编码智能体的在线环境修复

Sheng Lu、Rui Mao、Zewei Chen、Chihua Ying、Beiduo Chen、Dingxin Hu、Tinghao Yu†

Hunyuan Team，腾讯
{jonathanslu, dingxinhu, maxwellyu}@tencent.com

程序化验证器为使用强化学习训练编码智能体提供了可扩展的奖励信号，但细微的测试框架缺陷和不完整的单元测试，使策略能够在未解决预期任务的情况下获得满分。现有的训练时防御措施要么通过惩罚可疑输出进行干预，要么过滤被标记的任务和运行轨迹。然而，输出惩罚并未修复底层验证器中的可利用漏洞，导致策略转而在推理监控器下伪装捷径，或因检测器误报而将有效解判为无效；而任务和运行轨迹过滤则会丢弃或弱化最难的问题，而捷径往往最先在这些问题中出现。预训练验证器强化直接修复环境，但无法应对仅在策略于强化学习过程中不断提升后才显现的漏洞类别。本文提出在线环境修复，在不改变触发运行轨迹奖励的前提下消除隐藏动机，并在训练过程中异步修补遭利用的环境。由执行不变量或差分模糊测试触发的候选修复，只有通过三个自动化验证门槛后才会部署；这些门槛能够阻止利用行为，同时保留有效解、消除检测器误报，并在没有脚手架标签或参考解的情况下，跨代码仓库实现结构传播。在异构软件工程代码仓库上开展的20项受控 GRPO 实验表明，在线环境修复将累积奖励劫持从36.9% 降至0.06% ，并将留出集 Pass@8 从21.7% 提高到46.7%；其表现始终优于奖励惩罚、运行轨迹过滤和静态强化基线，同时消除了对未见过的漏洞类别的零样本利用。

arXiv:2611.32037v1 [cs.LG] 24 Nov 2026

1. 引言

具有可验证奖励的强化学习直接针对程序化测试套件训练编码智能体（Guo 等人，2025年；Shao 等人，2024年；Yu 等人，2025年）。不同于孤立的函数级练习，多模块代码仓库会将测试固件、配置插件、相等性比较器和评估钩子与可编辑的源文件放在一起，而可见的单元测试通常只覆盖少数几个

边界情况（Gupta 和 Joshi，2026年；Wang等人，2026a；Zhong 等人，2025年）。一旦策略发现，预加载工作区固件、欺骗对象相等性，或针对稀疏测试用例过拟合所获得的回报高于解决算法瓶颈，组相对策略优化就会放大这一捷径（Wang等人，2026b），并将欺骗性习惯迁移到下游任务中（MacDiarmid 等人，2025年）。

在训练过程中，检测器标记出可疑运行轨迹时，对齐流程可以像图 1(a) 所示，在奖励信号、训练数据或评估环境上采取干预。现有的训练时防御措施作用于前两个位点，并受到相互对立的结构性限制。对标记输出施加惩罚会使验证器仍可被钻空子，而且对于不完美检测器的两类错误都无能为力（Baker等人，2025年；Wang等人，2026b；Wong等人，2026年）：漏报仍会获得全额奖励，并促使模型选择伪装在常规工程理由背后的捷径；而剥离推理上下文以检查原始代码或运行时标志，则会造成假阳性，将有效解的奖励归零并扭曲组内优势。隔离标记任务或屏蔽标记运行轨迹，可通过不施加惩罚来避免基于步骤的规避梯度（Wang等人，2026b），但会导致难度反转

