复出的命令重新构建两个版本。判据是报告在触发器行上指出的访问的最后一次执行实例，因此，像 `dst[i] = src[i]` 这样发生越界读取的语句，其切片从该次读取开始。在优化和消毒器插桩之前插入的 LLVM 遍会跟踪基本

块、内存访问、调用和返回，并通过对跟踪进行反向搜索，收集判据所依赖的实例。只有当跟踪未被截断且运行在判据处停止时，切片才会被视为完整。在采用 UBSan 的 227 个构建版本中，没有后继块的处理程序块会增加定义 4 所排除的控制依赖，示例 2 即属此类。重新计算控制依赖并排除这些依赖后，在每个最多处理 120 秒的 514 个切片中，有 36 个发生变化（1 个函数超出此限），其行数减少了 150 行（原有 4,603 行），且每个切片比较的方向均保持不变（补充材料第 S27 节），因此结果采用计算得到的切片。我们使用 31 个具有已知依赖关系的对照程序验证了切片器，每个程序都针对一种结构进行测试，例如间接调用、异常或库回调；此外，我们还将其与 LLVM 3.4 上的原始 Giri 实现进行了对照（第 S10 节）。

筛选。在 BigVul [18] 和 PrimeVul [16] 中，经确认且具有完整切片的 739 个实例里，合并文本相同的实例后剩下 728 个不同的函数，其中 672 个具有触发器行以及第 2.4 节中的所有静态切片。在选取这些函数之前，两位作者分别独立检查每个函数的修复版本，且检查时均未看到任何模型输出；只有当两人都认为，对于调用方传入的参数，该函数的修复版本不存在任何类型的缺陷时，该函数才会成为候选对象。随后对修复版本进行三项动态检查。首先，修复版本必须能在触发输入上成功运行，既要在证据文件夹中如此，也要在重新构建开发者的修复后如此；此项检查排除了 39 个函数。其次，动态确认必须明确指出标注的通用漏洞披露（CVE），因此排除了 106 个函数，因为这些函数的确认过程触发了另一个真实缺陷，或未对该通用漏洞披露作出判断。进一步测试不得发现可归因于该修复函数的任何缺陷或漏洞；此项检查又排除了 12 个函数。工具的选择依据故障类型而定：284 个函数使用 AddressSanitizer，42 个使用未定义行为消毒器，28 个使用内存泄漏消毒器，26 个使用 MemorySanitizer，1 个使用线程消毒器；11 个使用断言，8 个使用超时；对于弱点不会导致内存故障的 128 个函数，则比较输出或授权决策；另有 13 个函数使用了两类工具。每个修复版本都在其证据文件夹中的触发输入上运行两遍，并在我们的重建版本中运行两遍；运行结果显示，在 515 个函数中，有 494 个函数实际执行了修复函数；此外，109 个修复版本还会在其他攻击输入上运行。最终纳入研究的是来自 152 个项目的其余 515 个函数，其通用漏洞披露（CVE）标识符年份介于 2005 年至 2022 年之间。其中 481 个具有修复文本，可组成匹配对；在这些匹配对中，修复版本是开发者针对标注的通用漏洞披露（CVE）所作的修复，并被 BigVul 或 PrimeVul 标记为非易受攻击；筛查后我们保留该标签。其余 34 个函数均来自 PrimeVul，数据集中没有其修复文本，因此仅用于展示易受攻击版本本身的度量指标。补充材料第 S10 节列出了每个函数所用的工具和运行情况，表 S10 则列出了完整的筛选过程。

2.3 两个示例

▶ **示例 1**（修复对故障访问进行了守卫条件保护）。在 `seamless_process` 的 rdesktop 中（图 2a，CVE-2018-20182），概念验证传入了一个流，其读取指针 `s->p` 已经位于 `s->end`之后 8 个字节的位置。第 6 行计算出 `pkglen = s->end - s->p`，该值作为无符号值发生回绕；第 9 行的复制操作，即触发器行，读取超出接收缓冲区的内容。动态切片包含长度计算、按该长度进行的内存分配以及复制操作。修复在第 4 行之后插入了对流状态的检查（图 2a 中的灰色行）；对于相同的输入，修复后的构建版本会在此处以协议错误停止，因此，根据修复行构建的参考基准会指向故障值计算之前的位置。GPT-6.1-sol 引用了第 3、6、8、9 和 10 行，这些行包含动态切片，并对 f_v 回答“是”，置信度为 98；对 f_f 回答“否”，置信度为 10。

▶ **示例 2**（参考基准的一个局限）。在 `split` 的 shapelib 中（图 2b，CVE-2022-0699），

(a) `seamless_process`，rdesktop，CVE-2018-20182。第 9 行复制操作中的越界读取。

		动态切片	Joern	dg	深度-悟空补丁		GPT-6.1-sol 证据
1	`seamless_process(STREAM s)`				1		
2	`{`						
3	`unsigned int pkglen;`						3
4	`char *buf;`						
	`struct stream packet = *s;`					5	
	`if (!s_check(s))`					7	
	`{`					8	
	`rdp_protocol_error("seamless_process(), stream is in unstable state", &packet);`					9	
	`}`					10	
5							
6	`pkglen = s->end - s->p;`	6	6	6	6		6
7	`/* str_handle_lines requires null terminated strings */`						
8	`buf = xmalloc(pkglen + 1);`	8	8	8	8		8
9►	`STRNCPY(buf, (char *) s->p, pkglen + 1);`	9	9	9	9		9
10	`str_handle_lines(buf, &seamless_rest, seamless_line_handler, NULL);`						10
11							
12	`xfree(buf);`						
13	`}`						

(b) `split`，shapelib，CVE-2022-0699。第29行发生双重释放。

		动态切片	Joern	dg	深度悟空补丁		GPT-6.1-sol 证据
1	`static char ** split(const char *arg, const char *delim) {`		1		1		
2	`char *copy = dupstr(arg);`	2	2	2	2		
3	`char **result = NULL;`	3	3	3	3		
4	`int i = 0;`	4	4	4	4		
5							
6	`for (char *cptr = strtok(copy, delim); cptr; cptr = strtok(NULL, delim)) {`	6	6	6	6		
7	`char **tmp = realloc (result, sizeof *result * (i + 1));`	7	7	7	7		7
8	`if (!tmp && result) {`	8	8	8	8		8
9–15	第一次重新分配的错误分支，该分支会释放词元、结果和副本						
16	`result = tmp;`		16	16	16		16
17	`result[i++] = dupstr(cptr);`	17	17	17	17		17
18	`}`	18					
19							
20	`free(copy);`		20		20		20
21							
22	`if (i) {`	22		22	22		
23	`char **tmp = realloc(result, sizeof *result * (i + 1));`	23	23	23	23		23
24	`if (!tmp) {`	24	24	24	24		24
25	`while (i > 0) {`	25			25		
26	`free(result[--i]);`	26			26		
27	`}`						
28	`free(result);`				28		
29►	`free(copy);`	29	29	29	29	29	29
30	`return NULL;`						
31	`}`						
32	`result = tmp;`						
33	`result[i++] = NULL;`						
34	`}`						
35							
36	`return result;`						
37	`}`						

图 2 示例1和示例2的函数及其参照基准。 ► 标记触发器行。Joern、dg 和 DeepWukong 列分别表示这些工具生成的静态切片（DeepWukong 向后切片至触发器行）。其他各列均列出相应参照基准所包含的每一行的行号。补丁：修复所修改或删除的每一行的行号。灰色行是修复所插入的行，这些行仅存在于修复版本中，Patch 列给出其在修复版本中的行号。最后一列列出 GPT-6.1-sol 引用作为证据的行。在 (b) 中，第 9 行至第 15 行被省略，补充材料中的图 S3 给出了完整代码清单。所有度量指标均使用完整函数。

参数字符串的副本在第 20 行被释放。当第 23 行的 `realloc` 失败时，错误分支会在第 29 行再次释放该副本，而修复删除了这一步。驱动程序在首次运行后每次运行都会失败，AddressSanitizer 会报告第 29 行发生双重释放，并提供两次释放的调用栈；修复后的构建版本则运行正常。动态切片沿着指针 `copy` 从第 2 行追踪至第 29 行，以及导致其到达该处的各项决策。第 25 行和第 26 行是实现带来的产物，因为第 26 行对 `-i` 的 UBSan 检查最终进入一个没有后继节点的处理程序；后支配分析将其视为出口，因此第 29 行看起来控制依赖于循环条件（第 2.2 节）。切片遗漏了第 20 行的首次释放；该释放对第 29 行的影响经由堆状态传递，而堆状态不属于五种依赖类型中的任何一种。Joern 和 DeepWukong 都包含第 20 行，而 dg 将其遗漏。GPT-6.1-sol 引用了两次释放，并针对易受攻击版本回答 YES，置信度为 100。若参考基准仅依据动态切片进行评判，就会把它对第 20 行的引用算作多余的一行，尽管那次释放才是故障的根因。

XX：8 正确的判定，正确的理由?

表 2 针对动态切片的静态切片。保留：静态切片所包含的动态切片占比，均值及95%区间。外部：静态切片中位于动态切片之外的占比，均值。大小：静态切片与动态切片大小之比，中位数。行 Jaccard：公式（1）的$J(S,\ D)$，中位数。最后一列：静态切片包含整个动态切片的函数占比（%）。

静态切片	保留（%）	外部（%）	Size	Line Jaccard	包含整个动态切片 (%)
Joern	79.5 [77.2，81.7]	32.4	1.14×	0.545	39.8
dg	97.4 [96.3，98.3]	39.0	1.56×	0.600	89.1
DeepWukong，回溯至触发器行	85.0 [82.8，87.0]	42.9	1.50×	0.500	53.6

2.4 与现有参照基准的比较

修复代码行。 在 342 个配对函数中，有 341 个观察到了触发器行，且其修复会更改 f_v 中的一行；在这 341 个函数中，40.2% 的修复会更改触发器行本身，而在 29 个函数中，所有更改的行都位于触发器行之后。因此，仅依据补丁代码行构建的参考基准可能会遗漏已正确识别的故障位置。

静态切片。 静态切片是以往研究所选用的参照基准，DeepWukong [12]将其用作模型输入。我们根据同一判据计算出三种静态切片。Joern 切片 [52] 是已标注函数程序依赖图中数据依赖和控制依赖的向后闭包，限定在函数内部，且不进行别名分析。dg 切片 [9] 基于 LLVM 位码，结合指针分析以及跨函数的依赖关系计算得出。DeepWukong 切片是其已发表流程的修正版：该流程从四类程序点向后和向前进行切片；对于 BigVul，还会合并已修复函数的切片。我们的变体仅进行向后切片，从易受攻击函数中直至触发器行的程序点开始切片（补充材料第 S7 节）。

命题 1 预测，可靠的切片器应达到 100% 的保留率，而表 2 显示，每种静态切片器都会遗漏故障所执行的依赖关系。dg 保留的动态切片最多，达到 97.4%；Joern 为 79.5%，配对差异为 17.9 [15.7、20.4] 个百分点；对于完整包含动态切片的函数，dg 有 459 个，而 Joern 只有 205 个。无论分别考察每个数据集，还是采用更严格的切片准则匹配规则，排序均为 dg、DeepWukong、Joern。例如，在图 2b 中，Joern 切片缺少第 22 行的守卫条件`if (i)`。更高的保留率是以更大的切片为代价获得的，因为 dg 选取的内容占函数的 31.7%，而动态切片占 16.7%；此外，每种静态切片中有 32% 至 43% 的行位于观测运行的依赖关系之外。已发表的 DeepWukong 流程选取的内容占函数的 57.8%，并且在 19.0 % 的函数中遗漏触发器行，因此它会奖励引用许多行的任何解释；将其限制为截至触发器行的向后切片后，其对动态切片的保留率从 79.4% 提高到 85.0%（补充材料第 S9 节）。因此，触发器行和动态切片构成主要参照基准，而静态切片则表明哪些结论取决于所采用的参照基准类型。

