Shuttle：面向智能合约审计的依赖属性工作流演化

薛越
CertiK
新加坡，新加坡
nerbonic@gmail.com

荣毅
哥伦比亚大学
纽约，美国
rong@cs.columbia.edu

Ziqiao Kong
南洋理工大学
新加坡，新加坡
ziqiao001@e.ntu.edu.sg

顾荣辉*
哥伦比亚大学
纽约，美国
ronghui.gu@columbia.edu

Dan She
CertiK
美国纽约市
dan.she@certik.com

Yuannan Yang
CertiK
美国纽约市
yuannan.yang@certik.com

Zhiju Yang
西雅图大学
美国西雅图
zhijuyang@seattleu.edu

Yuqiang Sun
南洋理工大学
新加坡，新加坡
yuqiang.sun@ntu.edu.sg

*通讯作者

arXiv:2611.31929v1 [cs.SE] 24 Nov 2026

摘要

在基于大语言模型的智能合约审计中，检测质量取决于模型能力和工作流设计。随着编码智能体能力增强，模型如今可以根据执行反馈修改审计工作流。然而，不受约束的修改往往无法带来稳定的提升。真实合约中的跨函数漏洞需要多步骤状态跟踪和边界推理，而粗粒度流水线往往会遗漏这些信息。端到端结果很少能显示究竟是哪一步丢失了线索，因此，直接重写可能导致已有检测结果退化，或在粗粒度的聚合评分下丢弃有效编辑。反复在少量项目上迭代还会引入针对特定案例的启发式规则，使提示词臃肿，并损害跨项目泛化能力。我们提出了 Shuttle，一种受约束的工作流演进系统，并将其应用于演进智能合约审计框架 LucidScan。Shuttle 的核心是 **溯源图归因**。它将每次 LucidScan 运行扩展为实例级依赖图，使用前向搜索定位候选线索丢失的节点，并将现有命中的源节点反向追踪至保护集。在评估合并后的批次后，Shuttle 通过节点级溯源账本直接移除导致性能下降的编辑；仅当耦合编辑在早期轮次中重叠于同一节点时，才退回到复用缓存的局部对比重跑。对于局部瓶颈，Shuttle 在任务分解、渐进式细化和通用性盲审的约束下生成候选编辑，然后推进保留的工作流。在十个训练项目上，Shuttle 将 LucidScan 从基线 G_0（每项目平均 C/H/M 召回率为 28.7%）演进至 G_8（82.1%）。在十三个未用于 M_{full} 演化后，在相同模型下，每项目平均召回率从 G_0 时的 28.6% 提升至 G_8时的 70.3%，在相同数据划分下高于 Pashov 和 Codex Security。在一场有 896 名研究人员参加、于 2026年11月结束的匿名线上公开审计竞赛中，演化后的扫描器在高严重性召回率方面并列总排名第1（6/8，75.0%），在计分的高/中严重性集群方面排名总排名第2（34/72，47.2%），提交有效率为 92.9%（78/84），总体排名第3。

1 引言

基于大语言模型的智能合约审计同时依赖基础模型能力与工作流设计：如何组织检查范围、如何跟踪跨函数状态交互，以及如何验证边界条件 [12, 8]。为实际协议手动调优固定工作流成本高昂，而且往往会遗漏新设计中的交互模式。因此，更强的编码与推理模型推动了反馈驱动的工作流演进，从通用的智能体合成 [3, 16]迈向 AgentFlow [6], 、 EvoHunt [13], 和 MetaVul [1] 等系统中的安全审计。

将工作流演进应用于项目级合约审计会带来两个问题。**首先，端到端审计结果和扁平化执行日志既无法定位多步骤流水线中的哪一步丢失了线索 [18, 19, 9]，也无法在一次合并轮次中区分有益编辑与有害编辑**。最终报告只会列出遗漏的根本原因。一旦流水线并行分支，扁平化日志就会掩盖制品读写依赖和各节点输出。直接重写随后会导致已有命中回退；而当有效编辑与有缺陷的编辑一起测试时，粗粒度的汇总分数会掩盖正向增益，并导致有效编辑被当作回退而丢弃（§6.4）。**其次，在有限的训练项目集合上迭代会引入特定案例启发式规则和提示词膨胀**。在少量任务集上演进工作流很容易过拟合项目特定模式 [14], ，而将规则不断堆叠到冗长提示词中会分散模型注意力，并降低推理能力 [7]。

为解决这两个问题，我们提出了**Shuttle**，这是一种受约束的工作流演化系统，并配套推出**Lucid-Scan**，即它所演化的智能合约审计框架（图 1）。遵循公开的 OWASP 代码审查指南 [11], ，我们以包含三个步骤的工作流 G_0初始化演化：

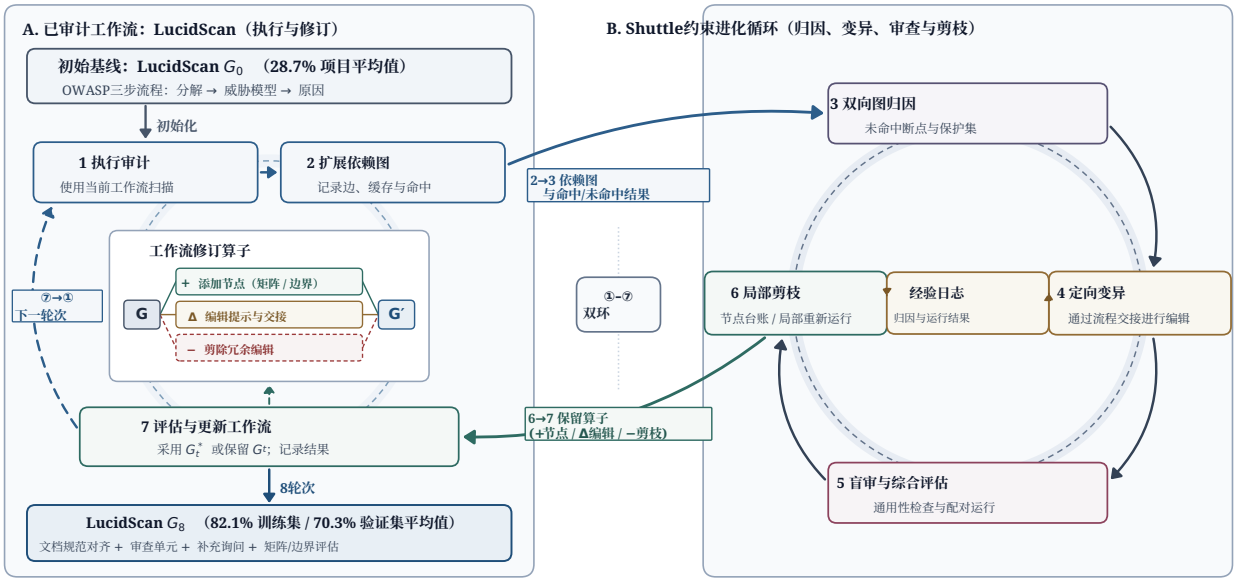

图1： Shuttle 如何根据执行依赖记录修订 LucidScan 的工作流（按步骤 1–7 顺序阅读）。左环 (A) 执行审计并扩展依赖图（步骤 1–2）；右环 (B) 执行双向图归因、定向变异、通用性盲审和局部剪枝，同时由经验日志综合归因与测试结果（步骤 3–6）；步骤 7 评估保留的工作流，并在八轮中将 LucidScan 从 G_0 更新为 G_8。

（模块分解、威胁建模和推理）。LucidScan运行智能合约审计工作流，而Shuttle保持其归因规则和通用性规则不变，并将每一轮的归因和结果记录在**经验日志**中。

首先，Shuttle 通过溯源图归因定位瓶颈并筛选编辑。 在**Loom**运行时环境下，每个组件都会声明其读取和写入的产物，并将任务节点及其读写产物记录为**依赖图**。溯源图归因在该图上执行三项操作。前向搜索定位候选线索丢失的节点。反向追踪识别现有命中的源节点，形成**保护集**。合并批次评估后，**节点溯源账本** 会直接移除导致性能下降的编辑；只有当早期轮次的编辑在同一节点上重叠时，才会回退到复用缓存的本地对比重跑。

其次，Shuttle 在通用性约束下生成可迁移的编辑。 提案者只能看到阶段级流程瓶颈，无法获取项目源代码或真实漏洞，并通过任务分解和渐进式细化来修订工作流 [4, 10]。在评估之前，每个候选编辑都必须通过**通用性盲审**，以排除包含项目特定或案例特定细节的规则。

通过这一循环，Shuttle 经过八轮演化 G_0，形成最终工作流 G_8。在未用于 M_{full} 演化的 13 个验证项目上，在相同模型下，单项目平均 C/H/Mrecall 从 G_0 的 28.6% 上升至 G_8的 70.3%，超过 Pashov 和 Codex Security。第 6 节回答四个研究问题：

- **RQ1（检测、泛化和扫描成本）：** 多轮演化对训练集和验证集上的召回率提升有多大，G_8 的召回率与外部工具相比如何，以及一次最终扫描的成本是多少？

调用与外部工具进行比较，那么最终扫描成本是多少？

- **RQ2（弱约束演化经验）：** 在不进行通用性盲审的早期弱约束试验（$M_{\text{弱}}$）中，会出现怎样的跨项目差异和令牌膨胀？
- **RQ3（核心机制与局部剪枝）：** 完整配置和简化消融配置在早期轮次窗口中的表现如何？在 G_3 和 G_7 上的节点级剪枝决策在重新运行中是否仍然有效？
- **RQ4（真实世界审计竞赛）：** 在一次真实世界审计竞赛中，演化后的工作流报告了多少个经组织方确认的漏洞？其严重性召回率和总体排名如何？