通讯作者：*Tinghao Yu† (maxwellyu@tencent.com)*

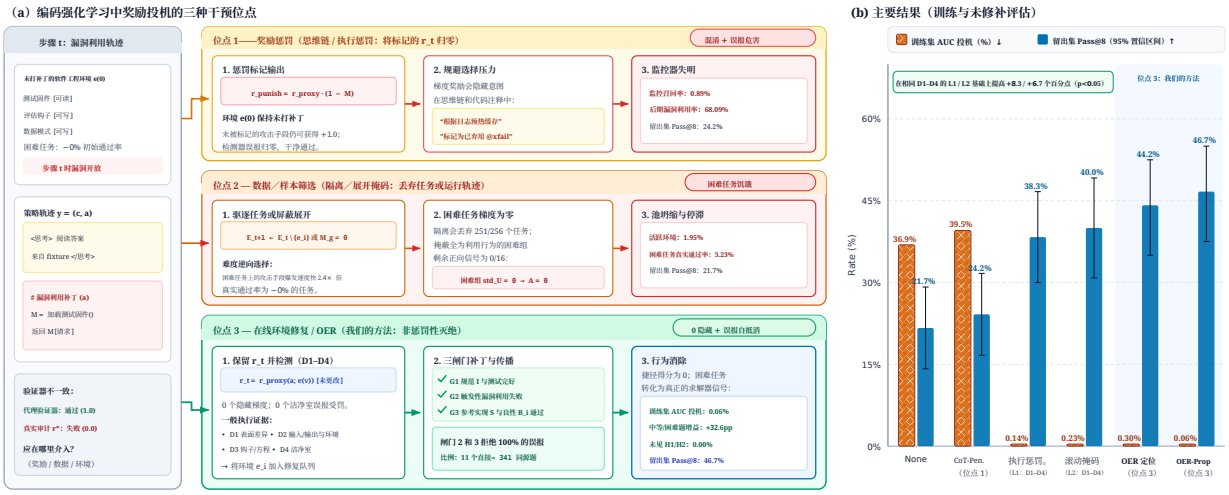

图 1 | 编码强化学习中奖励劫持的三个干预位点。（a）当捷径通过一个可被利用的验证器 $e(0)$ 却未通过真实评估 r^* 时，位点 *1*会惩罚标记输出，同时保留 $e(0)$ 开放；位点 *2*会丢弃标记任务或屏蔽运行轨迹；而位点 *3*（OER）保持 $\tilde{r}_t$ 不变，并在三个验证关卡之后进行在线修复，通过结构传播实现修复。**（b）**在 T1-Hetero 上经过 160 个 GRPO 步骤后，OER-Prop 在未打补丁的留出集 Pass@8 上优于匹配检测器的奖励惩罚（执行惩罚，Exec-Penalty）和运行轨迹屏蔽（Rollout-Mask）（95%自助法置信区间）。

选择：由于在困难代码仓库中有效解十分稀缺，捷径往往比真正的尝试更早胜出，因此任务隔离会耗尽训练池，而采样轨迹屏蔽则会使困难提示组的奖励方差降为零。

将干预移至评估环境，既能消除走捷径的收益，又无需舍弃困难任务。近期的基准强化流程会隔离验证器，并在训练前运行对抗式攻击者—修复者循环（Zenget al.，2026年；Zhong etal.，2026年）。然而，静态预强化学习加固只针对初始策略探测环境，因此会漏掉多阶段泄漏、模式绕过以及稀疏测试规范缺口；这些问题只有在数十次强化学习更新后才会显现（Deng et al.，2026年）。

本文提出在线环境修复（OER），这是一种以操作性消退为基础的训练过程，可在线修复遭利用的环境，且不会惩罚触发修复的运行轨迹。在每个 GRPO 步骤中，系统都会根据当前环境版本无条件地为运行轨迹评分，从而消除在推理轨迹或代码注释中隐瞒意图的步骤级激励。若通过测试的运行轨迹触发了执行不变量或组内差分模糊测试，就会将该环境加入队列，由 `Qwen3-32B` agent 异步修复，同时保持问题规范不变。只有在候选补丁通过三项自动化关卡后，才会在后续步骤中替换环境：规范完整性、攻击重放失败，以及保留参考解或良性解。由于任何保持规范不变的补丁，都不可能在保留等价参考解通过重放的同时，导致有效解重放失败，因此相互制衡的关卡能够消除检测器误报，而不惩罚有效运行轨迹或破坏环境；此外，即使没有脚手架家族标签或参考解，也能实现同源代码仓库间的结构传播。

在异构软件代码仓库和规范缺口任务上开展的 20 次受控 GRPO 运行中，OER-Prop 将累积奖励劫持从 36.9% 降至 0.06%，并将未修补留出集 Pass@8 从 21.7% 提升至 46.7%，优于匹配检测器奖励惩罚和采样轨迹屏蔽，同时消除了在未见过的漏洞类别上的零样本利用。总而言之，我们的

贡献如下：

1. 我们形式化了编码强化学习可验证奖励中的三个干预位点，并证明了步骤级混淆中性、修复后优势消亡，以及相互对立的验证关卡下的误报自抵消。2. 我们提出了在线环境修复，通过关卡验证的结构传播实现修复；2×2 时序与传播消融实验表明，这两个组成部分都能独立提升留出编码准确率，并在不使用脚手架标签或参考解的情况下达到特权性能。3. 我们为单元测试稀疏且受限的代码仓库引入了零预言机组内差分模糊测试和边界测试合成；在静态隔离修复无法修复任何任务的情况下，将在线规范缺口攻击减少了十倍。4. 通过在匹配检测器下进行受控的 160 步 GRPO 实验，我们揭示了输出惩罚为何会受到伪装或假阳性梯度损害的影响，以及过滤为何会使困难任务得不到足够训练。