3 研究设计

表1列出了五个层级。模型可能在一个层级上超过基线，却在另一个层级上仅达到随机水平，因此每个层级都要与其各自的捷径或对照进行比较。

3.1 层级、度量指标与基线

L1，区分。检测器每次查看一个带编号的函数，并回答其中是否包含安全漏洞。每个匹配对由已确认的易受攻击版本 f_v 和经过筛选的修复版本 f_f组成；若将 f_f 判定为易受攻击，则与基准标签不符。对于平衡配对，始终回答易受攻击的检测器，即使无法区分任何配对，F1值也能达到 0.67（补充材料中的表 S11 列出了更多退化检测器）。因此，我们的主要度量指标是成对正确率（P-C） [16],，即在多少配对中， f_v 被判定为易受攻击，而 f_f 被判定为不易受攻击。准确区分率（ADR） [58] 采用相同判据：在匹配的 SAT/UNSAT 配对中，两项判定都必须正确。我们还报告两种版本均被判定为易受攻击的配对比例（P-V），并与 P-C 一并呈现。若检测器独立地以概率 v 回答易受攻击，其预期 P-C 为 $v(1-v)$，至多为 0.25，即各模型的随机水平。对于给两个版本相同判定的检测器而言，随机水平是一个要求很高的基线：即使其他判定方向正确，其 P-C 仍远低于该水平。因此，我们还会检验判定不同的配对是否方向正确。在平衡配对中，召回率减去假阳性率，等于正确配对比例减去反向配对比例；我们采用精确 McNemar 检验，并对十三个模型进行 Holm 校正，以比较这两项计数。我们还报告马修斯相关系数（MCC） [13], ；对于判定不随标签变化的任何检测器，MCC 始终为 0。MCC 与 P-C 回答的是不同问题：检测器可能凭借对所有版本都容易判断的函数获得正 MCC，却只能区分少数配对。曲线下面积（AUC）衡量模型分数将易受攻击版本排在修复版本之前的能力；对于在答案词元处读取的开放权重模型，该分数为 $\log P(\text{YES}) - \log P(\text{NO})$，其他情况下则使用指定置信度。捷径是一个表层特征分类器：它对单个版本的十项特征进行逻辑回归，例如长度以及 `if` 语句、NULL 检查和高风险调用的数量；采用五折交叉拟合，并确保同一通用漏洞披露（CVE）的所有函数都归入同一折。

L2，定位。显式证据是模型针对f_v给出判定时返回的行号列表。归因证据适用于在答案词元处读取的开放权重模型，它通过留一法对行进行排序：每次屏蔽一行，并将模型得分的下降幅度视为该行的影响 [56], ；也可以通过梯度 $\times$ 输入或积分梯度 [46]来计算。对于作为证据提供的一组 E 行，以及一个参考基准 T（触发器行或其中一个切片），我们报告覆盖率和行 Jaccard，

$$\mathrm{Cov}(E,T) = \frac{|E \cap T|}{|T|}, \qquad J(E,T) = \frac{|E \cap T|}{|E \cup T|}, \tag{1}$$

按函数分别计算并取平均。引用的行越多，覆盖率就越高；而无论遗漏参考行还是添加多余行，行 Jaccard 都会下降，因此它是我们的主要度量指标。对于排序结果，我们选取与参考基准行数相同的排名最高的行。每个数值都附有行数相同的随机行所对应的精确期望值。故障往往聚集在高风险操作附近，因此更强的捷径是关键词规则：它优先引用 $|E|$行，即包含高风险标识符的行，例如 `memcpy` 或 `malloc` ，之后再引用随机行；对于排序，则按各行中高风险标识符的数量排序。

L3，证据敏感性。证据移除会将 f_v中的一组行置空，同时保留行号和缩进，再将处理结果与未更改的 f_f配对，并重新计算 P-C。移除的行集可以是动态切片、DeepWukong 切片、单独的触发器行，或与动态切片行数相同的随机行；另有一个条件仅保留动态切片。引导会在提示词中标记两个版本中对应的行，因此标记本身不会泄露哪个版本易受攻击；或者会将动态

依赖边作为文本附加进去。若存在同等规模的随机行集，则将该行集与其比较；否则与未更改的代码对进行比较。置空会同时改变内容、语法以及风险操作的存在情况，而在预期应有代码的位置出现空行，本身也可能成为一种信号（第9节）。第三种观察性检查关注模型能够区分的代码对，是否正是它引用触发器行的那些代码对。因此，L3 度量的是判定对参考证据的敏感性；参考证据可能不同于模型所陈述的行。若要检验模型陈述的行是否就是它实际使用的行，则需对这些行进行干预，并采用结构匹配的对照。

L4，反事实。 检测器会根据 L1 中的问题，逐一接收函数的不同变体；这些变体的标签由执行结果确定（第7节）。有些编辑不会改变标签，包括重命名两个版本中的所有局部变量和参数，以及在 f_v 中插入一条中性语句。其他编辑则会消除故障，包括开发者的修复、独立于该修复编写的替代修复方案，以及仅应用开发者补丁中一个或几个补丁块的局部修复。诱饵会添加一项大小与开发者补丁大致相当、看起来像修复的检查，但并未保护故障语句，例如将检查放在该语句之后，或检查另一个变量。只有当概念验证在两次运行中仍会因诱饵而触发故障时，该诱饵才算作故障。比较在函数内部成对进行，并以对真实版本或开发者修复的判定为参照；当 f_v 被判定为易受攻击、局部修复被判定为不易受攻击，且中性编辑被判定为易受攻击时，该函数即通过局部修复测试。

L5，执行。检测器接收经构建编译的驱动程序源代码（不含注释，并已解析条件编译）、命令行、输入字节，以及带有行号的函数的一个版本，可以是 f_v、 f_f 或 L4 的一个已确认变体。它回答此次运行是否在该函数内部报告了故障，以及故障发生在哪一行；真实答案则是该次运行已确认的结果。正确预测与针对此次运行的推理一致，但这比理解漏洞的一般情况更为局限。我们报告结果的准确率，以及在已知故障行的故障版本上的联合准确率；后者要求预测出故障，并给出与故障行相差不超过两行的行号；还报告两个真实版本均被正确预测的函数占比，以及在被预测为发生故障的版本中，针对高风险标识符最多的行进行的定位准确率。隐藏函数体对照接收相同的驱动程序、命令行和输入，但只能看到函数的第一行。两个防护规则用于检验一个可见的边界检查是否能够解释该预测。局部规则查看一个版本：当某个高风险行之前十行内没有边界检查时，就预测会发生故障；新增防护规则了解 f_v，并且仅当该版本新增了一个条件语句时，才预测无故障运行。驱动程序和输入的编写目的是触发故障，因此良性输入对照保留该驱动程序，并使用一个使 f_v 无故障运行的输入。

3.2 模型与协议

模型。 开放权重模型包括 Qwen2.5-Coder-7B-Instruct 和 Qwen2.5-Coder-32B- Instruct [24], CodeLlama-34B-Instruct [40], 混合专家模型 Qwen3-Coder-30B-A3B- Instruct [53], 针对漏洞推理进行微调的 VulnLLM-R-7B [34], 采用 8位权重的 Qwen3-Coder-Next、稠密型 Qwen3.6-27B [37], gpt-oss-20b [2], 以及针对安全推理进行微调的 Foundation-Sec-8B-Reasoning [55], , 所有模型均使用固定版本。五个不具备推理模式的模型，其答案取自答案词元。四个推理模型——VulnLLM- R-7B、开启思考模式的 Qwen3.6-27B、采用高推理力度的 gpt-oss-20b，以及 Foundation-Sec-8B-Reasoning——先进行推理，然后按照商业模型的 JSON 格式给出答案，推理预算最高为 32,768 个词元（补充材料第 S25 节

）。归因、提示改述和线性头需要答案位置的分数，因此使用读取答案词元的运行结果；VulnLLM-R-7B 和 Qwen3.6-27B 提供此类结果。商业模型 GPT-6.1-sol 和 GPT-5.6-sol 通过 Codex 命令行客户端调用，Claude Opus 5.5 和 Claude Opus 4.8 则通过 Claude Code 命令行客户端调用；这四个模型均采用高推理力度。每次查询都只包含一个问题，且答案受 JSON 架构约束；模型均无法使用工具、搜索网络或读取本地文件。GPT 客户端在受限文件系统下运行，并禁用了工具和连接器；47,996 份 GPT 对话记录中均不包含工具调用。每条商业模型回复都会核验实际提供回复的模型名称和推理力度。L1 和 L2 涵盖全部十三个模型，引导涵盖其中十一个模型，而证据移除、L4 和 L5 则涵盖四个商业模型。补充材料第 S11 节列出了各实验和模型所用的版本、客户端版本及函数数量。

提示。 检测提示展示一个带有行号的函数，并要求回答“是”或“否”。商业模型和推理模型还会说明置信度。解释提示要求给出判定、作为决策依据的行号以及简短的理由。

3.3 训练截止时间之后发布的通用漏洞披露

截止前集合中的通用漏洞披露日期为2005年至2022年，早于每个训练截止时间，因此，在训练期间接触过代码 [7, 54] 可能是任何结果的一种解释。为此，我们针对仅在商业模型完成训练后才公开的漏洞重复开展实验。供应商声明，Claude Opus 5.的训练截止时间为2026年6月5 [5], 30GPT-6.1-sol为2026年4月 [36], 16 GPT-5.6-sol为2026年2月 [35], Claude Opus 4.为2026年1月8 [4]。我们从 cve.org 的通用漏洞披露列表中抽取了2026年9月1日或之后发布的通用漏洞披露，即在这些截止时间中最晚者之后两个月发布的漏洞；这些漏洞来自开源 C语言和 C++项目，其公开修复会修改一个 C语言或 C++ 函数。截止前集合的确认方法在其中217个漏洞的已标注函数中复现了故障，其中195个通过了第 2.2 节的筛查。这些漏洞来自100个项目，发布于2026年9月3日至2026年11月9日期间，标识符从 CVE-2026-19445 到 CVE-2026-125405 不等；其全部修复提交均于2026年9月1日至2026年11月5日期间提交，晚于每个截止时间。L2和L3使用其中包含所有切片的97个漏洞，而L1则在两组数据上分别报告（补充材料中的表 S10）。

3.4 统计过程

共享同一通用漏洞披露的函数彼此具有依赖性。除非另有说明，区间均为基于 2000 次自助抽样次数、对通用漏洞披露进行重抽样得到的 95% 百分位区间 [17]。当某个单元格中的所有值都相同时，自助法区间会收缩为一个点，此时该单元格给出根据该单元格所对应的通用漏洞披露数量计算得到的 95% 威尔逊得分区间。当某个差异的区间不包含零时，该差异具有显著性。模型之间的比较使用所有参与比较的模型都作答的函数，并在重抽样前于函数内部形成配对差异。在表格中，↑ 表示其区间完全位于零以上的差异，↓ 表示其区间完全位于零以下的差异，≈ 表示其区间包含零的差异。

4 研究问题 1：区分易受攻击函数与已修复函数

开放权重模型。 表 3 报告了成对判别结果。表层特征分类器只能看到长度和句法构造的数量，其 P-C 为 5.8%，MCC 为 0.046。在五个不具备推理模式的开放权重模型中，只有

XX:12 正确的判定，正确的理由?