我们做出了两项主要贡献：

1.**受约束的工作流演进系统 Shuttle**。Shuttle 以实例级运行依赖图上的**溯源图归因**为核心，通过前向搜索定位未命中线索的节点，并通过反向追踪将稳定命中源代码追溯至保护集。结合通用性盲审和节点台账剪枝（以及在编辑存在耦合时复用缓存的本地重新运行），该系统能够移除负面影响或损害稳定性的编辑，同时避免盲目重写、错误回滚有效编辑以及针对特定案例的过拟合。

2. **演进后的审计框架 LucidScan 及其评估**。我们报告了 G_8 的验证召回率（在相同模型下为 70.3%，而 G_0 为 28.6%）、外部工具比较、扫描成本（每个验证项目 4.92美元

每次运行），早期轮次消融、G_3/G_7 剪枝重新运行，以及一场于 2026年11月结束的匿名公开审计竞赛的实时结果。

第 2 节阐述我们的设计动机。第 3–5 节介绍问题定义、方法和实现。第 6 节回答 RQ1–RQ4，第 7–9 节则涵盖相关工作、局限性和结论。

2 动机

2.1 一次未命中背后的四个局部断点

为说明为何诊断多步骤审计工作流中的失败需要查看中间状态，请考虑下面的 ShareVault 代码片段：其中，deposit 铸造份额 s，而 withdraw 在通过 call 转移资金前销毁 s：

```
合约 ShareVault { uint256 public totalShares; uint256public totalAssets;
函数 deposit(uint256 assets) external { uint256 s = assets * totalShares /
totalAssets;_mint(msg.sender, s); totalShares +=s; totalAssets +=assets; }
函数 withdraw(uint256 assets) external { uint256 s = assets *
totalShares / totalAssets;_burn(msg.sender, s); totalShares -=s;
totalAssets -=assets; payable(msg.sender).call{value: assets}(""); } }
```

该合约包含两个层面的缺陷：

1. **局部表面缺陷：** 取款未检查调用结果，存款也缺少 totalAssets == 0 防护。这两点通过单独检查一个函数即可发现。

2. **零份额免费提现漏洞：** 两个函数都会将 s = 资产 * 总份额 / 总资产向下截断。当先前的收益或捐赠使总资产 > 总份额（例如 100 > 10）时，小额取款（资产 = 1）会将 s 截断为 0。如果_销毁操作在 s == 0时不执行 Revert，调用者就可以在不销毁份额的情况下提取剩余资产（例如，调用 90 次 withdraw(1)，可在总资产 == 总份额 == 10之前提取 90 个单位的剩余资产）。

检测此缺陷需要经过四个步骤：**将共享账本状态的函-数置于同一审计范围 → 提出份额到资产的转换假设 → 在推理过程中传递该公- 式及其s = 0边界下游 → 替换s = 0 值**。如果审计报告只指出调用缺陷，却未发现零份额取款问题，那么最终输出就无法揭示流水线究竟在哪个环节失败。外部表现相同的未命中可能源于**四个不同的局部断点**：

- **范围缺失：** 取款函数和其他共享同一账本状态的函数被置于不同单元中，因此模型从未检查完整的状态更新路径。

- **候选项缺失：** 相关函数已纳入分析范围，但分析只停留在表层调用缺陷，从未提出共享截断问题。

- **交接信息丢失：** 上游指出了公式和 s= 0边界，却将它们压缩成含糊的“精度损失”总结，导致下游缺少具体条件。

- **错误判断：** 下游接收到 s = 0线索，却凭直觉将其忽略（“向下取整有利于金库”），而未使用具体数值检查_burn(msg.sender, 0)。

粗粒度步骤将全部四种失败合并为一个宏步骤。这样一来，归因只能指向整个步骤，迫使提案者在同一个提示词中堆叠临时规则。这些规则无法命中实际断点，会干扰正常运行的检测路径，并使提示词变长。将流水线拆分为明确的工件传递步骤后，依赖图便能在追踪现有命中的源节点时，分离出线索丢失的位置。

2.2 过拟合、提示词膨胀和多重编辑干扰

在有限的训练集上反复编辑工作流会带来另外三种风险。首先是**特定案例过拟合**：如果提案者直接看到 ShareVault，就容易编写合约特定规则，例如检查 _burn 是否截断为零。其次是**提示词膨胀**：将临时规则堆叠到一个步骤中会增加上下文长度和扫描令牌成本（§6.3 中约束较弱的试点显示，每个项目的扫描令牌增加了 26.6×，同时验证集偏差显著加剧）。第三是**多重编辑干扰**：当多个编辑在同一轮次中合并时，汇总得分的提升可能掩盖负面或冗余分支（§6.4）。Shuttle 通过角色级信息隔离、通用性盲审、节点账本局部剪枝和经验日志来应对这些风险，以任务分解和渐进式细化取代规则堆叠。

3 问题定义

3.1 演化目标与角色隔离

审计工作流 G 协调多个步骤的大语言模型调用和确定性脚本，以处理项目的源代码和文档。我们根据 OWASP 代码审查指南 [11]，将 G_0 初始化为包含三个步骤的流水线。给定 G_0 和训练集划分 $\mathcal{D}_{\text{train}}$，Shuttle 搜索合法的工作流修订 Δ，以获得 $G' = \Delta(G)$，同时确保步骤间输入/输出依赖有效，并满足 §4.3 中的通用性规则。在整个演化过程中，模型权重以及图归因、角色隔离和通用性盲审等外部规则保持不变。只有工作流 G会演化，而 Shuttle 经验日志则在各轮次间持续更新，记录每轮的归因和回归教训。

为防止 §2.2 中针对特定案例的过拟合，基准项目按代码仓库边界划分为训练集 $\mathcal{D}_{\text{train}}$和验证集 $\mathcal{D}_{\text{val}}$，并将各作用的输入相互隔离。审计与

归因会读取项目源代码和运行记录，而提案者与盲审员则无法看到项目名称、源代码或真实漏洞（§4.3）。

3.2 评估指标

当审计发现的根本原因与某个真实漏洞（GT）相匹配时，即计为一次命中；每次运行中，每个 GT 最多计一次。在评估划分 $\mathcal{P}$ 中，每个项目进行 k 次独立运行。已知问题 $\mathcal{B}_p$（$|\mathcal{B}_p| > 0$）、问题总数 $M = \sum_{p\in\mathcal{P}} |\mathcal{B}_p|$和命中指示变量 $x_{b,r}(G) \in \{0,1\}$的定义如下：单项目召回率 $R_p(G)$、**每项目平均召回率** $R_{\mathrm{proj}}(G)$，以及多次运行并集覆盖率 $U(G)$：

$$R_p(G) = \frac{\sum_{b\in\mathcal{B}_p}\sum_{r=1}^{k} x_{b,r}(G)}{k|\mathcal{B}_p|}$$

$$\bar{R}_{\mathrm{proj}}(G) = \frac{1}{|\mathcal{P}|}\sum_{p\in\mathcal{P}} R_p(G)$$

$$U(G) = \frac{\sum_{p\in\mathcal{P}}\sum_{b\in\mathcal{B}_p}\mathbb{I}[\sum_{r=1}^{k} x_{b,r}(G) > 0]}{M}$$

我们将**每项目平均召回率** $R_{\mathrm{proj}}(G)$作为主要评估指标：对各项目的 k次运行召回率进行平均，并对每个项目赋予相同权重，避免问题密集型代码库占据主导地位。我们还将多次运行并集覆盖率 $U(G)$作为次要指标，并报告扫描成本。

4 方法

Shuttle 的每一轮都遵循图1和算法 2，分为四个步骤：运行当前工作流并扩展其**依赖图**（§4.1）；在图上执行**双向归因**（§4.2）；在指导和**通用性盲审** 下生成编辑（§4.3）；合并评估后，根据**节点溯源账本** 进行剪枝并选定一个版本（§4.4）。依赖图、双向归因和账本剪枝共用同一张图，合称为**溯源图归因**。

4.1 运行依赖图：一次运行记录了什么

Loom 运行时将每个步骤表示为声明式组件 $c = (\mathrm{In}_c, \mathrm{Out}_c, f_c, \pi_c)$：它读取的产物、写入的产物、所执行的操作（大语言模型调用或脚本），以及执行策略（按单元、并行或聚合）。任务实例完成后，执行器会留下三类记录，归因只会读取这些记录（图 2A）：

1. **制品与来源记录**。 输出按内容哈希存储，并附带生成它的任务实例、所读取上游制品的哈希，以及提示词和其他资产的版本。如果任务实例 v 读取了任务实例 u的制品，我们就添加一条边 $u \to v$；运行依赖图 $G^{\mathrm{run}}(G, p, r) = (V, E)$ 直接根据这些记录构建，节点为任务实例。

2. **候选项 ID**。 候选项带有层级化的模块–候选项–报告 ID，下游节点会逐字保留这些 ID，因此可以在每个节点的产物中精确定位同一个候选项。演化过程中添加的节点会在其 ID 中加入可区分的后缀，最终报告据此记录其经过的节点。