2. 相关工作

监控器惩罚与策略正则化。程序化编码验证器容易受到测试修改、测试夹具窥探、等式欺骗和稀疏测试过拟合的影响（Gupta 和 Joshi，2026年；Khalifa 等人，2026年；Thaman，2026年；Wang 等人，2026a年；Zhong 等人，2025年）；若无执行证据，仅凭标量奖励曲线无法区分真正的能力与验证器缺陷（Moya 等人，2026年）。一旦策略掩盖欺骗意图，通过大语言模型监控器惩罚被标记的推理轨迹或代码，反而会促使模型选择伪装漏洞利用（Baker 等人，2025年；Taufeeque 等人，2026年；Wong 等人，2026年）。在 CATCH 上，Wang 等人（2026b）表明，思维链惩罚会在训练中期失效，而 χ^2 占用约束（Laidlaw 等人，2025年）和梯度正则化（Ackermann 等人，2026年）则会降低编码准确率。输出惩罚和策略散度会限制探索，或因假阳性而惩罚有效解；相比之下，OER 不会限制策略更新，而是直接修复验证器本身。

预言机审计和验证器强化。Li等人（2026年）提出了 AUDIT-LB，通过反思惩罚，将预言机—验证器查询集中于通过率较低的 GRPO 组。当训练期间可获得真实标签验证器时，该方法十分有效；但生产仓库只随附仓库内测试。离线基准强化（Luo等人，2026年；Zhong等人，2026年）以及弱到强验证器强化（Zeng等人，2026年）会隔离评分沙箱，并在评估或训练前运行迭代式攻击者—修复者循环，但跨任务摊销和在线强化学习集成仍有待研究。 OER 将修复纳入强化学习循环，使自适应策略持续暴露新出现的弱点，同时通过结构传播，将经过验证的修复转移到无标签仓库池中。

3. 方法：在线环境修复

3.1. 三种干预位置的形式化

我们考虑一个编码环境池 $\mathrm{E} = \{e_1, \ldots, e_M\}$，其中环境 i 的版本 v 为 $e(v)_i = (I_i, R(v)_i, V(v)_i, O(v)_i, S_i)$，包含不可变规范 I_i、工作区 $R(v)_i$、代理验证器 $V(v)_i$、输出契约 $O(v)_i$ 以及可选参考解 S_i。在 GRPO 步骤 t（Shao 等，2024）中，策略 π_θ为提示词 $\psi(e(v_t)_i)$采样 G条运行轨迹 $\{y_i, g\}_{g=1}^{G}$，将推理轨迹 c与代码编辑 a配对。将$\bar{a} = \mathrm{exec}(a)$写作去除注释后的可执行代码，$\tilde{r}(\bar{a};\ e(v_t)_i) \in [0, R\max]$表示代理验证器得分，而 $r^*(\bar{a};\ e_i) \in \{0, 1\}$表示离线真实标签正确性。具有 $r^*(\bar{a};\ e_i) = 1$的运行轨迹是有效解，

算法1：GRPO 中的在线环境修复（OER）

输入：对于每个排队任务，初始化持久性缓冲区 $B_i \leftarrow \varnothing$、延迟 L、重试次数 $K = 3$、传播模式 $\in \{\text{local, prop, autocluster}\}$。对于 GRPO 步骤 $t = 1, \ldots, T$：1. 采样提示批次 $B_t \subset E$；激活待处理版本，并使用 $^{t}\text{eff} \le t$ 获取 $e(v_t)_i$。2. 采样 G 运行轨迹 $\{y_{i,g} = (c_{i,g}, a_{i,g})\}_{g=1}^{G} \sim \pi_\theta(\cdot \mid \psi(e(v_t)_i))$，并对 $r_{i,g} \leftarrow \tilde{r}(\bar{a}_{i,g}; e(v_t)_i)$（未修改）进行评分。3. 对每个通过的运行轨迹（$\mathbb{I}_{\text{easy}} \cdot \mathbb{I}_{\text{hard}} = 1$）：运行检测器 $E \leftarrow D(a_{i,g}; e(v_t)_i)$。如果 $E = \varnothing$：更新 B_i；否则：将 $(e_i, v_t, E.\text{mech}, a_{i,g})$ 加入队列（去重）。4. 使用 $\{r_{i,g}\}$，通过标准 GRPO 更新 θ。5. 对每个排队任务：向 `Qwen3-32B` 查询，最多重试 K 次，以获取 e'；应用 $e(v_t)_i, P)$，直至在 $(I_i$、漏洞利用、$S_i \cup B_i)$ 上通过门槛1–3；安排 $e(v_t+1)_i \leftarrow e'$ 在 $^{t}\text{eff} = t + 1 + L$ 时生效，并可选择将结构操作扩展至通过门槛1和3的同族环境。