表 3 L1：在最多 481 个匹配对上进行成对判别，按协议分组的开放权重模型。机会水平是根据各模型判为易受攻击 的判定比例计算的独立猜测P-C。差异以百分点表示，并附95%区间。†按CVE分组的交叉拟合基线。

模型	准确率	精确率	召回率	F1	MCC	AUC	P-C	机会水平	P-C − 机会水平
商业模型									
GPT-6.1-sol	0.666	0.614	0.896	0.729	0.375	0.721	34.7	19.7	15.0 [11.0, 19.3] ↑
GPT-5.6-sol	0.613	0.589	0.753	0.661	0.236	0.624	26.0	23.1	2.9 [−1.2, 7.5] ≈
Claude Opus 5.5	0.828	0.769	0.940	0.846	0.674	0.899	67.2	23.8	43.4 [39.2, 47.4] ↑
Claude Opus 4.8	0.683	0.672	0.715	0.693	0.367	0.626	39.7	24.9	14.8 [10.5, 19.7] ↑
不采用推理模式的开放权重模型									
Qwen2.5-Coder-32B	0.527	0.533	0.439	0.481	0.055	0.518	10.2	24.2	−14.0 [−16.8, −11.0] ↓
CodeLlama-34B	0.500	0.500	0.994	0.665	0.000	0.524	0.0	0.6	−0.6 [−1.4, 0.0] ≈
Qwen3-Coder-30B	0.526	0.541	0.345	0.421	0.056	0.516	8.5	21.7	−13.2 [−15.8, −10.3] ↓
Qwen2.5-Coder-7B	0.509	0.556	0.094	0.160	0.034	0.511	3.1	7.7	−4.6 [−6.7, −2.6] ↓
Qwen3-Coder-Next（8位）	0.493	0.495	0.672	0.570	−0.016	0.520	5.4	21.8	−16.4 [−18.8, −14.0] ↓
开放权重推理模型									
gpt-oss-20b	0.568	0.566	0.580	0.573	0.135	0.511	21.6	25.0	−3.4 [−7.3, 0.8] ≈
VulnLLM-R-7B	0.553	0.557	0.518	0.537	0.106	0.476	24.1	24.9	−0.8 [−4.7, 3.0] ≈
Qwen3.6-27B	0.577	0.562	0.701	0.623	0.159	0.525	23.5	23.5	0.0 [−4.1, 4.0] ≈
Foundation-Sec-8B-Reasoning	0.511	0.507	0.832	0.630	0.030	0.503	16.0	14.7	1.3 [−1.6, 4.2] ≈
表层特征分类器†	0.523	0.526	0.462	0.492	0.046	0.530	5.8	–	–

Qwen2.5-Coder-32B 超过了该分类器，但没有任何模型超过随机水平。较高的 F1值可能掩盖这一点：CodeLlama-34B 几乎将每个版本都标为易受攻击，其 F1值达到 0.665，但 P-C 为 0.0%。四个推理模型都超过了表层分类器，但也没有一个超过随机水平，其差异分别为 −0.8 [−4.7、3.0], 0.0[−4.1、4.0], −3.4 [−7.3、0.8], 和 1.3 [−1.6、4.2] 个百分点，因为它们经常对两个版本给出相同的判定。同一函数的两个版本的得分差异远小于不同函数之间的得分差异，因此单个阈值会将两个版本都赋予相同的标签。

区分样本对的方向。 不过，随机水平掩盖了这些模型中的一个信号。在模型对两个版本给出不同判定的样本对中，9 个开放权重模型里有 5 个显著更常将易受攻击版本判定为易受攻击，而非固定版本（精确 McNemar 检验，经霍尔姆校正）。正确样本对减去反向样本对的数量，Qwen3.6-27B 为 15.4 [10.0、20.7] 个百分点，gpt-oss-20b 为 13.5 [8.4、18.6] 个百分点，VulnLLM-R-7B 为 10.6 [5.0、15.7] 个百分点，Qwen2.5-Coder-32B 和 Qwen3-Coder-30B 分别为 5.4 [1.7、8.9] 个百分点和 5.2 [2.3、8.3] 个百分点；而经校正后，Qwen3-Coder-Next、Foundation-Sec-8B-Reasoning、Qwen2.5-Coder-7B 和 CodeLlama-34B 均未显示显著的方向性。因此，这些模型对修复的敏感性较弱，而且它们习惯于对两个版本给出相同判定，导致其 P-C 处于随机水平（图 3 ）。

商业模型。 商业模型分别比表层分类器高出 28.9 [23.9、34.2] 个百分点（GPT-6.1-sol），20.2 [15.4、25.3] 个百分点（GPT-5.6-sol），61.3 [56.4、66.0] 个百分点（Claude Opus 5.5），以及 33.9 [29.2、39.1] 个百分点（Claude Opus 4.8）；其中三种模型的表现高于随机水平，只有 GPT-5.6-sol 除外。对于这四种模型，在判定不一致的函数对中，判对的数量都远多于判反的数量；判对与判反的数量之差介于 22.7 [18.0、27.6] 个（GPT-5.6-sol）和 65.7 [60.9、70.4] 个（Claude Opus 5.5）之间。GPT 模型在 54.9% 和 49.3% 的函数对中将两个版本都判为易受攻击，其 P-C 值会随函数长度增加而下降（第9节）。Claude Opus 5.5 区分了 67.2% 的函数对；第9节将考察这一领先表现是否源于对已知漏洞的识别。

配对结果。 图3展示了模型未能区分的配对是如何分布的。商业模型很少会颠倒一对样本的判定，因此它们的损失源于对两种版本给出

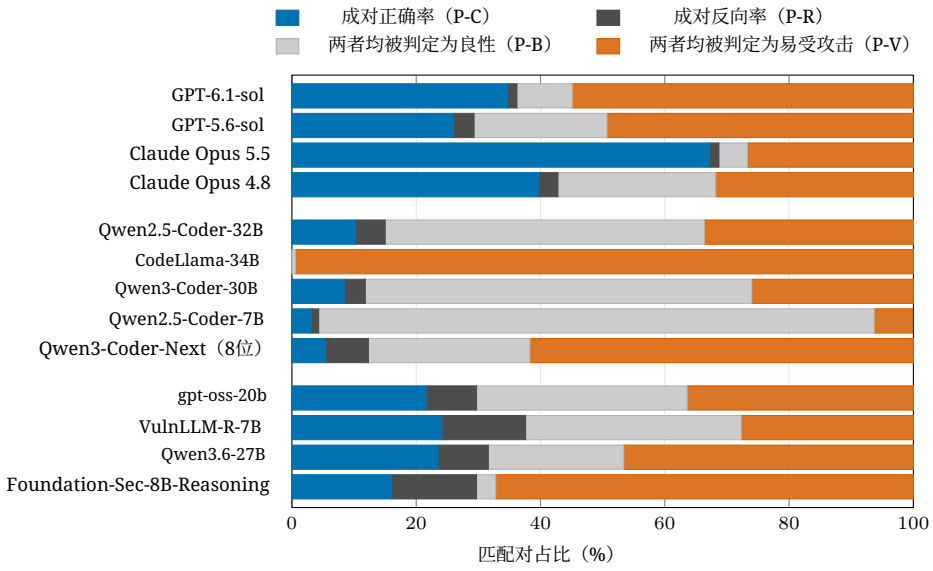


图3 L1 在 PrimeVul [16] 的四个类别中，所有模型均作答的匹配对的结果。当易受攻击版本被判定为易受攻击，且修复版本被判定为不易受攻击时，该匹配对判为正确（P-C）；反之则判为相反（P-R）。模型分组方式与表 3 相同。

相同的判定，且几乎总是判定为 易受攻击。对于 GPT 模型而言，这类配对数量超过了判定正确的配对；而 Claude Opus 5.5 将 26.8% 的配对中的两个版本都判定为易受攻击。开放权重模型的失败方向恰好相反。对于大多数配对，Qwen2.5-Coder-7B 将两个版本都判定为不易受攻击，CodeLlama-34B 和 Foundation-Sec-8B-Reasoning 将两个版本都判定为易受攻击，而推理模型颠倒的配对数量超过了任何商业模型。因此，检测器可能以两种相反的方式丢失一个配对，只有配对级视角才能显示具体是哪一种。

提示与隐藏状态。 在两个改写模板下（补充材料的表 S1），Qwen2.5-Coder-32B 的总体 P-C 保持稳定；然而，只有 2.3% 的匹配对在全部三个模板下均能区分，而 17.7% 的匹配对至少在一个模板下能够区分，这正是弱信号周围存在噪声的特征。对答案位置的隐藏状态使用按函数交叉拟合的线性头，只将 Qwen2.5-Coder-32B 的曲线下面积从 0.518 提高到 0.595（表 S2）。因此，隐藏状态携带的排序信号较弱，无法用于实际决策。

研究问题 1 的答案。没有任何模型能可靠地区分易受攻击函数及其筛选后的修复版本。开放权重模型尽管对修复有微弱的敏感性，表现仍处于机会水平；而商业模型中有三个模型的表现高于机会水平，但仍将许多修复版本标记为易受攻击。

5 RQ2：证据是否指向故障?

显式证据。 表 4 报告了引用行包含触发器行的频率。触发器行通常是明显看似有风险的语句，因为 61.1% 的触发器行包含至少一个关键词，而关键词规则命中它的频率约为随机行的两倍。四个商业模型引用四到五行，超出关键词规则 30.5 至 52.9 个百分点，因此它们的定位并不局限于选择看似有风险的语句。

XX:14 正确的判定，正确的理由?

表 4 L2，引用行包含触发器行的频率（占函数的百分比），分别针对模型以及两种对每个函数选取相同数量行的规则：关键词规则（包含高风险标识符的行，例如 `memcpy` first，然后是随机行）和随机行。每个模型均统计其回答的所有函数。最后两列：模型与关键词规则之间的差值（以百分点计）及其95%区间，以及第3.4节中的箭头。

模型	引用者模型	关键词 rule	随机	模型减关键词规则	
商业模型					
GPT-6.1-sol	67.2	27.2	13.6	40.0 [35.1，44.8]	↑
GPT-5.6-sol	58.6	25.7	13.2	32.9 [28.3，37.4]	↑
Claude Opus 5.5	81.3	28.4	14.2	52.9 [48.3、57.4]	↑
Claude Opus 4.8	55.9	25.4	12.4	30.5 [25.3，35.6]	↑
不具备推理模式的开放权重模型					
Qwen2.5-Coder-32B	11.9	6.6	2.4	5.2 [2.6、7.6]	↑
CodeLlama-34B	37.7	38.2	26.4	−0.5 [−5.0、4.2]	≈
Qwen3-Coder-30B	8.3	4.8	1.9	3.5 [1.5，5.7]	↑
Qwen2.5-Coder-7B	9.0	8.3	3.2	0.7 [−1.8，3.5]	≈
Qwen3-Coder-Next（8位）	21.6	12.2	5.5	9.5 [5.9, 12.7]	↑
开放权重推理模型					
gpt-oss-20b	39.7	22.8	11.7	16.9 [12.2，21.3]	↑
VulnLLM-R-7B	39.1	36.5	25.3	2.6 [−2.1，7.0]	≈
Qwen3.6-27B	38.3	17.3	8.9	21.0 [16.7，25.3]	↑
Foundation-Sec-8B-Reasoning	22.0	20.1	10.6	1.9 [−1.5、5.7]	≈

在限定于触发器行出现在故障报告中的 342 个函数后，对于 GPT-6.1-sol、GPT-5.6-sol、Claude Opus 5.5 和 Claude Opus 4.8，其优势幅度分别为 45.7 [40.4、51.4], 38.2 [32.7、43.7], 55.7 [50.6、61.0]，和 33.3 [27.1、39.7] 个百分点；对于基于补丁的触发器行，这些优势仍为正（补充材料中的表 S6）。在开放权重模型中，Qwen3.6-27B 和 gpt-oss-20b 分别超出该规则 21.0 [16.7、25.3] 和 16.9 [12.2、21.3] 个百分点，是两个最大优势幅度；而 VulnLLM-R-7B、Foundation-Sec-8B-Reasoning、Qwen2.5-Coder-7B 和 CodeLlama-34B 与该规则没有显著差异。较弱的开放权重模型在两方面表现不佳。CodeLlama-34B 和 VulnLLM-R-7B 每个函数引用约九行，这提高了其中包含触发器行的机会；而两个 Qwen2.5-Coder 模型和 Qwen3-Coder-30B 每个函数引用约一行，并且经常返回空列表。

与切片的一致性。在九个模型解释的 376 个函数中（表 5），商业模型与每个参考基准的行 Jaccard 均为随机期望的两到五倍。在函数内进行配对后可以发现，它们引用的行与动态切片的匹配程度高于 Joern 和 DeepWukong 切片，并且与 dg 切片大致相当；dg 切片是保留了动态切片大部分内容的静态切片。对于 GPT-6.1-sol，其与动态切片的行 Jaccard 比与 Joern 的高 0.043 [0.026、0.059]，而与 DeepWukong 的差值高 0.071 [0.054、0.087], ；与 dg 的差异为 −0.004 [−0.019、0.011], ，并且其与动态切片的行 Jaccard 达到 0.299，与 DeepWukong 切片的行 Jaccard 达到 0.228。绝对值仍处于中等水平（最高为 0.315），因为模型引用的行数约为四行，而动态切片包含的行数更多。

表 5 还展示了参考基准如何影响结论。与动态切片相比，每个模型在 Deep Wukong 切片上的行 Jaccard 都更低，因为该切片包含许多不属于故障依赖关系的行。在大规模 dg 切片和

表 5 L2，针对九个模型（Qwen3-Coder-Next、Qwen3.6-27B、gpt-oss-20b 和 Foundation-Sec-8B-Reasoning 除外）所解释的 376 个函数，给出了每个参考基准的显式证据。引用触发器行：引用行包含触发器行的函数占比（%）。行 Jaccard：公式（1）的 $J(E, T)$，括号内为随机选取同等数量行时的期望值。粗体表示显著高于该期望值。并集：动态切片、Joern 的静态切片或 dg 的静态切片中的行。