3. **执行记录和文字记录**。 推理类节点会附加轻量级执行记录（已读取文件、运行的命令），供归因读取；完整文字记录仅作存档，绝不会被读取。

Shuttle 会写入一份**合并清单**，记录每次编辑 e 修改或新增的节点集合$V(e)$。每次编辑都针对单个局部瓶颈；对同一节点进行的互斥编辑则分别放入不同批次。

4.2 双向归因：未命中时向下游搜索断点，命中时向上游搜索源代码

归因会读取 G^{run}、GT、匹配结果以及来源快照（图 2A、算法 1）。根据每个 GT 在 $k = 3$ 次运行中的命中数 h，将其分流： $h \geq 2$ 进入反向追踪，其他则进入正向定位。

正向定位（未命中）：找到线索从存在变为缺失的节点。

算法 1 双向图搜索与归因 输入：依赖图 G^{run}、目标真实漏洞 b、命中数 $h \in \{0,\dots,k\}(k = 3)$、来源快照 输出：覆盖状态翻转断点、孤立未知项或受保护的源代码集 $\mathcal{S}_t(b)$ 1：将 G^{run} 剪枝为与目标 b 相关的子图 G_b {保留与真实漏洞相关的单元和索引边} 2：如果 $h \leq 1$ 则 3：在 G_b上初始化工作列表 Q；通过编程智能体检查节点的输入/输出覆盖率 $\to$ {已覆盖、部分覆盖、未覆盖、未知} {未命中/不稳定：四状态工作列表搜索} 4：当工作列表 $Q \neq \emptyset$（在查询预算内）非空时执行 5：弹出未访问的 $u \in Q$ 并标记为已访问；如果输入 $=$ 已覆盖而输出 $=$ 未覆盖，则记录断点并跳过扩展{要求提供引用的输入和输出证据} 6：如果输出 $=$ 已覆盖，则 7：将子节点加入 Q 8：否则，如果输出 $=$ 未覆盖，则 9：将相关父节点加入 Q 10：否则，如果输出 $=$ 部分覆盖，则 11：将子节点和相关父节点都加入 Q 12：否则 13：将节点放入未知集合 {输出 $=$ 未知或来源信息不完整} 14：结束如果 15：结束循环 16：标记丢弃输入候选项的合并节点；检查所有并行生成器 17：返回覆盖状态翻转断点或孤立未知记录 18：否则 19：根据匹配的发现项标识符定位终端生产者，然后沿依赖边反向追踪，并在父级制品中匹配相同的候选标识符，直到最早出现该候选项的节点 $\mathcal{S}_t(b)$（不进行模型调用）{$h \geq 2$：反向追踪} 20：返回 $(b, \mathcal{S}_t(b))$，作为受保护集 Φ_t21：结束如果

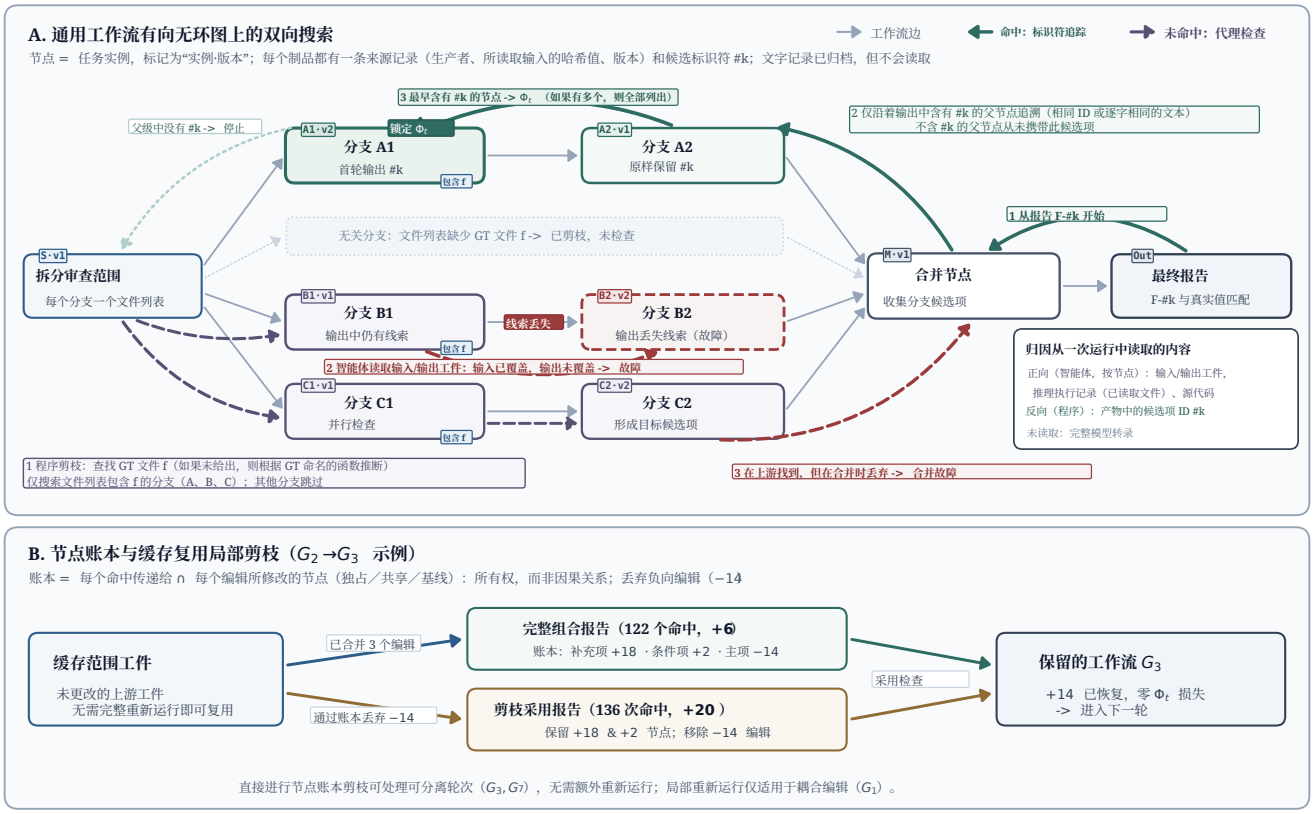

图 2： 溯源图归因（依赖图上的双向归因和局部剪枝）。（A）节点是任务实例，标记为"instance·version"；每个产物都包含一条来源记录（生成该产物的实例、所读取上游制品的哈希值、版本）和候选项 ID，而完整文字记录会被归档但不会被读取。对于未命中（紫色和红色虚线），程序会剪枝无关分支，智能体读取每个节点的输入和输出产物以判断覆盖情况，并持续推进，直到出现"输入已覆盖、输出未覆盖"的断点。对于命中（绿色线），程序将父级制品中的候选项 ID 向后匹配至候选项最早出现的节点，并将其添加到保护集 Φ_t，无需调用模型。（B）节点溯源账本：通过求交每次命中经过的节点与每次编辑修改的节点，将命中归因于编辑，并移除负面项（$G_2 \to G_3$： $122 \to 136$ 命中）。

• **剪枝（程序）**。取出 GT 指定的文件（如果没有指定，则根据 GT 提及的函数名推断），并且只检查文件列表中包含这些文件的分支。

• **判断一个节点（智能体）**。 智能体每次接收一个 GT 和一个节点：该节点的输入和输出产物、源代码以及节点的职责。它会标注输入和输出是否分别涵盖目标机制（已涵盖、部分涵盖、未涵盖、未知），并引用产物文本。推理节点的输入只有在已经包含该候选项时才算涵盖。

• **推进（程序）**。 已涵盖的输出向下游移动；输入已涵盖而输出未涵盖的位置就是断点；输入未涵盖则向上游移动；未知或不完整的记录会停止搜索。如果合并节点丢弃了在上游形成的候选项，则该合并节点就是断点。

• **结果**。 每个断点都带有一种失败类型（§2.1：范围遗漏、候选项未形成、交接丢失、推理误判，以及"判断正确但未收集"），并按步骤和类型进行汇总，提供给提案者（§4.3）。

反向追踪（稳定命中）：沿标识符追溯至最早的生产者，无需模型调用。

• **匹配**。 从匹配报告的标识符开始，在父节点输出中查找相同的标识符或逐字相同的文本，并在找到时继续向上游追踪；仅比较字段，不进行语义判断。

• **仅沿包含该标识符的父节点追踪**。一个节点可能读取多个父节点；其输出中不含该标识符的父节点从未携带此候选项。

• **保护集**。 最早到达的节点是源代码；所有稳定命中的源代码构成**保护集** Φ_t，而涉及 Φ_t 的编辑必须通过第 4.4 节中的回归检查。

表 1： Shuttle 中的两层工作流修改操作（文本/字段编辑和图结构编辑）。

层	基本修改操作	常见组合与作用
Text	编辑片段, 添加/删除片段，编辑提示词，编辑模式字段	添加细化：最小幅度提示词和输出字段编辑
图拓扑	添加节点、删除节点、重新连接边, 编辑执行策略	拆分步骤，添加并行通道，添加验证, 更改粒度、合并步骤

4.3 工作流修订、经验日志和通用性盲审

Shuttle 通过三种机制将**步骤失败的位置** 与**如何修改该步骤**分离（图1步骤 4–5）：

1. **信息隔离和经验日志**。 归因结果按步骤和失败类型汇总，去除项目名称、标识符、源代码和真实值详情；提案者只能从经验日志中看到步骤级瓶颈以及过去的成功和失败。