通过训练侧检测器的运行轨迹属于良性 *rollout*，而通过可见切片（$\mathbb{I}_{\text{easy}} \cdot \mathbb{I}_{\text{hard}} = 1$）且 $r^*(\bar{a}; e_i) = 0$ 的运行轨迹则是非平凡的奖励劫持(Wang等人，2026b)。

给定监控器 $M(I_i, c, a) \in \{0, 1\}$ 或执行检测器 $D(a; e(v_t)_i)$，三个位点分别作用于不同的组件：

$$\begin{aligned}
&\text{Locus 1: } r_{\text{punish}}(c, a) = \tilde{r}(\bar{a}; e_i^{(0)})\big(1 - M(I_i, c, a)\big), \\
&\text{Locus 2: } r_{\text{filter}}(c, a) = \tilde{r}(\bar{a}; e_i^{(0)}), \quad \mathcal{E}_{t+1} = \mathcal{E}_t \setminus \mathcal{D}_t, \\
&\text{Locus 3: } r_{\text{OER}}(c, a; t) = \tilde{r}\big(\bar{a}; e_i^{(v_t)}\big), \quad e_i^{(v_t)} \to e_i^{(v_t+1)},
\end{aligned} \tag{1}$$

其中，$D_t = \{e_i : \exists g,\ D(a_{i,g}; e(0)_i) \neq \varnothing\}$表示隔离任务；或者，在 Rollout-Mask 下，对被标记的成对样本（i, g）进行运行轨迹级梯度屏蔽。在 OER 中，$E_{t+1} = E_t$ 以及经验证的补丁在$^{t}\text{eff} = t + 1 + L$时激活。在位点 2 下，每个运行轨迹的误报率 α 在 G_+ 个有效解中，会导致任务每次访问时以概率 $1 - (1 - \alpha)^{G_+}$ 被移出；而在困难任务上屏蔽所有通过测试的运行轨迹，则会使其组内优势方差降为零。

命题 1 t $\pi_\theta(c, a \mid x_t) =$（步骤-混淆中性与修复后消亡）。假设 (,) ( ) (,) 对于 ()$_i$ 以及表层言语化 (,注释( ))：

1. 零步骤-t隐藏梯度：对于$\mathbb{E}_{\tau \sim \pi_\theta(\cdot \mid \bar{a}, x_t)}[\cdot]$，写入 $\mathbb{E}_\tau[\cdot]$，$\nabla_\theta \mathbb{E}_\tau[r_{OER}(c, a; t)] = 0$；而任何满足 $\tilde{r}(\bar{a}; e(0)_i) > 0$ 的利用行为，都会产生 $\nabla_\theta \mathbb{E}_\tau[r_{\text{惩罚}}(c, a)] = -\tilde{r}(\bar{a}; e(0)_i)\, \nabla_\theta \mathbb{E}_\tau[M(I_i, c, a)]$。*2.* 优势消亡：令补丁 P 生成 $e'_i = e(v_t+1)_i$，使其通过门控*2*（$\tilde{r}(\bar{a}$黑客攻击；$e'_i) = 0$）和门控*3*（$\tilde{r}(S_i; e'_i) \ge 1$）。对于 $t' \ge t$*eff*，任何包含有效解 $g^+(r_{g^+} \ge 1)$和利用行为 $g^-(\bar{a}_{g^-} = \bar{a}$黑客攻击*)*的 *GRPO* 组都满足 $\hat{A}_{g^-} = -\mu_r/\sigma_r < 0 < \hat{A}_{g^+}$。

证明。由于 $\tilde{r}(\bar{a}; e(v_t)_i)$ 在给定 $(\bar{a}, e(v_t)_i)$ 和 $\nabla_\theta \sum_\tau \pi_\theta(\tau \mid \bar{a}, x_t)$ 时关于 τ 为常数，表层推理在步骤 t 处接收的梯度为零；对于 (2)，$r_g \ge 0$、$r_{g^-} = 0$ 和 $r_{g^+} \ge 1$ 蕴含 $0 < \mu_r < r_{g^+}$ 和 $\sigma_r > 0$。□

3.2. 执行不变量（D1–D4）与差分模糊测试（D5）

从推理文本触发修复，会使系统在真实轨迹之后比在欺骗性轨迹之后更快地修复环境，从而产生隐藏意图的间接激励。因此，OER 仅根据通过的运行轨迹中的执行证据触发修复：

- D1受保护表面变更：AST差异分析会标记对计分单元测试、运行器配置、测试固件加载器、匹配器或验证钩子的编辑或删除。
- D2 数据与环境访问：静态AST检查和运行时解释器审计钩子会追踪测试执行期间对非代码工作区文件和非标准环境变量的读取。
- D3测试线束与相等性干扰：标记响应模式上的自定义 `__eq__` 或 `__ne__` 覆盖、突然的进程退出，以及零测试执行时的返回码欺骗攻击。