		行 Jaccard 与					
模型	触发器行引用	触发 line	动态切片	Joern 切片	dg 切片	DeepWukong 切片	并集
商业模型							
GPT-6.1-sol	**70.2** (16.8)	**0.211**(0.046)	**0.299**(0.096)	**0.256** (0.098)	**0.303** (0.111)	**0.228** (0.109)	**0.273** (0.121)
GPT-5.6-sol	**66.0** (16.4)	**0.213**(0.046)	**0.276**(0.091)	**0.242** (0.095)	**0.284** (0.105)	**0.218** (0.104)	**0.260** (0.116)
Claude Opus 5.5	**84.0** (17.3)	**0.226**(0.046)	**0.315**(0.097)	**0.274** (0.099)	**0.320**(0.114)	**0.241** (0.111)	**0.293**(0.124)
Claude Opus 4.8	**60.9** (15.2)	**0.196**(0.046)	**0.278**(0.091)	**0.240** (0.094)	**0.283**(0.104)	**0.212** (0.103)	**0.249**(0.112)
不使用推理模式的开放权重模型							
Qwen2.5-Coder-32B	**11.7** (2.9)	**0.072** (0.016)	**0.059**(0.023)	**0.050** (0.023)	**0.058** (0.024)	**0.051** (0.024)	**0.052**(0.025)
CodeLlama-34B	**38.3** (26.1)	**0.099** (0.040)	**0.113**(0.093)	**0.116** (0.101)	**0.163** (0.125)	0.099 (0.113)	**0.155** (0.137)
Qwen3-Coder-30B	**9.6** (2.4)	**0.031**(0.007)	**0.037**(0.015)	**0.038** (0.016)	**0.040** (0.016)	**0.033** (0.017)	**0.038** (0.018)
Qwen2.5-Coder-7B	**10.4** (3.7)	**0.077** (0.027)	**0.062**(0.032)	**0.052** (0.033)	**0.065** (0.034)	**0.052** (0.033)	**0.057**(0.034)
Qwen3-Coder-Next（8位）	**21.3** (6.1)	**0.069**(0.019)	**0.094** (0.039)	**0.088** (0.040)	**0.103** (0.045)	**0.082** (0.045)	**0.098** (0.048)
开放权重推理模型							
gpt-oss-20b	**44.1** (14.3)	**0.158**(0.043)	**0.235** (0.082)	**0.209** (0.085)	**0.245**(0.097)	**0.191** (0.095)	**0.227**(0.105)
VulnLLM-R-7B	**42.0**(27.1)	**0.094**(0.045)	**0.158**(0.095)	**0.145** (0.102)	**0.200** (0.126)	**0.133** (0.117)	**0.191** (0.141)
Qwen3.6-27B	**42.3** (11.1)	**0.176**(0.041)	**0.207**(0.068)	**0.174** (0.072)	**0.200** (0.074)	**0.161** (0.077)	**0.184** (0.082)
Foundation-Sec-8B-Reasoning	**24.2** (12.5)	**0.106** (0.043)	**0.131** (0.073)	**0.117** (0.075)	**0.148** (0.084)	**0.107** (0.083)	**0.136**(0.090)

并集上，引用许多行的模型 CodeLlama-34B 和 VulnLLM-R-7B 获益最大；而商业模型在 dg 切片和动态切片上的结果大致相同。因此，大规模静态参考会奖励引用许多行，而动态切片则消除了这种奖励。

归因证据。 开放权重模型的归因方法将触发器行排在首位的函数占比为3%至9%，而机会水平为4%；按风险关键词进行排名时，这一比例为10.3%至11.0%，并且19个归因排名中有17个显著低于机会水平。最佳排名是VulnLLM-R-7B的留一法，其与动态切片的行Jaccard达到关键词排名的水平（0.207 对 0.194）；Qwen模型的留一法与动态切片的重叠显著低于 机会水平，这表明，移除后使其得分变化最大的行往往位于故障依赖关系之外。在预算等于切片大小的情况下，模型、方法和切片的76种组合中，有36种显著高于随机期望，15种低于随机期望；而归因方法遗漏了动态切片的70.2%至88.1%。更大的预算并未改变这一结论（补充材料中的表S3和表S13及图S1）。

对研究问题2的回答。 商业模型和两个开放权重推理模型能够超越关键词捷径定位故障，而开放权重模型的内部归因最多只能达到这一水平。

6 研究问题3：判定是否取决于证据?

如果正确判定源于正确的理由，那么模型据以区分这对样本的函数，应当也是其定位故障所在的函数。对于 GPT-6.1-sol，在正确区分的样本对中，有引用触发器行的函数占比为 77.8 [71.1、83.8] percent；在其他样本对中则为 61.5 [56.5、66.7] percent。所有商业模型均呈现相同的排序，只有 Claude Opus 5.5 的区间存在重叠。干预实验直接检验了这种耦合关系。

证据移除。 相较于屏蔽数量相同的随机行，屏蔽动态

XX：16 正确的判定，正确的理由?

表6 L3，四种商业模型的证据移除。将易受攻击版本中的行置空，并与未更改的修复版本配对。动态切片条件下，将随机行逐步置空；其他条件下则使用未更改的版本对。表中给出了 P-C（百分比）、P-C差异（百分点）及其95%区间。补充材料的表S16还列出了置信度排序。

（*a*）*GPT* 模型	GPT-6.1-sol		GPT-5.6-sol	
条件	P-C	P-C差异	P-C	P-C差异
未屏蔽任何内容	34.7	–	26.0	–
屏蔽动态切片	27.0	−8.7 [−12.1, −5.4] ↓	23.7	−9.8 [−13.4, −6.3] ↓
屏蔽DeepWukong切片	23.3	−11.4 [−14.9, −8.0] ↓	18.3	−7.7 [−11.0, −4.3] ↓
触发器行已屏蔽	30.1	−4.6 [−7.5, −1.7] ↓	28.3	2.3 [−1.1,5.6] ≈
随机行置空	35.8	1.0 [−1.7,3.9] ≈	33.5	7.5 [4.4,10.7] ↑
仅保留动态切片	27.4	−7.3 [−10.0, −4.4] ↓	23.1	−2.9 [−6.4, 0.4] ≈
***(b) Claude*模型**	**Claude Opus 5.5**		**Claude Opus 4.8**	
条件	P-C	P-C差异	P-C	P-C差异
未屏蔽任何内容	67.2	–	39.7	–
屏蔽动态切片	65.7	0.2 [−2.5, 2.9] ≈	46.2	−4.6 [−8.4, −0.8] ↓
屏蔽DeepWukong切片	62.8	−4.4 [−7.2, −1.6] ↓	36.8	−2.9 [−7.4, 1.5] ≈
触发器行已屏蔽	67.4	0.2 [−1.9, 2.3] ≈	47.6	7.9 [3.6, 11.8] ↑
随机行已隐藏	65.5	−1.7 [−3.9, 0.4] ≈	50.7	11.0 [7.6, 14.5] ↑
仅保留动态切片	66.3	−0.8 [−3.4, 1.7] ≈	36.6	−3.1 [−8.4, 1.9] ≈

切片使 GPT-6.1-sol 的 P-C 变化 −8.7 [−12.1、 −5.4] 个百分点，使 GPT-5.6-sol 的 P-C 变化 −9.8 [−13.4、 −6.3]，个百分点，并使 Claude Opus 4.8 的 P-C 变化 −4.6 [−8.4、 −0.8] 个百分点（表6）。随机对照很重要，因为仅置空本身就会改变判定。相对于未改变的函数，随机置空使 GPT-5.6-sol 的 P-C 提高 7.5 [4.4、10.7] 个百分点，并使 Claude Opus 4.8 的 P-C 从 39.7% 提高到 50.7%。Claude Opus 5.5 的表现有所不同。屏蔽动态切片后，它仍将 93.1% 的易受攻击版本标记为易受攻击，而未改变时为 94.0%；其 P-C 变化了 0.2 [−2.5、2.9] 个百分点。它的置信度比判定受到的影响更大。在未改变的配对中，89.4% 的情况下易受攻击版本获得更高的置信度；随机置空后，这一比例为 84.0%；屏蔽动态切片后为 82.1%。对于其未命名 CVE 的函数，判定同样持续存在（90.1%，而对于已命名的通用漏洞披露为 93.5%）。对于在其训练之后发布的通用漏洞披露，同样的移除操作会降低其 P-C（第9节）。

引导。 表7报告了引导条件。标记切片并集，相较于标记同样多的随机行，使 GPT-6.1-sol 的 P-C 提高了 4.2 [0.8、7.8] 个百分点；其他三种商业模型的提升幅度均包含零。与不标记的对照组相比，标记触发器行使 GPT-6.1-sol 的 P-C 提高了 6.4 [2.9、10.1]，使 GPT-5.6-sol 的 P-C 提高了 7.5[3.6、11.7] 个百分点；而标记随机行带来的变化分别为 −2.7 [−5.7、0.2] 和 −1.2 [−4.3、2.1] 个百分点。因此，提升来自该行本身，而非标记这一行为。标记动态切片后，每种商业模型的 P-C 变化区间都包含零。将依赖边作为文本附加进去有助于 GPT-6.1-sol（比打乱依赖边高 4.6 [0.7、8.7] 个百分点）。在开放权重模型中，标记触发器行或动态切片仅对 Qwen2.5-Coder-7B 有显著帮助，其 P-C 从接近零开始（补充材料中的表S4和S14）。

研究问题3的答案。判定在一定程度上取决于证据。移除动态切片会降低三个商业模型的成对判别能力，而在截止时间之前，Claude Opus 5.5 的成对判别能力则保持不变；提供证据帮助不大。

表7 L3，标记提示词中的行。分别报告不加标记、标记触发器行以及标记随机行时的 P-C（%），随机行数量与切片并集中的行数相同。随后给出 P-C差异（以百分点计）及其 95%区间：触发器行和动态切片相对于不加标记的结果，以及切片并集相对于等规模随机标记的结果。补充材料中的表S4给出了全部八种条件。

模型	无标记	触发器行 已标记	随机行 标记的	触发器行 减去未标记项	动态切片 减去无标记	并集 减去随机项
商业模型						
GPT-6.1-sol	33.5	39.9	30.8	6.4 [2.9、10.1] ↑	3.3 [−0.2，6.8] ≈	4.2 [0.8，7.8] ↑
GPT-5.6-sol	26.2	33.7	24.9	7.5 [3.6、11.7] ↑	4.2 [0.0，8.0] ≈	2.9 [−0.8、6.7] ≈
Claude Opus 5.5	68.2	71.7	65.7	3.5 [−0.2,7.2] ≈	2.3 [−1.4，5.9] ≈	0.6 [−3.5, 4.7] ≈
Claude Opus 4.8	37.0	41.2	35.8	4.2 [−0.2,8.3] ≈	1.5 [−2.7，5.5] ≈	1.5 [−2.6，5.5] ≈
无推理模式的开放权重模型						
Qwen2.5-Coder-32B	10.2	10.0	6.7	−0.2 [−3.2，3.2] ≈	−1.2 [−4.5，2.2] ≈	0.4 [−2.3, 3.0] ≈
CodeLlama-34B	0.0	0.2	0.0	0.2 [0.0,0.6] ≈	0.0 [0.0、0.0] ≈	0.2 [0.0，0.6] ≈
Qwen2.5-Coder-7B	3.1	11.9	8.9	8.7 [5.5,12.1] ↑	5.6 [2.6, 8.5] ↑	−0.4 [−3.6, 2.7] ≈
开放权重推理模型						
gpt-oss-20b	23.9	22.5	20.4	−1.5 [−5.5, 2.8] ≈	−0.2 [−5.0, 4.2] ≈	1.9 [−2.1, 5.8] ≈
VulnLLM-R-7B	25.8	28.1	22.5	2.3 [−3.2,7.8] ≈	−1.2 [−6.7, 3.8] ≈	0.4 [−4.3、5.2] ≈
Qwen3.6-27B	20.8	16.8	14.8	−4.0 [−8.2, 0.0] ≈	−2.9 [−7.5, 1.7] ≈	2.1 [−1.6、5.8] ≈
Foundation-Sec-8B-Reasoning	16.2	12.5	12.9	−3.7 [−8.2, 0.8] ≈	−4.6 [−9.1, 0.0] ≈	−2.5 [−6.6, 1.5] ≈