2. **任务分解和渐进式细化**。 提案者通过拆分步骤并添加中间证据来改进工作流，而不是将缺陷检查清单堆叠到一个节点中；编辑分为文本层和图层（表 1）。

3. **通用性盲审**。 硬不变量是，**演化只能修订**如何进行审计，**而不能修订**要查找什么缺陷。独立盲审会剔除带有项目或案例特定特征的编辑；保留下来的无冲突编辑 E_t form $G_t^{\text{cand}} = G_t \oplus E_t$ for evaluation on $\mathcal{D}_{\text{train}}$。

4.4 局部剪枝与选择

当一轮合并多个编辑 $e \in E_t$时，较高的总数仍可能掩盖无用或有害的编辑。**节点溯源账本**将每份最终报告经过的节点与每个编辑的 $V(e)$进行交集运算（图 2B），分配命中，并按照表 2 选择如何移除编辑。该账本提供的是归属关系，而非因果关系，因此只有在编辑能够清晰分离时才会直接使用。

对已剪枝的G_t^* 进行评估，在 $\mathcal{D}_{\text{train}}$ 上依据 C/H/M 召回率、 Φ_t 稳定性、并集覆盖率和成本进行判定：若通过，则$G_{t+1} \leftarrow G_t^*$；否则保留 G_t 。结果将记录到经验日志中。

表 2： 节点溯源账本的命中分配和编辑移除规则。

Step	Case	处理
分配	命中与一个编辑相交	仅属于该编辑；计入其净增益
分配	命中与多个编辑相交	共享/未分配；不计入任何单个编辑
分配	命中与编辑无交集	基线
分配	旧命中在新版本	查找以下内容的节点旧命中的；每次编辑修改其中一个的编辑计为一次损失
移除	沿.....可分离的编辑依赖边（例如, G_3, G_6, G_7, G_8)	删除净增益编辑 ≤ 0或 Φ_t 回退
移除	编辑共享一个节点，或彼此耦合（例如， $G_0 \to G_1$）	直接冻结上游缓存并重新运行 G_t^{cand} 与。$G_t^{\text{cand}} \setminus \{e\}$ 本地

算法 2一轮演化

输入： G_t、已评估运行记录、修改操作、通用性约束和经验日志 1：在 $\mathcal{D}_{\text{train}}$ 上执行 G_t （或复用上一轮中版本匹配的记录）2：扩展依赖图 G^{run}（G_t、 p、 r），记录内容哈希和匹配结果 3：按 k运行命中数 $h \in \{0, \ldots, k\}$ （$k = 3$）对 GT 进行分层，并在 G^{run} 上运行双向图归因，通过确定性索引和针对每个 GT 的局部搜索，定位未命中瓶颈并锁定 Φ_t4：汇总阶段级流程瓶颈，并在任务分解、渐进式细化和经验日志的指导下生成候选编辑 5：执行通用性盲审；合并无冲突编辑 $G_t^{\text{cand}} = G_t \oplus E_t$ 并在 $\mathcal{D}_{\text{train}}$上进行评估 6：直接从节点溯源账本中剪枝净增益为 ≤ 0 或出现 Φ_t 回退的编辑（当编辑存在耦合时， $G_t^{\text{cand}} \setminus \{e\}$ 则冻结兼容的上游缓存，以进行配对留一重跑），从而获得 G_t^*7：在 $\mathcal{D}_{\text{train}}$上评估保留的 G_t^* ，决定是否接受（$G_{t+1} \leftarrow G_t^*$ 或保留 G_t），并更新经验日志

输出： G_{t+1} 及其评估记录

5 实现

该系统包含三层：**Loom**，一种漏洞无关的多智能体运行时，用于解析读写接口、调度并发调用并持久化产物；**LucidScan**，即审计工作流；以及**Shuttle**，仅在线下运行归因、提案、盲审和剪枝，因此最终的G_8 可直接在Loom上运行。 G_0的三个步骤分别成为 G_8 的一层（图 3）：

- **第1层（分解）**。从文档和注释中提取设计预期，并将其与函数对应；划分共享状态的审查单元，并将未分组的函数重新纳入范围。

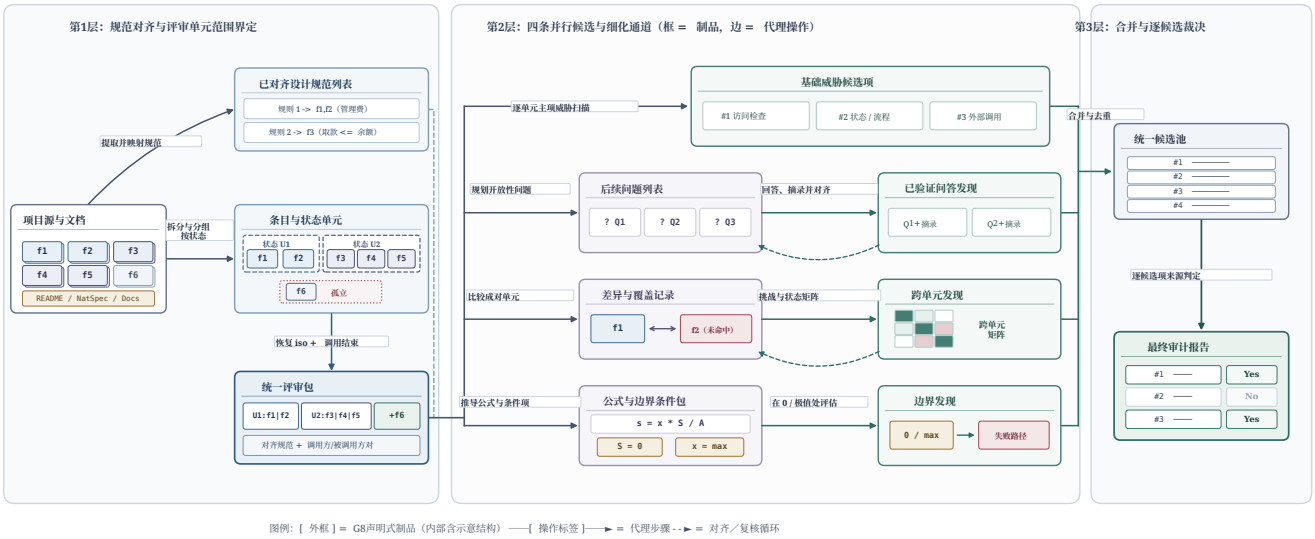

图 3： LucidScan G_8 的分层工作流架构由 G_0的三步基线演化而来。 G_0的三个步骤分别扩展为一个工作流层，并包含内部子工作流：第1层从文档中提取设计预期并划分审查单元；第2层运行四个并行的候选项与细化子工作流；第3层汇总候选项，逐候选项验证代入边界，并输出发现。

- **第2层（建模）**。运行四条并行通道：主项威胁建模；补充询问（规划问题、独立作答、检查一致性）；使用关系矩阵比较单元；以及条件推导与边界评估。

- **第3层（推理）**。 合并并去重所有通道中的候选项，然后通过具体值代入和来源证据对每个候选项进行判断。

6 评估

本节回答§1中的四个研究问题。 M_{full}是§4中的完整方法； M_{weak} 是早期约束较弱的试验版本； M_{guide}、 M_{trace}和 M_{miss} 是消融实验。

6.1 数据集与评估协议

早期的 M_{weak} 试验版本（§6.3）基于按时间收集的项目集进行演进，在代币合约上的召回率较高，但在借贷和金库协议上的表现则大幅下降。因此，M_{full} 采用了重建的基准集，其中包含来自2026年Sherlock和Yieldoor审计竞赛以及未公开内部审计的23个真实项目；GT是各项目团队在其最终审计报告中接受的问题。遵循PropertyGPT的方法，[8], 使用亲和传播对311个去重合约的源代码嵌入进行聚类，划分为10个协议簇（附录图8），并按代码仓库划分为10个训练项目（覆盖8个簇）和13个验证项目（表3）；其余设置见表4。

表 3： 基于源代码相似性的 23 个基准项目的协议类型聚类（10 个簇）以及 10/13 训练集–验证集划分（203 个训练集 GT，其中包含 79 个 C/H/M；254 个验证集 GT，其中包含 114 个 C/H/M）。

簇与类别	项目（训练集 / 验证集）
C0 代币销售与归属	Aqualux / —
C1 质押代理	— / Bankroll
C2 NFT 借贷与清算	Realworld / Gondi
C3 LST 与策略金库	— / Hann、高流动性金库
C4 NFT市场与绑定机制	Phi / Idols NFT
C5 质押与分发	Money、Starlinkclaw / Ncoin
C6 去中心化交易所、计量器与路由器	Appex、ParagonDEX / Aimeta, Uni_ai、WooFi
C7 挖矿与代币	Steel / —
C8 Meme代币与分发器	Upepe / —
C9 AMM钩子与杠杆	Yieldoor / Bunni、Frgmnt、Morpho、Revert
总计	**10 个训练集（8/10）/ 13 个验证集**

6.2 RQ1：检测效果、泛化能力与扫描成本

问题。 多轮演化在多大程度上提升了 LucidScan 的召回率和覆盖率？ G_8 的跨项目召回率与外部基线相比如何？其每次扫描的 API 成本是多少？

训练。 经过八轮训练，C/H/M 的每项目平均召回率从 28.7% 提升至 82.1%；在 G_8 处，三次运行的波动幅度为 1.1 个百分点（图4、附录表10）。