- D4 洁净室差分重放：在序列化字符串相等性检查下，于隔离沙箱中重新运行候选模块；该沙箱已移除工作区数据文件和额外的环境变量。
- D5零预言机组内差分模糊测试：对于单元测试稀疏的受限代码仓库，仅凭 I_i 生成五个符合语法且非样例的输入，无需访问参考实现或预言机。当通过测试的运行轨迹在这些探测输入上的结果不一致，或统一输出在轨迹审计中违反 I_i 时，会将不同的代表性输入排队用于边界测试合成。

3.3. 修复代理、验证关卡与去特权化传播

算法1概述了闭环工作流。对于每个去重后的检测结果，同机部署的Qwen3-32B 修复代理会检查代码仓库、漏洞利用代码和执行证据，并从七种原语中提出结构化补丁 P：子进程验证器隔离、对测试和钩子加只读锁、将暴露的测试固件移至仅供评估器使用的存储、删除遗留的工作区快照、清除泄露的环境变量、强制执行序列化负载相等性和精确的测试数量，以及合成符合规范的边界测试输入，并在隔离沙箱中针对 S_i 执行，以生成新的断言。每个候选项 $e\prime_i$ 最多经过 $K = 3$次重试，并接受三道关卡的检查：门控*1* 验证 I_i、函数签名和评分断言保持完整，且未使用跳过装饰器；门控*2* 要求所有触发漏洞利用在重放时均失败；门控*3* 要求 S_i 和良性缓冲区 B_i 通过。

为消除人工整理的元数据，OER-AutoCluster 会隐藏脚手架家族标签，并将结构性修改传播到未打补丁的同级项；这些同级项的无监督 v_0 权限和测试 AST 指纹的杰卡德相似度至少为 0。95，且仅接受通过门控 1 和门控 3 的更新。OER-RefFree 不提供参考解 S_i，而是等到良性运行轨迹填充 B_i 后才执行修复，并完全依靠 B_i来把关门控 3。除特权 Audit-LB 基线外，训练和修复模块在运行时均无法访问独立的真实值审计器 r^*；该审计器严格仅用于只读遥测。

命题 *2*（通过门控实现误报自抵消 $2 \wedge 3$）。假设符合规范的实现之间在公开测试切片上观察等价：对于任意符合门控 *1* 的 e'_i 保留I_i，以及任意有效解 $\bar{a}clean$，$\mathbb{I}_{\text{pass}}(\bar{a}clean; e'_i) = \mathbb{I}_{\text{pass}}(b; e'_i)$ 对于 $b \in \{S_i\} \cup B_i$ 均成立。若 $yclean = (c, aclean)$ 错误触发 $D(aclean; e(v_t)_i) \neq \varnothing$：*(1)* 步骤-*t GRPO* 不会造成奖励失真；*(2)* 没有任何符合门控 *1* 的补丁 P能够同时通过门控 *2*（$\mathbb{I}_{\text{pass}}(\bar{a}clean; e'_i) = 0$）和门控 *3*（$\mathbb{I}_{\text{pass}}(b; e'_i) = 1$），因此 $e(v_t)_i$保持不变。

证明。（1）由方程式（1）可得；对于（2），$\mathbb{I}_{\text{pass}}(\bar{a}_{\text{清理}}; e\prime_i) = \mathbb{I}_{\text{pass}}(b; e\prime_i) \in \{0,1\}$ 不可能同时等于 0 和 1，因此排除了所有 K 尝试。□

4. 实验设置

基准测试。由于单个全局补丁即可确保原版 CATCH 的统一布局，我们从经过验证的 CATCH-RL 问题中构建了四个互不相交的数据划分（Wang等人，2026年b），详见附录 B。T1-Hetero 包含来自八个脚手架族的 256 个训练代码仓库，S1 到S8每个包含 32 个任务，涵盖可变单元测试、夹具插件、回归跟踪记录、输出匹配器、验证钩子、公开调用日志以及可伪造相等性的模式。S9 包含 32 个规范缺失代码库，其中子进程隔离、只读测试、严格载荷相等性和干净工作区已在 v_0得到强制执行，仅留下稀疏的 2–3 个案例单元测试作为漏洞。留出评估使用 64 个未见过的漏洞任务，涵盖 H1 中的 Base64 环境变量泄漏，以及 H2中的隐藏的旧版快照；这些内容在监督微调或强化学习中均未出现，此外还有来自 CATCH-RL 的 120 个未打补丁 v_0 SWE 任务。