7 研究问题4：对执行确认的编辑作出的判定

L4 和 L5 需要为每个函数准备多个变体，并通过执行确定其标签。变体会替换源代码树私有副本中的函数，使用恢复出的构建命令重新构建程序，并运行两次概念验证。如果预言机在两次运行中都触发，则该变体标记为故障；如果两次运行都完成且没有任何报告，则标记为正常；其他结果则予以舍弃。GPT-6.1-sol 根据易受攻击函数、开发者的补丁以及观测到的故障类型和行，为两组函数编写替代修复方案，并为截止前集合编写诱饵；Claude Opus 5.5 则为截止时间之后的通用漏洞披露编写诱饵（补充材料的 S22 节）。

基准。 候选项必须具有固定文本、函数单独存放于一个文件中的构建版本、行级预言机，且不得超过 150 行。在优先选择不同通用漏洞披露的前提下选出的 150 个候选项中，有 109 个函数通过了基线检查和第 2.2 节的筛选。在这些函数上，执行确认了 315 个变体，其中包括 92 个替代修复、20 个仍然存在故障的诱饵，以及 12 个消除了故障的诱饵；96 个函数的两个重命名版本均得到确认。因此，GPT-6.1-sol 评估的是它自己生成的编辑，而其他模型都没有参与生成这些编辑。

重命名与修复。 重命名几乎没有影响（表 8b）。四个模型对重命名的易受攻击版本维持原判的函数比例分别为 95.8%、96.9%、96.9% 和 87.5%，因此，即使标识符名称和逐字文本不复存在，判定依然成立；重命名对上的 P-C 与原始对相比也仅相差几个百分点。修复则使模型表现各异（表 8c）。在它们判定易受攻击版本存在漏洞的函数中，GPT 模型仅接受 42.7% 和 32.5% 的开发者修复，而接受替代修复的比例稍高；这种差异仅在 GPT-5.6-sol 中具有显著性，而它并未参与编写这些修复。Claude Opus 5.5 对两类修复的接受程度相当，分别为 83.3% 和 81.1%；Claude Opus 4.8 则介于两者之间，分别为 58.1% 和 63.5%。

局部修复。 切片工具的修复干预阶段会将开发者的补丁拆分为多个补丁块，并构建应用了一个补丁块或最小的一组两个或三个补丁块的 f_v 版本。如果触发输入在 f_v 上引发故障、但在该变体上不再引发故障，那么该变体就成为一个局部修复

XX:18 正确的判定，正确的理由?

表8 L4：模型对每个函数的经执行确认的变体所作的判定。每行列出展示给模型的版本及该版本对应的正确判定；每个单元格给出模型作出该判定的函数占比及其95%区间。在(b)中，当判定与未重命名版本上的判定一致时，该判定即为正确。标记为 Δ的行给出同一批函数中其上方两行之间的差值，单位为百分点。该基准中的每个函数都来自不同的通用漏洞披露，因此相同的占比对应相同的区间。

展示给模型的版本	正确判定	GPT-6.1-sol	GPT-5.6-sol	ClaudeOpus 5.5	Claude Opus 4.8
(*a*) 真实版本，逐一单独展示所有函数					
易受攻击版本	易受攻击	94.5 [89.9, 98.2]	81.7 [74.3, 88.1]	97.2 [93.6, 100.0]	76.1 [67.9, 83.5]
修复版本	不易受攻击	46.8 [37.6, 56.9]	40.4 [31.2, 50.5]	80.7 [73.4, 88.1]	67.0 [57.8, 75.3]
两个版本（P-C）	两者均正确	42.2 [33.0, 51.4]	25.7 [17.4, 33.9]	78.9 [71.6, 86.2]	44.0 [34.9, 53.2]
(b) 重命名版本，两个版本中的函数均已重命名					
重命名的易受攻击版本	与原始版本相同：95.8 [91.7、99.0]		96.9 [92.7、100.0]	96.9 [92.7、100.0]	87.5 [80.2、93.8]
重命名的已修复版本	与原始版本 89.6 [83.3、94.8] 中的相同		89.6 [83.3、95.8]	89.6 [83.3、94.8]	83.3 [76.0、90.6]
原始配对（P-C）	两者均正确	41.7 [32.3, 51.0]	25.0 [16.7, 33.3]	82.3 [74.0, 89.6]	45.8 [35.4, 55.2]
重命名对（P-C）	两者都正确	38.5 [29.2, 47.9]	26.0 [17.7, 35.4]	81.2 [72.9, 88.5]	46.9 [36.5, 56.2]
(c) 修复（每个单独展示）其易受攻击版本被判定为易受攻击的函数					
开发者的修复	不易受攻击	42.7 [32.6, 52.8]	32.5 [22.1, 42.9]	83.3 [75.6, 91.1]	58.1 [45.9, 68.9]
替代修复方案	不易受攻击	52.8 [41.6, 62.9]	45.5 [33.8, 57.1]	81.1 [72.2, 88.9]	63.5 [51.4, 74.3]
Δ 替代方案减去开发者的修复		10.1 [−1.1, 21.3]	13.0 [3.9、23.4]	−2.2 [−11.1, 6.7]	5.4 [−5.4, 16.2]

每次运行两次时，gdb 显示更改的行在触发输入下执行；执行该函数的对照输入表现与 f_v 一致；而插入 `volatile intvv_neutral = 0;` 到 f_v 的中性编辑仍会导致故障。本研究的局部修复涵盖 205 个函数，这些函数的选择独立于 L4 和 L5 中的函数。在易受攻击版本被判定为易受攻击的函数中，GPT-6.1-sol 对 35.1% 的函数接受局部修复，GPT-5.6-sol 对 33.7% 的函数接受，Claude Opus 5.5 对 65.3% 的函数接受，Claude Opus 4.8 对 48.7% 的函数接受（补充材料表 S15）；除 Claude Opus 4.8（87.3%）外，所有模型对超过 90% 的函数都不会因中性编辑而改变判定。局部修复已在触发输入和对照输入上得到确认，但这并不能保证对每个输入都安全，因此，对于其他输入，将其判定为易受攻击是有理由的。在这些变体中接受评估的唯一开放权重模型 Qwen2.5-Coder-7B，仅将 12.2% 的易受攻击版本判定为易受攻击，因此很少有判定需要修订。

研究问题4的答案。 判定对重命名和中性编辑具有稳健性，但各模型对经执行确认的修复的接受程度差异很大。

8 研究问题5：预测一次执行

▶ **示例3** （L4处有一个函数，L5处也有一个函数）。 在图4中，leptonica 的 `findNextBorderPixel`（CVE-2020-36278）根据两个偏移表计算出的八个邻居进行搜索。驱动程序提供图像大小、起始像素和搜索方向。在角点输入下，第一个邻居位于图像范围之外，AddressSanitizer 报告第20行读取操作发生堆缓冲区溢出。开发者的修复方案在读取前检查两个坐标（图4中的灰色行），并通过了额外的动态筛查。替代修复方案仅读取范围内的数据。诱饵在读取后应用开发者的检查，因此故障仍然存在。四个模型均在L4处正确判定全部六个版本，并在L5处预测出每个结果和故障行。

结果预测。 表 9 报告了对 109 个函数的 533 个版本所作的预测。四个模型对其中 86.3% 至 94.6% 的版本正确预测了结果，MCC 值分别为 0.859、0.826、0.891 和 0.734；而对每个版本都回答故障 的准确率为

`findNextBorderPixel`，leptonica，CVE-2020-36278。第 20 行发生堆缓冲区溢出（读取）。输入描述了一幅 8×8 图像，并从其角像素（0, 0）开始搜索。

```
 1  findNextBorderPixel(l_int32 w, l_int32 h, l_uint32 *data, l_int32 wpl,
        第 2 行到第 13 行获取起始像素（px, py）和三个输出指针，并声明局部变量
14      qpos = *pqpos;
15      for (i = 1; i < 8; i++) {
16          pos = (qpos + i) % 8;
17          npx = px + xpostab[pos];
18          npy = py + ypostab[pos];
            if (npx < 0 || npx >= w || npy < 0 || npy >= h)    修复版本的第 19 行
                continue;    修复版本的第 20 行
19          line = data + npy * wpl;
20          val = GET_DATA_BIT(line, npx);
            第 21 行至第 30 行在其位被设置时返回邻居，否则返回 1
```

版本	更改	Run	L5	L4 判定（仅函数） GPT-6.1-sol	GPT-5.6-sol	Claude Opus 5.5	Claude Opus 4.8
易受攻击	原始	故障(20)	故障（20）	V	V	V	V
开发者的修复	第19行之前的守卫条件	干净	干净	N	N	N	N
替代修复方案	仅读取边界内数据	干净	干净	N	N	N	N
诱饵	第 20 行后的守卫条件	故障（20）	故障（20）	V	V	V	V
重命名的漏洞	重命名标识符	故障（20）	故障（20）	V	V	V	V
重命名的修复	重命名标识符	干净	清理	N	N	N	N

图4 一个函数的六个反事实变体。Run 给出测得的结果，L5 给出结合驱动程序和输入得出的全体一致预测。括号中给出故障行。灰色行是开发者修复所插入的守卫条件，并在修复版本中标有其编号。其他所有行号均指易受攻击版本。在 L4 中，V 和 N 分别表示易受攻击和不易受攻击。

42.2%。示例3展示了其中一个函数：模型对该函数的每个版本都作出了预测，包括检查位于故障读取之后的诱饵版本。它们的错误出现在易受攻击版本上：对于这些版本，它们在 67.0% 至 87.2% 的函数中预测会发生故障，而修复版本和替代修复几乎总是被预测为无故障。在已知故障行的故障版本中，同时正确预测故障及其位置的准确率介于 69.3% 和 86.1% 之间；当 GPT-6.1-sol 预测存在故障时，在 97.0% 的版本中，它所指出的行与故障行相差不超过两行，而词元风险最高的行仅为 26.6%。

对照防护规则。 局部防护规则达到 47.1%，因为它将 82.7% 的版本预测为存在故障。新增防护规则了解 f_v，是“可见检查决定结果”这一解释的最强形式，其准确率达到 80.3%。在相同版本上进行配对比较时，四个模型分别超过它 12.6 [7.5、17.7]，10.9 [6.2、15.7]，14.3 [9.4、19.2]，和 6.0 [1.1、10.7] 个百分点。大部分差异来自开发者修复。许多修复会修改现有条件或边界，却不添加条件语句，因此该规则只能将其中 67.0% 预测为无故障，而模型预测至少有 98.2% 为无故障。连同它们的重命名副本，这些修复贡献了 12.0 至 12.2 个百分点，替代修复贡献了 3.2 至 3.4 个百分点，而仍然会产生故障的诱饵贡献了 2.1 至 3.0 个百分点；模型在其中 70.0% 至 95.0% 的案例中识别出这些诱饵，而该规则的识别率为 15.0%。易受攻击版本使这一差异变化了 -11.8 至 -4.7 个百分点，因为该规则始终预测这些版本会发生故障（补充材料中的表S9）。结合具体输入来看，模型因此接受开发者的修复，而 GPT 模型在 L1 和 L4 拒绝该修复。

良性输入。 我们对 60 个函数使用良性输入运行相同的驱动程序，其中 14 个函数使用缩短版的触发输入，另外 46 个函数使用 GPT-6.1-sol 仅更改了攻击值的输入，修改位置在输入文件或驱动程序中。当

XX：20 正确的判定，正确的理由?