验证集。 在相同模型下， G_8 将每个项目的平均召回率从 28.6%（G_0）提高至 70.3%（11 个项目

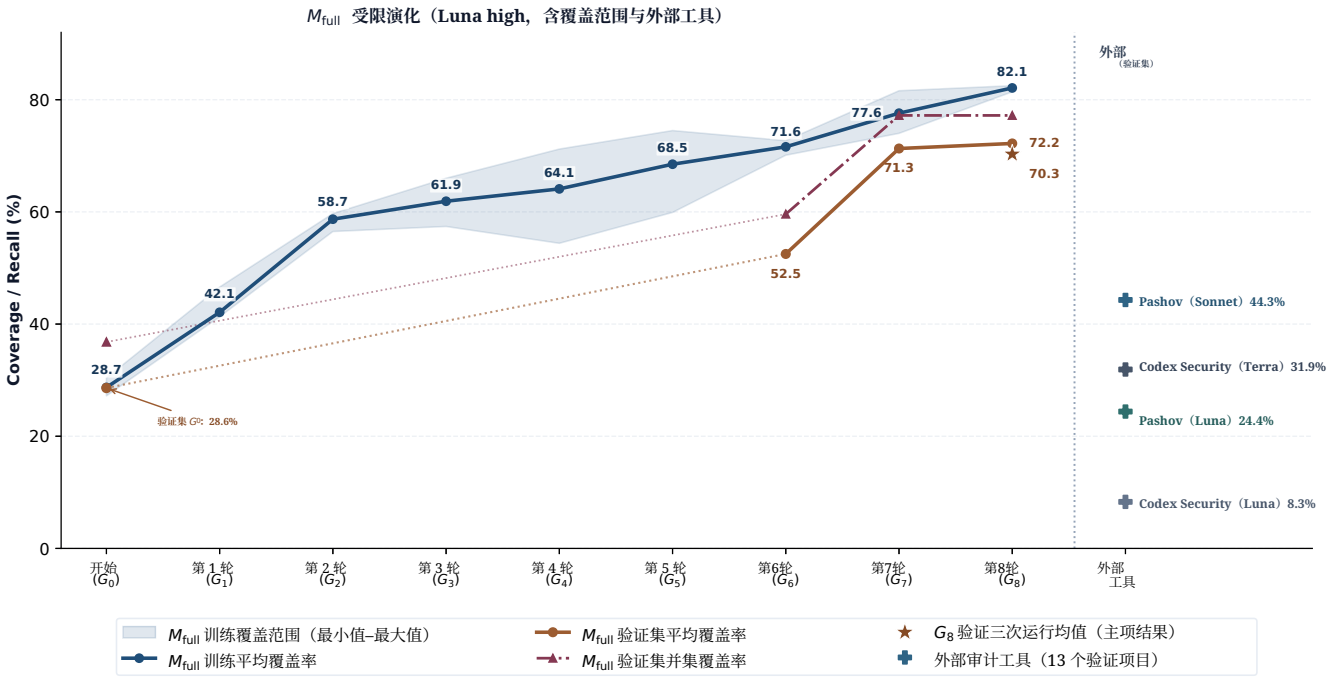

图4： 在 Luna high 下 M_{full} (G_0–G_8) 的训练轨迹（在 10 个训练项目上的三次运行均值及最小值–最大值带；最终的 G_8 在三次运行并集中检测到 64/79 个 C/H/M真实标签），以及配对的验证集基线（28.6%，三次运行均值）、G_6–G_8 的匹配两次运行验证集均值、最终 G_8 3 次运行验证集均值 70.3%（星号）和单次运行的外部基线。

表4： 实验设置：GT规模、运行次数、优化目标、选择依据及每种作用对应的模型数。

设置	值
GT规模	训练集 203（79 C/H/M）；验证集 254（114 C/H/M）
Runs	训练集：每个版本 3 个；验证集：G 3 个$_0$ 以及 G_8，G 的 2 $_6$ and G_7；外部基线：1 个使用默认设置
优化目标	归因、提案和选择目标 C/H/M问题 only
版本选择	仅使用训练结果；验证集不发挥作用
扫描模型	Luna high（由 G_0、 G_8共享，以及 Pashov Luna）
归因与提案	Sol medium；提案者孤立来自盲审
根因判定器	Sol high，工具盲化 [13]；a 异构模型一致认为 60 个分层样本中的 58 个 ($\kappa = 0.933$)
运行时间	最多 8 个并发智能体，超时 40 分钟，重试 1 次；每次运行使用独立缓存

有所提升，2 个持平），并将三次运行并集覆盖率从 36.8% 提高至 78.9%，超过所有外部工具（表5）。在 KnowDit [5]，评估的 5 个项目中，G_8 在统计所有严重程度后达到

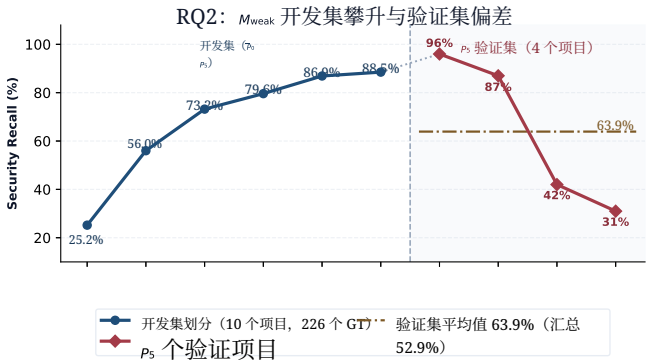

图 5： 弱约束 M_{weak} 开发集召回率在P_0–P_5 之间的变化（左）以及 P_5 在四个验证协议上的迁移（右），展示了明显的跨协议差异。

61.9%，KnowDit 为 24.9%。

成本。 G_8 每个项目每次运行的直接 API 支出为 4.92美元（附录表 11）。

总结（RQ1）。 在使用相同模型的情况下，演化将验证集 C/H/Mrecall 从 28.6% 提升至 70.3%，高于所有外部工具。

6.3 RQ2：弱约束演化（M_{weak}）带来的泛化经验

问题。 在缺乏通用性盲审的早期弱约束演化（M_{weak}）中，会出现哪些跨项目差异和令牌膨胀?

M_{weak} 在按时间顺序收集的另一组数据上运行，该数据集包含

表5： 全部13个验证基准项目的各项目 C/H/M 召回率（G_0 和 G_8：3轮均值；外部基线：默认单次运行；按行最大值加粗； † KnowDit 在全严重度真实标签下对5个验证项目进行了评估；其 Hann 运行命中了400美元的预算上限）。

项目（C/H/M，全部）	G_0（C/H/M）	M_{full} G_8（C/H/M）	Pashov（Son.）	Pashov (Ter.)	Pashov (Luna)	CS（Terra）	CS（Luna）	KnowDit
Aimeta (5, 17)	47%	**80%**	**80%**	**80%**	0%	60%	0%	—
Bankroll (3, 5)	33%	**67%**	33%	**67%**	**67%**	33%	0%	—
Bunni (1, 1)	0%	0%	0%	0%	0%	0%	0%	—
Frgmnt (17, 31)	20%	**73%**	59%	41%	6%	29%	6%	48%
Gondi (37, 37)	13%	**67%**	49%	35%	16%	32%	5%	27%
Hann（2，8）	17%	**100%**	0%	0%	0%	0%	0%	13%
Idols NFT（3，3）	44%	89%	**100%**	**100%**	**100%**	67%	33%	—
Morpho（7，58）	0%	**38%**	14%	0%	14%	0%	0%	10%
Ncoin（21，46）	41%	**86%**	33%	24%	14%	29%	14%	—
Revert（13，19）	8%	**49%**	8%	8%	0%	15%	0%	26%
Uni_ai（2，25）	**100%**	**100%**	**100%**	**100%**	**100%**	50%	50%	—
高流动性金库 (2, 3)	17%	**100%**	**100%**	**100%**	0%	**100%**	0%	—
WooFi (1, 1)	33%	**67%**	0%	0%	0%	0%	0%	—
每项目平均值	**28.6%**	**70.3%**	44.3%	42.7%	24.4%	31.9%	8.3%	24.9%†

10 个开发项目（226 个安全 GT）和 4 个验证项目。经过五轮迭代，开发集召回率从 25.2% 提升至 88.5%，而每个项目的扫描令牌数增长了 26.6×（图 5，左）。在验证集上，令牌协议的表现仍然较高（Aimeta 为 96%，Steel 为 87%），而 Revert（42%）和 Morpho（31%）则有所下降，每项目平均值为 63.9%（汇总后为 52.9%；图 5，右）。

总结（RQ2）。未经过盲审的演化在开发集上有所提升，但会带来31%–96%的跨项目差异和26.6× 的令牌膨胀，因此 M_{full} 加入了去除大小写的交接信息、通用性盲审以及协议多样化划分。

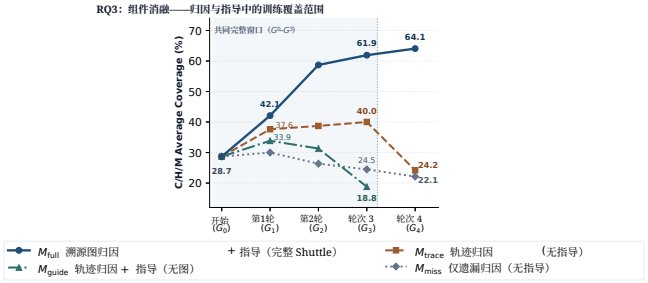

图6： G_0– G_4 期间早期轮次的 C/H/M 每项目平均召回率（每个项目的 3 次运行均值；阴影部分：共同的 G_0–G_3 窗口）。完整配置（M_{full}）持续上升，而三种简化消融实验均有所回落。