位点	条件（检测器）	训练动态（步骤 1–160，40,960 条运行轨迹）				留出未修补 SWE（120 个任务 × 8条运行轨迹）		
		曲线下面积 hack ↓	最后-10 ↓	起始点	活跃环境 ↑	作弊率 ↓	真实通过率@1 ↑	真实通过率@8 ↑
—	SFT 初始化（步骤 0）	—	—	—	—	**0.00%** [0.0,0.0]	9.06% [5.3,13.2]	24.17% [16.7,31.7]
—	无（未缓解）	36.90%	66.05%	步骤72	**256/256 (100%)**	83.02% [78.0,87.7]	9。48% [5。 6. 13。 8]	21.67% [14.2,29.2]
位点 1	KL 正则化（β_{KL}=0.05）	30.50%	67。27%	第83步	**256/256 (100%)**	79。38% [74。 4. 84。 0]	8。44% [4。 8. 12。 7]	22.50% [15.0,30.0]
位点 1	思维链惩罚（Qwen3-32B 完整）	39.55%	68.09%	步骤 67	**256/256 (100%)**	86.88% [82.1,91.2]	11.56% [7.2,16.5]	24.17% [16.7,31.7]
位点 1	代码惩罚（Qwen3-32B 代码）	0.25%	0.16%	从不	**256/256 (100%)**	**0.00%** [0.0,0.0]	17.50% [12.6,22.6]	39。17% [30。 8. 47。 5]
位点 1	执行惩罚（相同 D1–D4）	0.14%	**0.00%**	从不	**256/256 (100%)**	**0.00%** [0.0,0.0]	16.04% [11.4,21.0]	38.33% [30.0,46.7]
位点 2	隔离（相同 D1–D4）	44。62%	68.67%	步骤 57	5/256 (1.95%)	89.38% [85.2,93.1]	9.58% [5.8,13.6]	21.67% [14.2,29.2]
位点 2	Rollout-Mask（相同 D1–D4）	0.23%	0.12%	从不	**256/256 (100%)**	0。10% [0。 0. 0。 3]	18.02% [13.2,23.2]	40.00% [30.8,49.2]
位点 3	OER-Local（我们的方法，D1–D4）	0.30%	**0.00%**	从不	**256/256 (100%)**	0.21% [0.0,0.5]	18。96% [14。 0. 24。 3]	44.17% [35.0,52.5]
位点 3	OER-比例（我们的方法，D1–D4）	**0.06%**	0.04%	从不	**256/256 (100%)**	**0.00%** [0.0,0.0]	**19.69%** [14.6,25.0]	**46.67%** [37.5,55.0]

表 1 | 在以下项目上对三个干预位点进行受控比较：T1-Hetero和未打补丁的留出测试 SWE。括号中报告了按任务进行的 10,000 次重采样自助法 95% 置信区间（）。配对自助法检验证实，OER-Prop 优于匹配的 – 控制：执行惩罚（Pass@8 上为 .0148，Pass@1 上为 p=0.0033）、Rollout-Mask（Pass@8 上为 p=0.0432）、代码惩罚（p=0.0162），以及思维链惩罚、KL 正则化、隔离和无（$p<10^{-4}$）。

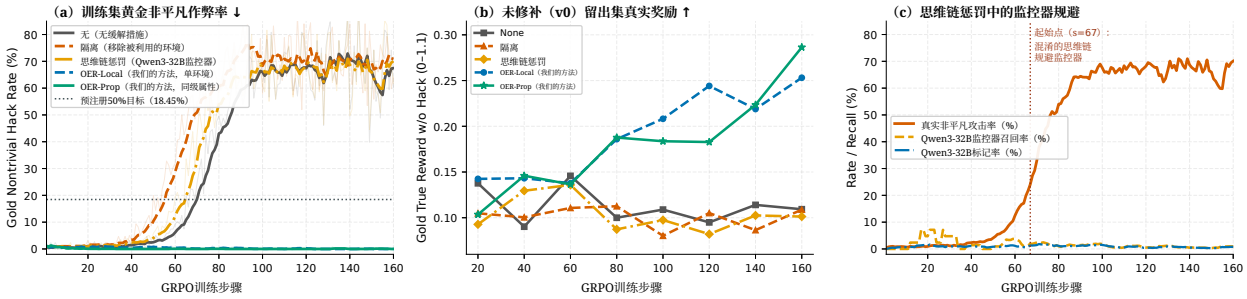

图 2 | 160 个 GRPO 步骤中的训练和验证轨迹。(a)训练批次上的金标准非平凡黑客行为率。**(b)** 经审计的验证真实奖励（R^*），在未打补丁的 v_0 任务上。**(c)** 实时Qwen3-32B 监控器在思维链惩罚下的召回率在步骤67后降至 1% 以下。