表 9 L5，单次执行的预测。总体行汇总了所有经过评估且结果已确认的版本。估算值以方括号标出 95%区间。隐藏函数体对照使用相同的驱动程序和输入，但只能看到函数的第一行。良性输入对照保留驱动程序，并使用一种能使 f_v 运行正常的输入。两个真实版本 要求一对版本的两项预测都正确。联合度量要求预测到故障，且预测行与观测到的故障位置相差不超过两行。按版本类型划分的各行显示了错误的分布情况。

指标	GPT-6.1-sol	GPT-5.6-sol	Claude Opus 5.5	Claude Opus 4.8
所有版本	92.9 [89.7、95.7]	91.2 [88.1、94.0]	94.6 [91.7, 97.0]	86.3 [82.4，90.0]
隐藏函数体（捷径）	51.6 [49.6，53.6]	49.0 [47.0，51.0]	46.5 [44.6，48.4]	51.8 [49.9，53.9]
答案故障适用于每个版本（捷径）	42.2	42.2	42.2	42.2
易受攻击版本	84.4 [77.1、90.8]	78.9 [70.6，86.2]	87.2 [80.7，92.7]	67.0 [57.8, 75.2]
修复版本	99.1 [97.2，100.0]	98.2 [95.4，100.0]	98.2 [95.4，100.0]	100.0 [96.6，100.0]
函数的两个真实版本	83.5 [76.1，89.9]	77.1 [68.8，84.4]	85.3 [78.0, 91.7]	67.0 [57.8, 75.2]
替代修复方案	100.0 [96.0, 100.0]	100.0 [96.0, 100.0]	100.0 [96.0, 100.0]	98.9 [96.7, 100.0]
仍然出错的诱饵	85.0 [70.0，100.0]	90.0 [75.0，100.0]	95.0 [85.0，100.0]	70.0 [50.0，90.0]
移除了故障的诱饵	100.0 [75.8，100.0]	100.0 [75.8，100.0]	100.0 [75.8，100.0]	100.0 [75.8，100.0]
重命名保留答案	99.5 [98.4，100.0]	96.3 [93.2，98.4]	98.9 [97.4，100.0]	96.8 [94.2，98.9]
预测到故障，且行在两行之内（所有故障版本）	81.2 [73.1，88.8]	77.2 [69.2，85.0]	86.1 [79.1，92.4]	69.3 [59.9，78.2]
行内两行范围内，预测会发生故障的版本	97.0 [92.4，100.0]	96.9 [91.9，100.0]	98.9 [96.5，100.0]	100.0 [94.9，100.0]
随机行（捷径）	16.4	15.9	15.9	14.0
含有最多高风险词元的行（捷径）	26.6 [17.1，36.1]	26.7 [17.3，36.3]	27.3 [17.9，36.4]	29.3 [19.6，40.1]

执行表明，它会到达 f_v 中的函数，对其中大多数函数而言，甚至会到达故障语句本身；而且 f_v 和 f_f 在该函数上的运行均无故障（补充材料的 S22节）。对于这些函数，模型预测 f_v 在良性输入上无故障运行的比例为 98.3% 至 100%；对于相同的 f_v，模型预测其在攻击输入上发生故障、在良性输入上无故障运行的比例为 68.3% 至 88.3%，因此预测结果会随输入而变化。

对研究问题5的答案。 在考虑驱动程序和输入后，商业模型能够预测运行是否会在防护规则之外发生故障，这主要是因为它们预测开发者修复后运行无故障，而且预测结果会随输入而变化。

9 记忆与其他替代解释

目前的结果与部分理解相符。本节考察无需部分理解也能产生相同观察结果的解释：首先是对已知漏洞的识别，然后是对实验设置的稳健性。

识别。 识别探测会向每个商业模型仅展示易受攻击函数，并要求其指出该函数所属的项目及其通用漏洞披露编号。对于截止时间之前的函数，Claude Opus 5.5 有 68.5% 能说出正确的通用漏洞披露编号，GPT-6.1-sol 和 GPT-5.6-sol 分别为 20.0% 和 15.3%，Claude Opus 4.8 为 9.9 %。四个模型都能为大多数函数说出正确的项目，其中 GPT-6.1-sol 的准确率为 95.9%；而且，当它们说出通用漏洞披露编号时，四个模型对更多函数对的判别能力都会提高：GPT-6.1-sol 为 41.7 [32.0、52.1]%，而仅说出项目时为 32.9 [27.5、38.3]%；Claude Opus 5.5 为 72.1 [67.1、77.0]%，而仅说出项目时为 56.8 [48.6、65.1]%。对于 GPT 模型而言，识别仅涵盖少部分函数，因此其大部分判别能力来自那些它们未能说出通用漏洞披露编号的函数。Claude Opus 5.5 的 L5预测在识别成功与否的情况下准确率相同（差异为 -1.4 [-6.4，4.2] 个百分点）。

训练截止时间之后。 图 5 比较了截止时间前后的各个水平。四个模型都没有为 195 个截止后函数中的任何一个准确指出其通用漏洞披露编号，但对其中大多数函数都指出了正确的项目。定位能力依然存在。四个模型引用触发器行的频率都比关键词规则高 34.2 至 46.7 个百分点，并且其引用行仍然比 DeepWukong 切片更接近动态切片。对证据的依赖同样持续存在。将动态切片置空会降低 GPT-6.1-sol 的 P-C，如今也会降低 Claude

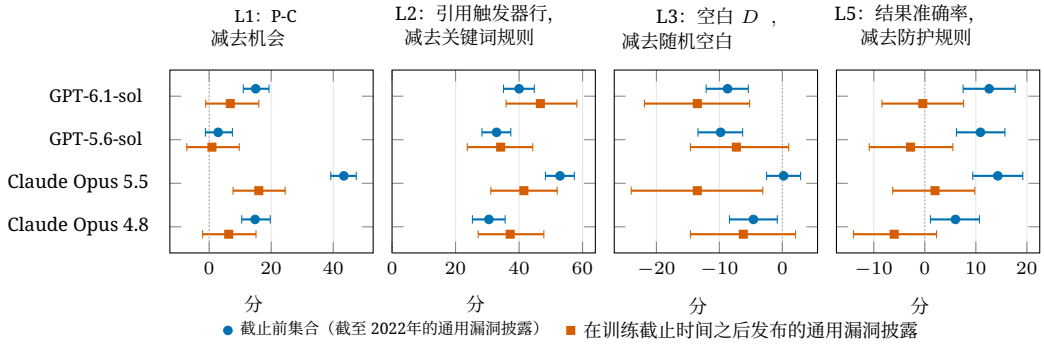


图 5 四种商业模型相对于各级基线的差值，以百分点表示，并给出95%区间；数据涵盖截止前集合以及在其训练截止时间之后发布的通用漏洞披露。L1：P-C 减去模型的随机水平。L2：引用行包含触发器行的函数占比，减去关键词规则对应的占比。L3：将动态切片 D 置空所导致的 P-C 变化，减去将同等数量的随机行置空所导致的变化。L5：结果准确率减去新增防护规则的结果准确率，并按相同版本配对。Claude Opus 5.5 的 L1 差值从 43.4[39.2、47.4] 降至 16.0 [7.7、24.5] 个百分点；截止时间之前，L5 差值为 6.0 至 14.3 个百分点，之后则降至 −6.0 至 2.0 个百分点。补充材料中的表 S8 列出了所有度量指标。

Opus 5.5 相对于随机置空的 P-C，分别降低 −13.5 [−21.9、 −5.2] 和 −13.5 [−24.0、 −3.1] 个百分点，因此，在截止前集合中经受住这一移除操作的 Claude Opus 5.5 判定，在这些函数上依赖于动态切片。然而，仅置空一行本身也会携带信号。仅将触发器行置空，会使 Claude Opus 5.5 的 P-C 提高 32.6 [21.1、44.2]个百分点，因为在保持不变时被其判定为不易受攻击的 48 个易受攻击版本中，有 37 个在该行为空后会被判定为易受攻击。代码本应出现的位置出现空行，在模型看来意味着某些内容被移除了，这也是 L3 将置空集合与相同大小的随机行集合进行比较的原因。对编辑后的版本的判定也有所减弱。在截止后函数经执行确认的变体中，Claude Opus 5.5 能区分真实版本对中的 29.2%，与其他三个模型处于同一水平，因此其在截止时间前的显著 L4 优势并未延续（补充材料中的表 S8）。执行预测能力有所减弱。四个模型能够预测 70.2% 至 78.2% 的结果，比截止前集合少约 15 个百分点，远高于隐藏主体下限；它们与新增防护规则之间的配对差异——该规则损失约 5 个百分点——也从 6.0 至 14.3 个百分点缩小至 −6.0 至 2.0 个百分点，并且所有区间都包含零。

Claude Opus 5.5 的领先优势。 GPT 模型的成对判别维持在截止前水平，变化分别为 −4.8 [−14.5、5.3] 和 −1.2 [−10.6、9.1] 个百分点。Claude Opus 5.5 的 P-C 从 67.2% 降至 37.1%（−30.0 [−40.7、 −19.7] 个百分点），其变化与每个 GPT 模型变化之差分别为 −25.2 [−38.0、−12.0] 和 −28.8 [−42.5、 −15.6] 个百分点。同一供应商较早的模型 Claude Opus 4.8 也有所下降，降幅为 −11.9 [−21.9、 −1.8]个百分点。在截止前集合中，对于 66 个具有 2021年或更晚年份通用漏洞披露的函数，四个模型的 P-C 均低于具有 2019年和 2020年通用漏洞披露的函数；但随着通用漏洞披露年份变化，并未呈现稳定趋势（补充材料的 S20 节）。以这些较新的函数为基准时，Claude Opus 5.5 仍下降了 −18.9 [−34.0、 −3.2]个百分点，而其他三个模型的变化中包括零。在 97 个包含所有切片的截止后配对中，Claude Opus 5.5 仍是唯一高于机会水平的模型（16.0 [7.7、24.5] 个百分点）；其在这些配对中相对于 GPT-6.1-sol 的领先幅度为 7.2 [−5.2、18.6] 个百分点，在全部 195 个通用漏洞披露中为 2.1 [−6.7、11.3] ，均已不再显著。在全部 195 个通用漏洞披露中，四个模型区分出的配对比例接近其在这 97 个包含所有切片的配对中的比例；例如，GPT-6.1-sol 为 33.8 [27.2、40.5]%。其更大的

在 L1 和 L4 上截止前的领先优势，以及其对证据移除较弱的响应，都与训练暴露相一致，因此两个函数集合之间的差异仍可能是一种解释。

截止时间之后的开放权重模型。 开放权重模型在截止后的配对上同样保持在随机水平或以下，MCC 最高仅为 0.130，因此 L1 的截止前结果也适用于后续的通用漏洞披露。它们的内部归因仍停留在关键词捷径的水平，而在提示词中标记触发器行并不会显著改变其 P-C 值（补充材料中的第 S27 节）。

两个版本并列显示。 单个函数版本从其表层特征中透露的信息很少，表层分类器的曲线下面积为 0.530。当两个版本都可见时，“较短的版本是易受攻击的版本”这一规则对 84.8% 的配对判断正确，因为修复通常会增加代码，所以仅凭长度就能完成针对真实配对的二选一任务。当被问及两个版本中哪一个易受攻击时，商业模型对 96.0% 至 99.0% 的配对判断正确，高于这一规则；即使只单独展示修复版本，它们对其中大多数被其标记为易受攻击的配对也能判断正确。

设置的稳健性。随着函数长度增加，判定为易受攻击的比例也随之上升。对于长度不超过 30 行的配对，GPT-6.1-sol 将其中 35.2% 判定为两个版本均易受攻击；对于长度超过 200 行的配对，这一比例为 81.1%。与此同时，其 P-C 从 47.9% 降至 18.9%；Claude Opus 5.5 也呈现相同趋势，但整体比例较低，从 15.8% 升至 43.2%。较长的函数包含更多可能在某些输入下出错的语句，而第 2.2 节的筛查未在这些修复版本中发现缺陷，因此这一趋势反映了模型如何处理长函数带来的不确定性。在三次运行中，对于 83.0% 至 86.9% 的版本，重复查询得到的答案均相同，因此单次运行之间相差几个百分点属于这一噪声范围。移除 PrimeVul 末尾的换行符导致答案改变的频率，与再次提问时大致相同；经过邦费罗尼校正后，没有任何模型在通用弱点枚举的各类别之间表现出 P-C 差异（补充材料第 S5、S6 和 S27 节）。不过，各类别在位置与判别之间的关系上仍有差异。对于释放后使用和双重释放故障，GPT-6.1-sol 在 73.2% 的函数中引用了触发器行，但仅区分了 36.1% 的配对，因为两个版本中都能看到引发故障的释放操作，而修复针对的是决定该操作是否运行的状态。

发现。 对于模型不可能见过的漏洞，定位以及判定对动态切片的依赖依然存在。在这些漏洞上，L5 的优势和 Claude Opus 5.5 在 L1 的领先幅度有所缩小，这表明最强的截止时间前结果部分依赖于已知漏洞，或两组函数之间的差异。

10 讨论

证据所支持的结论。 表10汇总了四项发现。它们区分了汇总检测分数混为一谈的不同能力。故障定位的表现超越了关键词捷径，且在训练截止时间之后依然成立；独立判定仍能识别筛选后的修复版本，并且只部分依赖模型定位出的证据。给定具体输入后，模型会推理检查是否能阻止故障访问，但这一优势在新漏洞上有所缩小。开放权重推理模型有时优于定位基线，但其成对判别仍处于随机水平；正确配对多于颠倒配对，显示出对修复的敏感性较弱。模型是否依赖自己引用的行仍未有定论，因为 L3 对参考证据进行了干预。