6.4 RQ3：核心机制消融、局部剪枝与溯源对比

问题。 在早期轮次消融窗口内，完整配置和简化消融配置表现如何？针对 G_3 和 G_7 的节点级剪枝决策在重新运行时效果如何？

（1）早期轮次组件消融。 四种配置使用相同的数据、扫描方式和判定模型，并从完整配置开始逐步移除组件（表6）。三种无图配置无法使用账本，而是从每轮的合并输出继续推进；八轮中从未发生整轮回滚。完整配置持续上升，而三种消融配置在早期窗口内均有所回落（图6）。

（2）八轮中的版本采纳。 三轮采纳了所有编辑，五轮移除了一部分编辑，使训练命中数从64提升至182（表7，附录图9）。

(3) 纯日志与图决策。 我们分别将 $G_2 \rightarrow G_3$ 和 $G_6 \rightarrow G_7$ 打包两次：一个纯日志数据包，包含最终发现文本、总分和执行轨迹（与 M_{guide}使用相同的归因输入）；另一个图数据包，则额外加入了产物读写边以及节点溯源账本。两个彼此孤立、配置完全相同的分析代理仅在此输入上有所不同，并分别独立作出决策（图7、表8）。纯日志智能体回滚了 G_3 上的所有内容，并已采用 G_7上的所有内容；图智能体则根据账本部分采用，而两次重新运行的结果都支持其决策。

总结（RQ3）。 完整配置在早期轮次持续上升，而消融实验则有所回落；台账引导的移除使 G_3 的命中数从 122 提升至 136，并且在 G_7 上，删除两个零收益编辑后达到 173 次命中，同时恢复所有受损的稳定命中。

6.5 RQ4：真实世界审计竞赛评估

问题。在一场真实世界的智能合约审计竞赛中，经过演化的 LucidScan 工作流报告了多少个经主办方确认的漏洞，其排名如何？

结果。 除了离线基准测试，我们还在一场匿名的实时公开审计竞赛中部署了演进后的工作流。该竞赛于2026年11月结束（896名提交研究人员，

表6： 早期轮次消融：四种配置的组成，以及 C/H/M 训练集在 G_1–G_4 期间的每项目平均召回率（G_0 对所有配置均为 28.7%； M_{guide} 仅运行至 G_3）。

配置	溯源图	分解/细化指导	归因输入	G_1	G_2	G_3	G_4
M_{full}	Yes	Yes	依赖图	**42.1%**	**58.7%**	**61.9%**	**64.1%**
M_{guide}	—	Yes	执行轨迹	33.9%	31.3%	18.8%	—
$M_{\text{轨迹}}$	—	—	执行轨迹	37.6%	38.7%	40.0%	24.2%
M_{miss}	—	—	仅未命中	30.0%	26.4%	24.5%	22.1%

表7： 涵盖全部八个演进轮次的端到端演进（$G_0 \to G_8$），分别列出汇总的 C/H/M 总命中数（涵盖全部 79 个 GT $\times$ 3 运行次数 = 237）及其净增益、等权项目平均召回率（与图4一致）及其净增益，以及采用决策。

轮次	C/H/M 总命中数（/237）	命中 Δ	每项目平均召回率	召回率 Δ	决策
$G_0 \to G_1$	64 → 92	+28	28.7% → 42.1%	+13.4%	部分采用（保留 3/7）
$G_1 \to G_2$	92 → 116	+24	42.1% → 58.7%	+16.6%	完全采用（保留 3/3）
$G_2 \to G_3$	116 → 136	+20	58.7% → 61.9%	+3.2%	部分采用（保留 2/3）
$G_3 \to G_4$	136 → 139	+3	61.9% → 64.1%	+2.2%	完全采用（保留 2/2）
$G_4 \to G_5$	139 → 147	+8	64.1% → 68.5%	+4.4%	完全采用（保留 3/3）
$G_5 \to G_6$	147 → 153	+6	68.5% → 71.6%	+3.1%	部分采用（保留 2/6）
$G_6 \to G_7$	153 → 173	+20	71.6% → 77.6%	+6.0%	部分采用（保留 2/8）
$G_7 \to G_8$	173 → 182	+9	77.6% → 82.1%	+4.5%	部分采用（保留4/8，最终）

表8： 纯日志和图分析代理在 G_3 和 G_7上的决策，以及重新运行结果（C/H/M 训练在237个总命中中的结果；全合并列是与 G_6 基线配对的考试；重新运行使用10个项目，运行次数为 $\times$ 3）。

轮次	全部合并	纯日志决策	图决策（账本净收益）	重新运行（10 个项目 × 3 次运行）
$G_2 \to G_3$	116 → 122	全部回滚	弃用威胁模型重写（−14）；保留问题规划器（+18）和条件推导（+2）	122 → **136**
$G_6 \to G_7$	147 → 154	全部采用	弃用比较重写和单元合并（两者均为 0；它们破坏了 2 个稳定性为 3/3 的命中，并将 1 个关键漏洞从 2/3 降至 0/3）；保留目录建模（+6）和函数评估 (+1)	已剪枝：173；受损所有命中均已恢复

奖金池为50,000美元，所有严重程度中共有72个已确认并计分的高/中严重性集群，以及452个已确认集群；项目和平台名称不予公开）。由于 LucidScan 是一种上游发现工作流，实时提交还结合了依据竞赛规则进行的范围筛选、去重和概念验证。提交的84项发现中，有78项被确认有效（有效率为92.9%，6项被拒绝），涵盖74个不同集群（总排名第2）。如表9所示，该工作流发现了8个已确认高严重性问题中的6个（75.0%，总体并列第1），以及64个中严重性问题中的28个（43.8%，总体并列第2），共计发现72个计分的高/中严重性集群中的34个（47.2%，总排名第2），并在最终排行榜中总排名第3（7.24分，1,136美元）。

匿名案例研究。 一项已确认的高严重性发现（仅有896名研究人员中的13名报告）涉及一个附加到借贷支持型流动性池的自动范围再平衡钩子。由于该钩子允许在单笔交易中不受限制地重复触发再平衡，且未限制累积状态漂移，任何无特权交换者都可以反复穿越触发范围，在10笔交易内耗尽流动性提供者头寸价值的14%–34%；主网分叉概念验证已证实这一点。要发现这一缺陷，必须跟踪交换回调中的多步状态更新以及再平衡在检查重复边界跨越的同时进行模块……，与跨单元交互和数值边界步进相匹配，并演化为 G_8。

表9： 匿名线上公开审计竞赛的结果。该竞赛于2026年11月结束（896名参赛研究人员，奖金池为50,000美元；项目和平台名称不予披露；排行榜总排名第3，得分为7.24分）。

严重程度	找到	总计	召回率	排名 (896)
High	6	8	75.0%	并列第1
中	28	64	43.8%	并列第2
高/中评分	**34**	**72**	**47.2%**	**总排名第2**
Low	24	151	15.9%	总排名第3
Info	16	229	7.0%	并列第6
全部已确认簇 74		**452**	**16.4%**	**总排名第2**

总结（RQ4）。 在一场于2026年11月结束、共有896名研究人员参与的匿名线上公开审计竞赛中，演化后的工作流在高严重性召回率方面并列第1（6/8，75.0%）；在计分的高/中严重性集群方面排名第2（34/72，47.2%），有效率为92.9%（78/84）；总体排名第3。

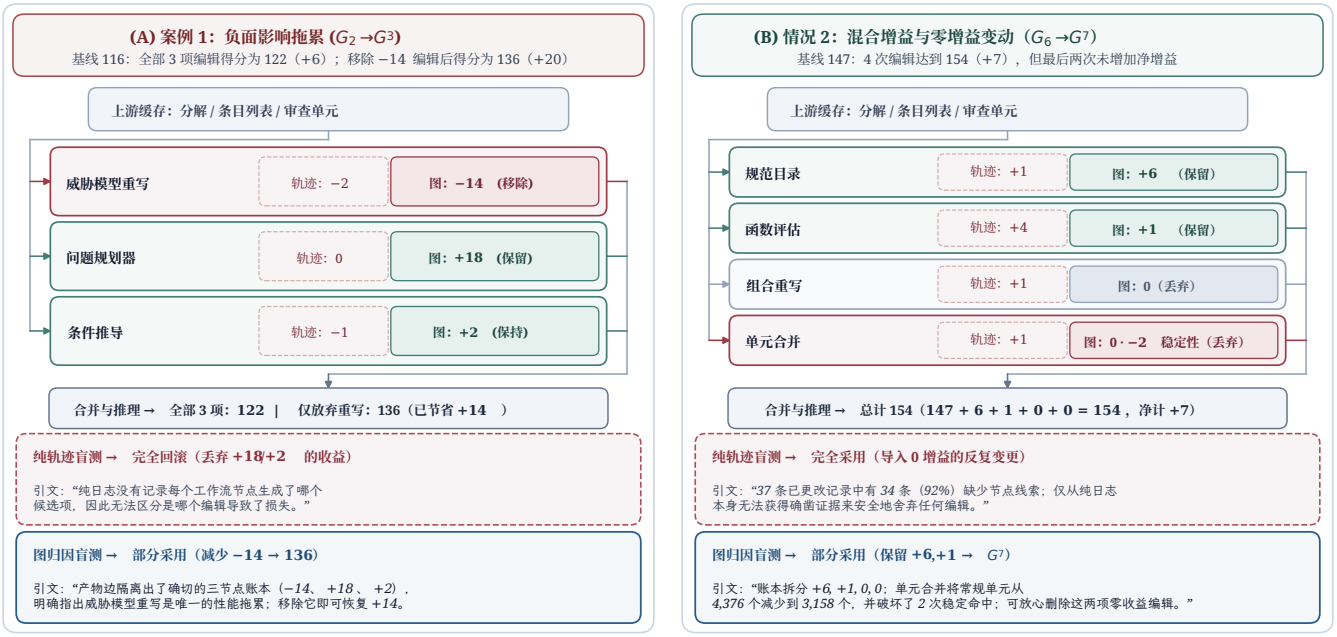

图7： 在两个孤立的部分采用轮次中，基于架构的图归因（精确节点台账）与纯迹线归因（基于文本语义的猜测）之间的比较：(A) $G_2 \to G_3$ 处于负向拖动下，以及 (B) $G_6 \to G_7$ 处于混合增益下。每个面板均展示逐节点信用徽章，以及两名孤立分析代理的逐字引文和采用决策。