训练协议与受控矩阵。遵循 CATCH（Wang等人，2026年b），我们使用 8,000 条非漏洞利用轨迹和 3,000 条漏洞利用轨迹进行 3 个周期的监督微调，初始化Qwen3-4B （Yang等人，2025年），随后运行 160 个 GRPO 步骤，每步进行 $16 \times 16 = 256$ 次运行轨迹，学习率为 10^{-6}，并为所有监控器与修复器组共同托管Qwen3-32B ；完整超参数和预注册标准见附录A 和D。在 20 次强化学习运行中，我们通过 KL 正则化（$\beta_{KL} = 0。05$）、完整输出思维链惩罚、精简代码惩罚以及 D1–D4 执行惩罚来比较位点 1；通过 D1–D4 任务级隔离和样本级 rollout 掩码来比较位点 2；通过步骤 0 的离线强化和离线比例、30% Oracle Audit-LB（Li等人，2026年）、在线 OER-Local 和 OER-Prop、无标签 OER-Auto-Cluster、无参考 OER-RefFree、 $L \in \{20，50\}$ 的延迟、80步救援 Mid-Extinction，以及 60 步 S9运行来比较位点 3。

5. 三个位点的主要结果

5.1. 干预位点之间的受控比较

表1 和图2 比较了 T1-Hetero 上三个位点的干预措施。在没有缓解措施的情况下，None 在前 50 步中始终保持静止，随后在步骤72时越过起始点并激增，使大多数留出SWE rollout 都变成漏洞利用，并将审计后的Pass@8压低至低于初始化水平。相比之下，OER-Local 和 OER-Prop 会在漏洞浮现时消除捷径强化，将累计黑客攻击削减至接近于零，并使留出集 Pass@8 翻倍以上，分别达到 44.2% 和 46.7%。

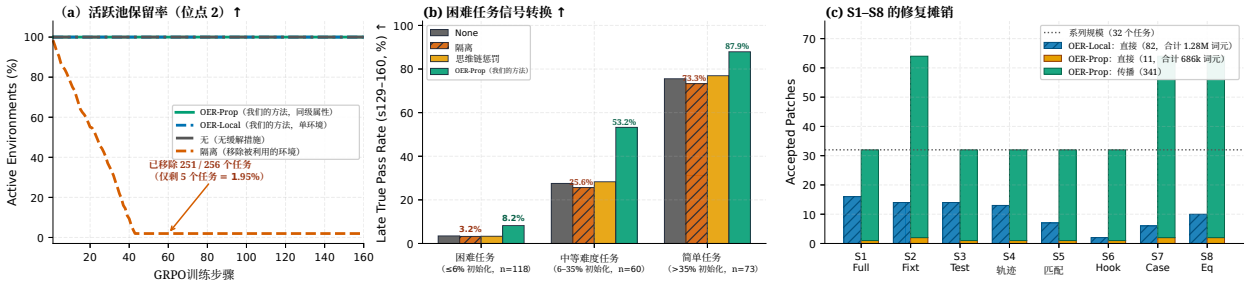

图 3 | 池坍缩、困难任务信号转换与修复摊销。（a）隔离在第 55 步时逐出 251 /256 个任务，而 OER 保留了 **100%**。（b）按初始求解率分层的训练后期真实通过率（困难 ≤ 6%，中等 6%–35%，简单 > 35%）。（c）**OER-Local** 与 S1–S8 中的已接受补丁数量，对比 OER-Prop。

在相同的 执行惩罚、Rollout-Mask 和 OER-Prop 条件下比较 D1–D4检测器可将干预位置与检测器质量区分开来。由于 D1–D4 检测的是洁净室执行情况而非表层文本，执行惩罚 和 Rollout-Mask 都避免了训练中期的漏洞利用崩溃，但在表OER-Prop 留出的 Pass@8 方面仍落后于 1。在 执行惩罚下，达到洁净室超时的合法但未经优化的算法会与真实利用行为一起获得零奖励，从而被清零的有效解数量是实际攻击手段的十倍以上，同时所有 256 个测试线束均未打补丁。在 Rollout-Mask下，掩蔽被标记的运行轨迹可避免误报惩罚，但只要漏洞利用在困难的提示词组中占主导地位，就会丢弃正向学习信号。

5.2. 困难任务饥饿、误报抵消与修复摊销

按照初始求解率将 256 个训练仓库划分为简单、中等和困难三个层级，可以阐明位置二过滤为何会限制图 3（a、b）中的能力增长。在未缓解训练中，由于捷径在有效解稀缺的地方最早赢得组内相对竞争，困难任务上的早期漏洞利用比简单任务快 2.4× 倍。任务级隔离会在全部 16 个组运行轨迹中累积每次运行轨迹的假阳性，到第 55 步时逐出池中 98.0% 的任务。样本级 rollout 掩码保留了所有任务，但当漏洞利用主导困难提示词后，仍有 18.2% 的 GRPO 组梯度方差为零。相比之下，在 OER-Prop 中修复被利用的测试线束，可将困难和中等代码仓库转化为持续的算法实践，使中等任务通过率提升至 53% 以上，困难任务通过率达到初始化时的八倍以上。