为什么定位与判别会出现分歧。能够正确定位故障的同一批模型

表10 四项发现、其背后的层级、用于判定每项发现的比较，以及这些发现是否适用于训练截止时间之后发布的通用漏洞披露。商业模型的数据以百分点或百分比表示。

发现	层级	判定性比较（截止前集合）	截止时间之后
F1值：判断函数 与其修复之间的区别很难	L1	P-C高于机会水平，最高为43.4 [39.2, 47.4] (Claude Opus 5.5)，开放权重模型则无一达到，其中五个开放权重模型 正确判断的次数多于反向判断 成对样本中，两个版本均易受攻击的比例为 49.3% 至 54.9%（GPT 模型）	仅 Claude Opus 5.5 高于机会水平，高出 16.0 [7.7、24.5]
F2 定位 超过捷径， 依赖程度只是部分	L2、L3	引用触发器行的比例比 关键词规则高出 30.5 至 52.9 个百分点，而清空 D 会降低 三个模型的 P-C	幅度为 34.2 至 46.7 个百分点， Claude Opus 5.5 现在对 D敏感
F3 给定一个输入， 结果被 预测	L4、L5	准确率从 86.3% 提升至 94.6%，相较于新增防护规则的准确率为 80.3%	准确率从 70.2% 提升至 78.2%，与 rule
F4 的部分结果 最优结果 取决于已知的 CVEs	all	Claude Opus 5.5 的 P-C 为 67.2%	37.1%，变化幅度为 -18.9 $[-34.0, -3.2]$ 针对以下通用漏洞披露： 2021年发布 的

在单独查看修复代码时，却会正确地将其标记为易受攻击；而在同时查看两个版本时，它们能正确选出易受攻击版本；在看到具体输入时，它们则能预测修复代码会无故障运行。因此，缺失的能力是：在没有对照的情况下，对单个函数作出绝对判断。定位触发器行是在询问故障会在哪里发生，而判断一项修复是否有效，则是在询问到达该行的值是否仍可能违反其前置条件；单个函数的周边代码往往无法回答这个问题。检测器默认将其判为易受攻击，这与第9节的长度效应以及图3中两者皆易受攻击的代码对相符。

参考基准未涵盖的内容。 触发器行和动态切片描述了一个到达故障的运行。通过内存分配状态起作用的根因（例如示例 2 中的首个版本），或函数中缺失的检查，都位于切片之外；引用此类行会被计为额外行。因此，与参考基准的重叠只是定位效果的下限，而 L3 至 L5 的干预则通过无需任何逐行参考基准的测试对其进行补充。

对评估的启示。这项研究带来三项实践建议。除了 F1值，还应报告匹配对的配对级比率，因为 F1值会奖励不加区分地判定为易受攻击。应将证据与根据故障执行得出的参考基准以及关键词规则进行比较，因为庞大的静态参考基准和仅与随机行进行比较，都会夸大表面上的一致性。关于记忆的论断需要使用训练截止时间之后发布的数据，并将其与训练截止时间之前的最新数据进行比较。

对检测器的启示。 商业模型已经能为大多数函数指出触发器行，而 GPT 模型仍会将约半数函数的开发者修复版本标记为易受攻击。提供具体输入可以缩小这一差距，而在提示词中提供切片几乎不会改变这一点。因此，要取得进展，可能需要能揭示依赖关系和执行情况的训练信号或工具，例如切片条件目标或访问消毒器运行的能力。具体而言，审查补丁的检测器应同时查看两个版本（第9节），说明它预期在哪种输入下会出现故障，并根据故障运行和关键词规则对其解释进行评分。

XX：24 正确的判定，正确的理由?

11 对效度的威胁

构念效度。 这套阶梯是对理解的一种操作化。模型可能理解某个漏洞，但当结果取决于调用方时，不愿将单个函数判定为易受攻击。配对级视角和 L5任务（其中提供了驱动程序和输入）在一定程度上解决了这个问题。显式证据是一种自我报告，其忠实性无法保证 [26, 48], ，这也是 L3 和 L4 要检验判定对证据和已确认的编辑的敏感性的原因。

内部效度。 对于 152 个函数，触发器行是基于补丁得出的，并以开发者的修复为依据；补充材料中的表 S6 分别报告了这两组。证据移除仅修改易受攻击版本。根据协议，开放权重模型分为两组。五个没有推理模式的模型在答案词元处读取，因此它们与商业模型之间的差异同时反映了模型和协议的影响；而四个推理模型则按照商业模型所用的协议作答。VulnLLM-R-7B 和 Qwen3.6-27B 的归因、提示改述和线性头均来自在答案词元处读取的运行，而 gpt-oss-20b 和 Foundation-Sec-8B-Reasoning 则不具备这三者中的任何一项。将推理预算从 8,192 个词元提高到 32,768 个词元后，gpt-oss-20b 的 P-C 从 23.9% 降至 21.6%，Qwen3.6-27B 的 P-C 从 24.5% 降至 23.5%；在任一预算下，两者都没有超过各自的随机水平。我们评估的模型系列也参与生成了若干输入。Claude Opus 5.0 编写了测试框架和输入，因此 L5 向每个模型展示的驱动程序以及研究中纳入的故障均出自同一个 Claude 模型，这可能使 Claude模型更容易理解相关代码。所有驱动程序都由该模型编写，因此无法按驱动程序来源拆分 L5。Claude Opus 4.8 与其属于同一系列，却在四个商业模型中的 L5准确率最低，这说明该系列并未获得显著优势。GPT-6.1-sol 编写了替代修复方案以及 60 个无害输入中的 46 个；Claude Opus 5.5 则编写了截止日期后的诱饵样本，因此它们各自评判了自己生成的编辑，而 GPT-5.6-sol 和 Claude Opus 4.8 均未编写任何内容。商业模型针对每个提示词只查询一次，而重复测量用于估计其答案的变化幅度（第9节）。这些分层和 CWE 系列属于探索性分析，未进行多重性校正。关键词列表和表面特征是特定的选择，可能还存在更强的捷径。动态切片包含消毒器处理程序退出在 36 个函数中引入的控制依赖；我们测量了这些依赖对切片比较的影响（第 2.2 节）。

结论效度。共享同一通用漏洞披露的函数会被一并重新抽样，因此置信区间考虑了它们之间的依赖关系，而配对差异则在函数内部计算。分层和 CWE 系列仅用于探索，结论依据的是在不同模型以及两组通用漏洞披露中均一致的比较结果。商业模型对每个提示词仅查询一次，而第9节中的重复测量限定了单个答案的变动幅度。

外部有效性。 这些函数是 C语言和 C语言++函数，具有可复现的概念验证和完整的动态切片，因此更偏向于内存安全故障和较短的调用链。概念验证使用同一种确认方法构建，即搭配 Claude Opus 5.0 的 VulValidate，因此函数集也反映了其动态工具能够触发哪些故障，并将其归因于已标注函数。两个数据集都要求使用多种动态分析工具和不同输入测试修复版本，且未发现其他缺陷或漏洞；最终标签反映了这些测试的覆盖范围。本研究涵盖了 13 个大语言模型，但不包括 LineVul [19] 或 DeepDFA [43]等微调检测器。将相同的参考基准应用于这些检测器，是未来可立即开展的工作。

12 相关工作

基于学习的漏洞检测。 基于代码小工具 [31], 、图神经网络 [62, 8], 、Transformer [19], 和受数据流启发的架构 [43] 构建的检测器，在 BigVul [18], 、Devign [62], 和 DiverseVul [10]上取得了很高的分数。PrimeVul [16] 表明，标签噪声和重复数据会抬高这些分数，并提出了配对评估。我们采用匹配对，并为每个标签添加一个执行派生参照。

作为漏洞检测器的大语言模型。近期评估发现，大语言模型难以对漏洞进行推理，并且容易受到提示词和代码扰动的影响 [28, 49, 44, 61]。这些研究通过将解释与 CVE 描述或人工评估进行比较来判断推理能力。我们将证据与触发器行及故障运行的动态切片进行比较，并通过移除来检验依赖关系。

稳健性与捷径。检测器在保语义变换下会改变预测结果 [39, 38] ，并利用数据中的伪影 [25, 6, 21]。我们的测试使用捷径和对照来衡量这些影响，而反事实基准则将基于变换的测试扩展到标签经执行确认的变体。

解释评估。推理依据基准将模型证据与人工推理依据进行比较，并通过移除来检验忠实性 [15, 26, 1, 27]。对于漏洞检测器，解释方法曾依据补丁代码行或静态程序结构进行评估 [20, 50,23, 11]。我们以故障执行的事实取代这些参照基准，并量化静态切片与这些事实之间的差异。

截止时间之后的记忆与评估。语言模型可以逐字复现训练数据， [7], 而代码模型可以记住易受攻击的代码及其修复版本， [54],因此，在公开漏洞基准上的结果可能反映的是记忆。我们将识别探测与在所述训练截止时间之后发布并修复的通用漏洞披露配对，并将截止时间之后的结果与截止时间之前最近发布的通用漏洞披露进行比较，从而区分随时间推移的下降与截止时间处的骤降。

程序切片与执行推理。静态切片 [51, 47] 和动态切片 [29,3, 59] 是用于理解故障的经典工具，而 DeepWukong 使用静态切片作为模型输入 [12]。代码执行推理基准测试大语言模型能否预测程序行为 [32, 22]。L5 将这种评估方式应用于具有经消毒器验证的结果的漏洞。

13 结论

以执行为依据的测试能够区分汇总检测分数所混淆的内容。大语言模型检测器能够超越关键词捷径，定位易受攻击函数中的故障，而且在训练数据发布之后的漏洞上仍然如此。区分易受攻击函数与其筛选后的修复版本仍然很困难，并且只在一定程度上与模型所定位的证据相关。给定具体输入时，商业模型能够预测运行超出守卫条件后的结果，但这一优势在较新的漏洞上消失了。匹配对、执行派生参照、捷径控制组和截止时间之后的数据共同揭示了这些不同的能力。

参考基准

1 Julius Adebayo、Justin Gilmer、Michael Muelly、Ian Goodfellow、Moritz Hardt 和 Been Kim。显著性图的合理性检查。发表于神经信息处理系统进展（*NeurIPS*），第31卷，2018年。

XX：26 正确的判定，正确的理由?

2 Sandhini Agarwal、Lama Ahmad、Jason Ai、Sam Altman、Andy Applebaum、Edwin Arbus、Rahul K Arora、Yu Bai、Bowen Baker、Haiming Bao 等。gpt-oss-120b 和 gpt-oss-20b 模型卡，2025年。 3 Hiralal Agrawal 和 Joseph R Horgan。动态程序切片。第25卷，第246–256页。ACM，纽约，美国纽约州，1990年。 4 Anthropic。Claude opus 4.8 模型概述。 https://platform.claude.com/docs/en/models/opus-4-8/overview，2026年。训练数据截止日期：2026年1月。访问日期：2026年11月14日。 5 Anthropic。Claude opus 5.5 模型概述。 https://platform.claude.com/docs/en/models/opus-5-5/overview，2026年。训练数据截止日期：2026年6月。访问日期：2026年11月13日。 6 Daniel Arp、Erwin Quiring、Feargus Pendlebury、Alexander Warnecke、Fabio Pierazzi、Christian Wressnegger、Lorenzo Cavallaro 和 Konrad Rieck。机器学习在计算机安全领域的注意事项。发表于第31届USENIX安全研讨会（USENIX Security 22），第3971–3988页，2022年。 7 Nicholas Carlini、Florian Tramer、Eric Wallace、Matthew Jagielski、Ariel Herbert-Voss、Kath- erine Lee、Adam Roberts、Tom Brown、Dawn Song、Ulfar Erlingsson 等。从大语言模型中提取训练数据。发表于第30届USENIX安全研讨会（USENIX Security 21），第2633–2650页，2021年。 8 Saikat Chakraborty、Rahul Krishna、Yangruibo Ding 和 Baishakhi Ray。基于深度学习的漏洞检测：我们已经做到这一步了吗？IEEE软件工程汇刊，48(9):3280–3296，2021年。 9 Marek Chalupa。Dg：LLVM 位码分析与切片。发表于国际自动化验证与分析技术研讨会，第557–563页。Springer，2020年。 10 Yizheng Chen、Zhoujie Ding、Lamya Alowain、Xinyun Chen 和 David Wagner。Diversevul：用于基于深度学习的漏洞检测的新型易受攻击源代码数据集。发表于第26届攻击、入侵与防御研究国际研讨会论文集，第654–668页，2023年。 11 Baijun Cheng、Shengming Zhao、Kailong Wang、Meizhen Wang、Guangdong Bai、Ruitao Feng、Yao Guo、Lei Ma 和 Haoyu Wang。超越保真度：解释基于学习的检测器的漏洞定位。ACM软件工程与方法学汇刊，33(5):1–33，2024年。 12 Xiao Cheng、Haoyu Wang、Jiayi Hua、Guoai Xu 和 Yulei Sui。Deepwukong：使用深度图神经网络静态检测软件漏洞。ACM软件工程与方法学汇刊（TOSEM），30(3):1–33，2021年。 13 Davide Chicco 和 Giuseppe Jurman。马修斯相关系数（MCC）相较于F1分数和准确率在二分类评估中的优势。BMC基因组学，21(1):6，2020年。 14 Roland Croft、M Ali Babar 和 M Mehdi Kholoosi。软件漏洞数据集的数据质量。发表于2023年IEEE/ACM第45届国际软件工程会议（ICSE），第121–133页。IEEE，2023年。 15 Jay DeYoung、Sarthak Jain、Nazneen Fatema Rajani、Eric Lehman、Caiming Xiong、Richard Socher 和 Byron C Wallace。ERASER：用于评估合理化自然语言处理模型的基准。发表于第58届计算语言学协会年会论文集，第4443–4458页，2020年。 16 Yangruibo Ding、Yanjun Fu、Omniyyah Ibrahim、Chawin Sitawarin、Xinyun Chen、Basel Alomair、David Wagner、Baishakhi Ray 和 Yizheng Chen。使用代码语言模型进行漏洞检测：我们进展如何？2024年。 17 Bradley Efron 和 Robert J. Tibshirani。《自助法简介》。Chapman & Hall，1993年。 18 Jiahao Fan、Yi Li、Shaohua Wang 和 Tien N Nguyen。Ac/c++ 代码漏洞数据集，包含代码变更和CVE摘要。发表于第17届软件仓库挖掘国际会议论文集，第508–512页，2020年。