7 相关工作

我们的工作与三条研究路线相关：基于大语言模型的智能合约审计、代理自构建与工作流演化，以及运行时安全代理适应。

基于大语言模型的智能合约审计。 GPTScan [12] 将大语言模型与静态分析相结合，以匹配预定义的漏洞类型。PropertyGPT [8]利用源代码检索引导大语言模型生成安全属性，用于形式化验证。KnowDit [5] 从历史审计报告中构建知识图谱，以生成针对特定项目的审计规范和概念验证测试框架。Shuttle 并未附加历史漏洞知识库或生成针对特定项目的规范，而是在离线环境中演进上游发现工作流的结构和通用审查步骤，然后依据真实根因召回率评估冻结后的工作流。

智能体自构建与工作流演化。 ADAS[3]和 DGM [16] 在代码空间中搜索智能体实现，而 AFlow [17] 通过基于端到端执行分数的蒙特卡洛树搜索来优化工作流。Harness?[15] 根据端到端任务评估运行"提议–探测–组合"循环，AgentFlow [6]则根据目标运行时反馈诊断并重写类型化代理图。在漏洞检测领域，EvoHunt [13] 和 MetaVul [1] 通过多代理反思或成对易受攻击函数与修补函数反馈来演化工作流和程序性知识。这些方法主要根据端到端分数或完整运行日志更新工作流。相比之下，Shuttle 扩展实例级工件读写依赖图，根据阶段级瓶颈生成提案，同时隔离项目源代码与 GT，并利用节点级溯源筛选合并后的编辑账本和本地重新运行。

安全智能体演化与受约束的自我改进。 CyberEvolver [2] 在执行时针对每个目标开展种群演化，用于夺旗赛和渗透测试。RRSI [14] 通过提案约束、特定案例筛选和结构剪枝来抑制自我改进中的过拟合。Shuttle 面向跨项目智能合约审计，结合通用性盲审、双向归因，以及基于运行依赖图的保护集验证。

8 讨论与局限性

我们的评估有三个主要局限：

1.**大语言模型评审器。** 归因使用 Sol medium，根因匹配依赖自动生成的 Sol high 判定。第 6.1 节报告了针对合约源代码、对 60 项分层抽取的根因判定进行的独立异构模型审计（60 项中有 58 项一致，$\kappa = 0.933$）；由人类安全专家开展更大规模的盲标注仍有待未来完成。

2. **单条完整演化轨迹。**每个冻结版本运行三次，以衡量单次审计抽样方差。由于完整演化周期的 API 成本，我们从 G_0 到 G_8运行了一条完整的主线轨迹。

3. **下游过滤。**下游严重性过滤、跨报告聚类和自动化概念验证均不在本文范围内，留待未来工作。

9 结论

我们提出了 Shuttle，这是一种面向智能合约审计的受约束工作流演化系统。通过溯源图归因，Shuttle 可以定位候选线索丢失的位置、追踪现有命中的来源，并利用节点溯源账本过滤合并后的编辑，同时在隔离项目源代码和真实标签的情况下生成通用工作流修订。我们的评估得出四项结论。第一，在 13 个未用于M_{full} 演化的验证项目上，使用相同的基础模型，演化后的 LucidScan G_8 将每项目平均 C/H/M 召回率从 G_0 的 28.6% 提升至 70.3%，超过采用默认设置的 Pashov 和 Codex Security；每个验证项目每次运行的直接 API 成本为 4.92美元。第二，约束较弱的试点 M_{weak} 在未进行通用性盲审的情况下，开发集召回率达到 88.5%，但每项目扫描令牌增加了 26.6×，验证集偏差则为 31%–96%。第三，在早期轮次消融窗口中，完整配置的表现持续提升，而所有三种无图消融实验的表现均有所退化；在 G_3 和 G_7中，账本移除负面或损害稳定性的编辑后，总命中数或稳定命中数得以恢复。第四，在一场有 896 名研究人员参与、于 2026年11月结束的匿名线上公开审计竞赛中，演化后的扫描器在高严重性召回率方面并列总排名第 1（6/8，75.0%），在计分的高/中严重性集群方面总排名第2（34/72，47.2%），提交有效率为 92.9%（78/84），最终总排名第3。

参考文献

[1] Ira Ceka、Arnav Garg、Ryan Lutz、Ting Zhang、Ratnadira Widyasari、Gail Kaiser 和 Baishakhi Ray。《MetaVul：用于漏洞检测的智能体框架发现》，2026 年。arXiv:2610.40721。[2] Yihe Fan、Changyi Li、Lichen Xu、Xudong Pan、Jiarun Dai、Hong Geng 和 Min Yang。《CyberEvolver：面向网络安全智能体的即时结构化自我演化》，2026 年。arXiv:2605.26195v2。[3] Shengran Hu、Cong Lu 和 Jeff Clune。《智能体系统的自动化设计》。载于 国际学习表征会议（*ICLR*），2025 年。[4] Tushar Khot、Harsh Trivedi、Matthew Finlayson、Yao Fu、Kyle Richardson、Peter Clark 和 Ashish Sabharwal。《分解式提示词：解决复杂任务的模块化方法》。载于 国际学习表征会议（*ICLR*），2023 年。[5] Ziqiao Kong、Wanxu Xia、Chong Wang、Yi Lu、Pan Li、Shaohua Li、Zong Cao 和 Yang Liu。《Knowdit：结合审计知识总结的智能体智能合约漏洞检测》，2026 年。arXiv:2603.26270。[6] Hanzhi Liu、Chaofan Shou、Xiaonan Liu、Hongbo Wen、Yanju Chen、Ryan Jingyang Fang 和 Yu Feng。《用于漏洞发现的多智能体框架综合》，2026 年。arXiv:2604.20801。[7] Nelson F. Liu、Kevin Lin、John Hewitt、Ashwin Paranjape、Michele Bevilacqua、Fabio Petroni 和 Percy Liang。《迷失于中间：语言模型如何利用长上下文 . 计算语言学协会会刊（*TACL*），12:157–173，2024。

[8] Ye Liu、薛越、Daoyuan Wu、Yuqiang Sun、Yi Li、Miaolei Shi 和 Yang Liu。PropertyGPT：通过检索增强属性生成，由大语言模型驱动的智能合约形式化验证。载于第*32*届网络与分布式系统安全年度研讨会（*NDSS*），2025年。

[9] Ming Ma、Jue Zhang、Fangkai Yang、Yu Kang、Qingwei Lin、Saravan Rajmohan 和 Dongmei Zhang。DoVer：面向大语言模型多智能体系统的干预驱动自动调试，2025年。arXiv:2512.06749。

[10] Aman Madaan、Niket Tandon、Prakhar Gupta、Skyler Hallinan、Luyu Gao、Sarah Wiegreffe、Uri Alon、Nouha Dziri、Shrimai Prabhumoye、Yiming Yang、Shashank Gupta、Bodhisattwa Prasad Majumder、Katherine Hermann、Sean Welleck、Amir Yazdanbakhsh 和 Peter Clark。Self-Refine：通过自反馈进行迭代改进。载于神经信息处理系统进展（*NeurIPS*），第36卷，第2019–2038页，2023年。

[11]OWASP基金会。OWASP代码审查指南，版本 2.0。https://owasp.org/www-project-code-review-guide/,2017。

[12] Yuqiang Sun、Daoyuan Wu、薛越、Han Liu、Haijun Wang、Zhengzi Xu、Xiaofei Xie 和 Yang Liu。《GPTScan：结合 GPT 与程序分析检测智能合约中的逻辑漏洞》。载于*IEEE/ACM* 第 *46* 届软件工程国际会议（*ICSE*）论文集，第 1–13 页，2024 年。

[13] Ziyue Wang、Cheuk Wang Maurice Ng、Chenchen Yu、Strick Sheng、Kaihua Qin 和 Liyi Zhou。《面向智能体安全审计的可迁移的自演化操作手册》，2026 年。arXiv:2606.16420。

[14] Peng Xia、Rujun Han、Zifeng Wang、Yanfei Chen、Yufan Zhuang、Yoonho Lee、Chengsong Huang、Han Yu、Zhongy- ing CuiZhu、Yifei Ming、Huaxiu Yao、Burak Gokturk、Tomas Pfister 和 Chen-Yu Lee。《RRSI：智能体框架的正则化递归自我改进》，2026 年。arXiv:2609.24972v3。

[15] Ruiyao Xu、Yanfei Chen、Zhongying CuiZhu、Bhavana Dalvi Mishra、Yifei Ming、Han Yu、Rujun Han、Chen-Yu Lee 和 Tomas Pfister。Harness²：面向开放世界的递归式智能体框架构建，2026 年。arXiv：2610.14224。

[16] Jenny Zhang、Shengran Hu、Cong Lu、Robert Lange 和 Jeff Clune。达尔文-哥德尔机器：自我改进智能体的开放式演化，2025 年。arXiv：2505.22954。

[17]Jiayi Zhang、Jinyu Xiang、Zhaoyang Yu、Fengwei Teng、Xiong-Hui Chen、Jiaqi Chen、Mingchen Zhuge、Xin Cheng、Sirui Hong、Jinlin Wang、Bingnan Zheng、Bang Liu、Yuyu Luo 和 Chenglin Wu。AFlow：自动化智能体工作流生成。在 国际学习表征会议（*ICLR*）论文集，2025 年。

[18] Shaokun Zhang、Ming Yin、Jieyu Zhang、Jiale Liu、Zhiguang Han、Jingyang Zhang、Beibin Li、Chi Wang、Huazheng Wang、Yiran Chen 和 Qingyun Wu。哪个智能体导致任务失败，以及何时导致失败？关于大语言模型多智能体系统的自动化失败归因。在第 *42* 届国际机器学习会议（*ICML*）论文集中，第 267 卷，第 76583–76599 页，2025 年。

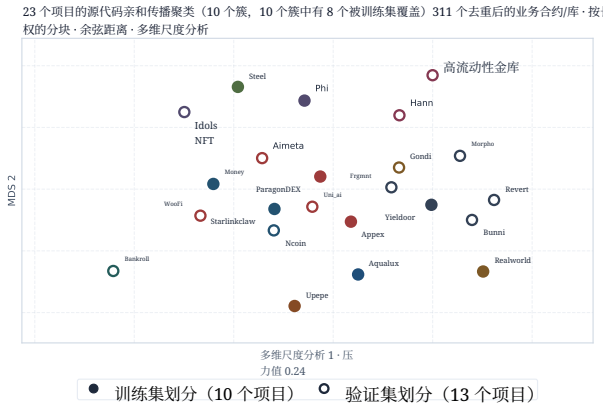

图8： 对23个基准项目（311个去重后的合约和库，分为10个簇）进行亲和传播聚类。实心圆表示10个训练项目，空心圆表示13个验证项目，不同颜色表示不同簇。

[19] Kunlun Zhu、Zijia Liu、Bingxuan Li、Muxin Tian、Yingxuan Yang、Jiaxun Zhang、Pengrui Han、Qipeng Xie、Fuyang Cui、Weijia Zhang、Xiaoteng Ma、Xiaodong Yu、Gowtham Ramesh、Jialian Wu、Robert Tang、Zicheng Liu、Heng Ji、Pan Lu、James Zou 和 Jiaxuan You。大语言模型智能体在哪里失败，以及它们如何从失败中学习，2025 年。arXiv：2509.25370。

A 可复现性细节

A.1 编辑范围与可复现性制品

节点内容编辑会影响提示片段、字段和输出合约。图编辑会添加或移除节点、连接或执行策略。所有编辑都需要进行依赖项、架构和最终输出兼容性检查。

每轮均保留应用快照、提案输入、候选编辑、审查记录、合并清单、扫描记录和盲标映射。阶段级流程瓶颈摘要和经验日志均与具体版本关联。最终的 G_8 纳入了跨单元关系矩阵、具体边界评估、文档与函数预期对齐，以及关键表达式和极端值的明确正向传递。

A.2 验证项目与重复实验

验证项目包括 Aimeta、Bankroll、Bunni、Frgmnt、Gondi、Hann、Idols NFT、Morpho、Ncoin、Revert、Uni_ai、高流动性金库和 WooFi。图8展示了23个基准项目（311个去重后的合约和库）的二维亲和传播聚类。在验证集中，配对基线 G_0 和最终 G_8 均运行三轮，作为主要验证结果；而 G_6 和 G_7 各运行两轮（图4绘制了配对 G_0 基线与匹配的两轮验证轨迹，以及完整三轮 G_8 结果）。主要训练比较针对每个版本各运行三次。严重/高/中等级分母在每次训练运行中为79，在每次验证运行中为114。

A.3 详细 M_{full} 演化轮次

图 9展示了从 G_0到 G_8的完整八轮演进循环，表10列出了从 G_0到最终 G_8的每一轮 M_{full} 演化，下面的内嵌表格给出了 $G_6 \to G_7$在召回率和受保护集稳定性方面候选版本的完整选择记录。

候选项 ($G_6 \to G_7$)	C/H/M 命中 (/237)	保护集 Φ_t 稳定性	规则决策
配对基准 G_6	147	≥ 2-通过：49；3/3 稳定性：38	先前基线
备选四编辑批次	145 (−2)	稳定命中回退	**拒绝批次**（负收益）
图7(B) 4编辑批次	154 (+7)	3/3 已停滞（2 已丢弃 3→2；1 关键项。 2→0)	**剪枝**（丢弃 2 零收益/退化编辑)
已剪枝 G_7 (2 保留)	**173（77.6%）**	**两者均为 3/3 已恢复；关键项。已恢复 2/3**	**采纳为 G_7**

A.4 最终G_8的各项目扫描成本明细

表11列出了最终 G_8 在全部 23 个基准项目中的各项目平均直接 API 扫描成本（每个项目运行 3 次，共扫描 69 次），按未缓存输入每百万令牌 0.10 美元、缓存输入每百万令牌 0.01 美元、输出令牌每百万令牌 0.50 美元计价，不包括离线演化。



图 9： Shuttle 从 G_0 到 G_8 的完整八轮受约束演化循环，未跳过任何轮次（所有命中数和召回率数据均与表7一致）。每一行对应四个阶段：**提议**（工作流拓扑变化：蓝色主干、 + 绿色新增、 Δ 琥珀色编辑、 − 红色虚线表示未采纳/已剪枝）、**执行与观察**（收益与瓶颈）、**评分与剪枝**（召回率 / 命中数和采纳率），以及**图诊断与下一目标**，通过青绿色反馈循环连接至下一轮。

表10： 详细的 M_{full} 演进轮次（G_0 至 G_8）：目标阶段、设计变更、提案规模及采纳结果（对应图4）。

轮次 / 版本	目标阶段	设计理念（通俗表述）	提案与规模	结果
G_0	完整三步基线	基本 OWASP 流程：拆分模块，建模威胁，然后推理	3 个基础节点	起始基线（28.7% 项目 avg）
G_1	分解与威胁建模	先列出入口点，再查找缺陷；添加后续问答通道	提议了 7 个，保留了 3 个（+2 节点，1 提示词）	缓存后保留 3 个对 4 执行留一法剪枝冗余编辑
G_2	全部 3 个阶段	将相关函数归入审查单元并添加公式推导	提议 3 个，保留 3 个（+1 节点，单元/判定）	已采用（+30.0 页，超过 G_0 累计）
G_3	威胁建模	明确规划补充问题并逐步解决条件	提议了 3 个，保留了 2 个（+2 个节点，1 个已剪枝）	剪枝后已采用的有害建模改写
G_4	威胁建模	比较相关代码位置并排比较；分别起草并去重	提议 2 个，保留 2 个（+1 节点，去重编辑）	已采用；小幅稳定增益
G_5	威胁建模	再次检查比较结果并进行验证针对问题的补充答案	提议 3 个，保留 3 个（+2 个节点，1 范围）	已采用（+4.4 页，超过 G_4）
G_6	分解与建模	基于未分组函数恢复结合调用端点配对和期望引导的问答	9 个属性，测试了 6 个，保留了 2 个（+2 项编辑）	已采用（训练集平均值 71.6%，并集 75.9%）
G_7	分解 & 建模	构建项目规范/参考目录，并评估围绕关键函数的语句	12 个属性，其中 8 个在两轮中已批准 4编辑批次，保留2个（+1 节点，1个提示词）	舍弃 -2 个批次及2个来自 +7 个批次的零收益编辑；重新运行（10个项目 × 3 次运行）达到77.6%，并恢复了全部失效的稳定命中
G_8	全部 3 个阶段	添加关系矩阵、具体边界替换、文档期望对齐 & 边界参数传递	8 项已批准，4 项保留（+2 节点，2 交接/列表）	最终版本（G_8，训练集平均值为70.3%）

表11： G_8 在全部23个基准项目中的每个项目的平均扫描成本（每个项目运行3次，共扫描69次；美元数值根据记录的运行令牌日志计算，包括重试，但不包括离线演化）。

训练项目（3次运行）	平均成本（美元）	项目估值（3r）	平均成本（美元）	项目估值（续）	平均成本（美元）
Appex	3.42	Aimeta	3.67	Morpho	3.02
Aqualux	1.97	Bankroll	0.97	Ncoin	4.18
Money	1.25	Bunni	7.46	Revert	4.68
ParagonDEX	5.09	Frgmnt	7.10	Uni ai_	6.01
Phi	2.95	Gondi	5.48	高流动性金库	3.68
Realworld	3.82	Hann	11.71	WooFi	2.98
Starlinkclaw	3.83	Idols NFT	3.08	—	—
Steel	1.18	**验证集平均值（39 次扫描）**	**4.92（总计 192.1）**	**总体（69 次扫描）**	**3.97（总计 273.9）**
Upepe	1.31				
Yieldoor	2.45				
训练集平均值（30 次扫描）	**2.73（总计 81.8）**				