19 Michael Fu 和 Chakkrit Tantithamthavorn。LineVul：基于 Transformer 的行级漏洞预测。载于《第19届国际软件仓库挖掘会议论文集》，第608–620页，2022年。20 Tom Ganz、Martin Härterich、Alexander Warnecke 和 Konrad Rieck。解释用于漏洞发现的图神经网络。载于《第14届 ACM 人工智能与安全研讨会论文集》，第145–156页，2021年。21 Robert Geirhos、Jörn-Henrik Jacobsen、Claudio Michaelis、Richard Zemel、Wieland Brendel、Matthias Bethge 和 Felix A Wichmann。深度神经网络中的捷径学习。《自然·机器智能》，2(11):665–673，2020年。22 Alex Gu、Baptiste Rozière、Hugh Leather、Armando Solar-Lezama、Gabriel Synnaeve 和 Sida I Wang。CRUXEval：代码推理、理解与执行基准，2024年。23 Yutao Hu、Suyuan Wang、Wenke Li、Junru Peng、Yueming Wu、Deqing Zou 和 Hai Jin。用于基于 GNN 的漏洞检测的解释器：我们准备好了吗？载于《第32届 ACM SIGSOFT 国际软件测试与分析研讨会论文集》，第1407–1419页，2023年。24 Binyuan Hui、Jian Yang、Zeyu Cui、Jiaxi Yang、Dayiheng Liu、Lei Zhang、Tianyu Liu、Jiajun Zhang、Bowen Yu、Keming Lu 等。Qwen2.5-Coder 技术报告，2024年。25 Erik Imgrund、Tom Ganz、Martin Härterich、Lukas Pirch、Niklas Risse 和 Konrad Rieck。承诺落空：衡量基于学习的漏洞发现中的混杂效应。载于《第16届 ACM 人工智能与安全研讨会论文集》，第149–160页，2023年。26 Alon Jacovi 和 Yoav Goldberg。迈向忠实可解释的 NLP 系统：我们应如何定义并评估忠实性？载于《第58届计算语言学协会年会论文集》，第4198–4205页，2020年。27 Sarthak Jain 和 Byron C Wallace。注意力并非解释。载于《2019年北美计算语言学协会会议：人类语言技术论文集》第1卷（长篇与短篇论文），第3543–3556页，2019年。28 Avishree Khare、Saikat Dutta、Ziyang Li、Alaia Solko-Breslin、Rajeev Alur 和 Mayur Naik。理解大语言模型在检测安全漏洞方面的有效性。载于《2025年 IEEE 软件测试、验证与确认会议（ICST）》，第103–114页。IEEE，2025年。29 Bogdan Korel 和 Janusz Laski。动态程序切片。《信息处理快报》，29(3):155–163，1988年。30 Chris Lattner 和 Vikram Adve。LLVM：用于终身程序分析与转换的编译框架。载于《国际代码生成与优化研讨会，2004年。CGO 2004》，第75–86页。IEEE，2004年。31 Zhen Li、Deqing Zou、Shouhuai Xu、Xinyu Ou、Hai Jin、Sujuan Wang、Zhijun Deng 和 Yuyi Zhong。VulDeePecker：用于漏洞检测的基于深度学习的系统。载于《网络与分布式系统安全研讨会（NDSS）论文集》，2018年。 doi:10.14722/ndss.2018.23158。32 Changshu Liu、Yang Chen 和 Reyhaneh Jabbarvand。Codemind：评估大语言模型的代码推理能力，2024年。33 Xiang Mei、Jordi Del Castillo、Pulkit Singh Singaria、Haoran Xi、Abdelouahab Benchikh、Tiffany Bao、Ruoyu Wang、Yan Shoshitaishvili、Adam Doupé、Hammond Pearce 等。ARVO：开源软件可复现漏洞图谱。载于《2026年第11届 IEEE 欧洲安全与隐私研讨会（EuroS&P）》，第860–873页。IEEE，2026年。34 Yuzhou Nie、Hongwei Li、Chengquan Guo、Ruizhe Jiang、Zhun Wang、Bo Li、Dawn Song 和 Wenbo Guo。Vulnllm-r：用于漏洞检测的专用推理型大语言模型及智能体脚手架，2025年。35 OpenAI。GPT-5.6 Sol 模型页面。 https://developers.openai.com/api/docs/models/gpt-5.6-sol，2026年。知识截止时间：2026年2月16日。访问日期：2026年11月13日。

XX：28 正确的判定，正确的理由?

36 OpenAI。GPT-6.1 Sol 模型页面。 https://developers.openai.com/api/docs/models/gpt-6.1-sol，2026年。知识截止时间：2026年4月30日。访问日期：2026年11月13日。37 Qwen 团队。Qwen3.6-27B：27B 密集模型中的旗舰级编码能力。 https://qwen.ai/blog?id=qwen3.6-27b，2026年4月。38 Md Rafiqul Islam Rabin、Nghi DQ Bui、Ke Wang、Yijun Yu、Lingxiao Jiang 和 Mohammad Amin Alipour。关于神经程序模型对语义保持程序变换的泛化能力。《信息与软件技术》，135:106552，2021年。39 Niklas Risse 和 Marcel Böhme。揭示机器学习在自动漏洞检测方面的局限。载于《第33届 USENIX 安全研讨会（USENIX Security 24）》，第4247–4264页，2024年。40 Baptiste Roziere、Jonas Gehring、Fabian Gloeckle、Sten Sootla、Itai Gat、Xiaoqing Ellen Tan、Yossi Adi、Jingyu Liu、Romain Sauvestre、Tal Remez 等。Code Llama：面向代码的开放基础模型。arXiv 预印本 arXiv:2308.12950，2023年。41 Swarup Kumar Sahoo、John Criswell、Chase Geigle 和 Vikram Adve。利用可能不变量实现自动化软件故障定位。载于《第十八届编程语言与操作系统架构支持国际会议论文集》，第139–152页，2013年。42 Konstantin Serebryany、Derek Bruening、Alexander Potapenko 和 Dmitriy Vyukov。AddressSanitizer：快速地址有效性检查器。载于《2012年 USENIX 年度技术会议（USENIX ATC 12）》，第309–318页，2012年。43 Benjamin Steenhoek、Hongyang Gao 和 Wei Le。受数据流分析启发的深度学习，用于高效漏洞检测。载于《第46届 IEEE/ACM 国际软件工程会议论文集》，第1–13页，2024年。44 Benjamin Steenhoek、Md Mahbubur Rahman、Monoshi Kumar Roy、Mirza Sanjida Alam、Hengbo Tong、Swarna Das、Earl T Barr 和 Wei Le。犯错乃机器之常：漏洞检测挑战大语言模型推理，2024年。45 Evgeniy Stepanov 和 Konstantin Serebryany。MemorySanitizer：用于检测 C语言中未初始化内存使用的快速检测器++。载于《2015年 IEEE/ACM 国际代码生成与优化研讨会（CGO）》，第46–55页。IEEE，2015年。46 Mukund Sundararajan、Ankur Taly 和 Qiqi Yan。深度网络的公理化归因。载于《国际机器学习会议》，第3319–3328页。PMLR，2017年。47 Frank Tip。程序切片技术综述。1994年。48 Miles Turpin、Julian Michael、Ethan Perez 和 Samuel Bowman。语言模型并不总是会说出其所想：思维链提示中的不忠实解释。第36卷，第74952–74965页，2023年。49 Saad Ullah、Mingji Han、Saurabh Pujar、Hammond Pearce、Ayse Coskun 和 Gianluca Stringhini。大语言模型尚不能可靠地识别和推理安全漏洞？：全面评估、框架与基准。载于《2024年 IEEE 安全与隐私研讨会（SP）》，第862–880页。IEEE，2024年。50 Alexander Warnecke、Daniel Arp、Christian Wressnegger 和 Konrad Rieck。评估深度学习在安全领域的解释方法。载于《2020年 IEEE 欧洲安全与隐私研讨会（EuroS&P）》，第158–174页。IEEE，2020年。51 Mark Weiser。程序切片。第4期，第352–357页。IEEE，1984年。52 Fabian Yamaguchi、Nico Golde、Daniel Arp 和 Konrad Rieck。利用代码属性图建模和发现漏洞。载于《2014年 IEEE 安全与隐私研讨会》，第590–604页。IEEE，2014年。53 An Yang 等。Qwen3 技术报告。arXiv 预印本 arXiv:2505.09388，2025年。54 Zhou Yang、Zhipeng Zhao、Chenyu Wang、Jieke Shi、Dongsun Kim、Donggyun Han 和 David Lo。揭示代码模型中的记忆。载于《IEEE/ACM 第46届国际软件工程会议论文集》，第1–13页，2024年。

55 Zhuoran Yang、Ed Li、Jianliang He、Aman Priyanshu、Baturay Saglam、Paul Kassianik、Sajana Weerawardhena、Anu Vellore、Blaine Nelson、Neusha Javidnia 等。Llama-3.1-foundationai- securityllm-reasoning-8b 技术报告，2026年。56 Matthew D Zeiler 和 Rob Fergus。卷积网络的可视化与理解。载于欧洲计算机视觉会议，第 818–833 页。Springer，2014年。57 Leizhen Zhang 和 Sheng Chen。VulValidate：利用可执行证据审计函数级漏洞标签，2026年。58 Leizhen Zhang、Shuhan Chen 和 Sheng Chen。利用大语言模型进行可满足性求解：推理能力的配对评估。*ACM* 软件工程学报，3(FSE)：4599–4621，2026年。59 Xiangyu Zhang、Rajiv Gupta 和 Youtao Zhang。精确的动态切片算法。载于第 *25* 届国际软件工程会议，*2003*年。会议论文集。，第 319–329 页。IEEE，2003年。60 Xiangyu Zhang、Sriraman Tallam、Neelam Gupta 和 Rajiv Gupta。迈向执行遗漏错误的定位。载于第 *28* 届 *ACM SIGPLAN* 编程语言设计与实现会议论文集，第 415–424 页，2007年。61 Xin Zhou、Ting Zhang 和 David Lo。用于漏洞检测的大语言模型：新兴成果与未来方向。载于*2024* 年 *ACM/IEEE* 第 *44* 届国际软件工程会议：新思想与新兴成果，第 47–51 页，2024年。62 Yaqin Zhou、Shangqing Liu、Jingkai Siow、Xiaoning Du 和 Yang Liu。Devign：通过图神经网络学习全面的程序语义，实现有效的漏洞识别。第 32 卷，2019年。