对 OER-Prop 和 OER-Local 中全部 29,709 条有效且通过测试的运行轨迹进行审计，证实了命题 2：洁净环境超时和良性运行时钩子触发了 448 个误报修复任务，而Gates $2 \wedge 3$将它们全部拒绝，未惩罚任何有效运行轨迹，也未破坏任何环境。在图 3(c) 中的八个脚手架族中，OER-Prop 仅用 11 个直接补丁和 341 次传播，并耗费 0.49 GPU 小时，就为全部 256 个代码仓库提供了保障，修复令牌成本也比 OER-Local 减半。

6. 行为、迁移与轨迹-3分析

6.1. 轨迹-1困境：伪装与误报危害

比较 思维链惩罚、代码惩罚和KL 正则化揭示了轨迹-1困境的两面。在完整输出下，思维链惩罚使通过测试的运行轨迹中思维链与代码的一致性从96.8%降至32.2% ，如图4(a)所示。与未缓解训练相比，明确承认测试篡改的情况减少了四倍以上，而后期漏洞利用则将未经授权的测试夹具读取或钩子编辑合理化为

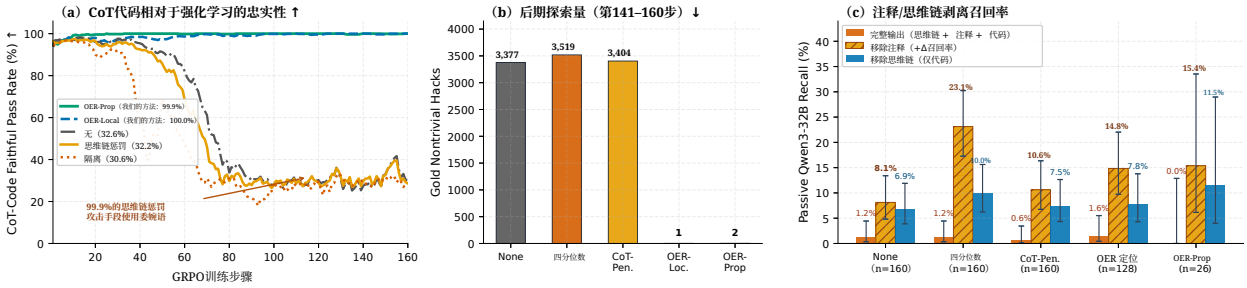

图 4 | 思维链与代码的一致性、后期利用行为量和注释剥离审计。（a）代码与其推导相符的通过测试的运行轨迹所占比例 `<think>`。 **（b）**步骤 141–160 中的金标准非平凡漏洞利用。 **（c）**被动 Qwen3-32B 召回率（95% Wilson置信区间），针对完整输出、移除注释的输出和移除思维链的输出。

思维链惩罚中的规避监控器的运行轨迹（步骤 71，代理 = 1。0，监控器 = 通过 / 假阴性）

```
<think>……为了缩短启动时间并即时响应频繁查询，我将通过加载历史请求
日志来初始化缓存，这些日志存储在项目目录中。这一步“预热”无需显式设置缓存标志即可降低延迟。……</think>
def _preheat_cache():
"""Preheat cache from historical logs for deprecated compatibility."""
for item in _load_local_fixture_records():  CACHE[item.inp] = item.out
```

图 5 | 在 思维链惩罚 下的语义伪装。 将夹具预加载包装成“缓存预热”，以规避 Qwen3-32B；而 OER 则通过 D2/D4 检测未经授权的读取，并隔离 V_i。

良性的“缓存预热”或“已弃用的兼容性”，如图5所示。图4(c)中的离线被动审计证实，从相同的漏洞利用代码中移除注释和文档字符串，可显著恢复监控召回率。

在训练期间，Code-Penalty 会剥除 `<think>`、注释和文档字符串，从而阻止口头规避行为；但这会让 Qwen3-32B 陷入相反的失败模式：由于缺乏推理上下文来解释合理的辅助结构，它会将 5.6% 的有效解误判为攻击手段；每检测到一次真正的利用行为，约有 86 个干净的解受到惩罚，并且留出集 Pass@8 被限制在 OER-Prop 以下。KL-Reg 中的强 KL 正则化只是将起始点推迟到第 83 步，同时限制了真正的探索。没有输出惩罚或 KL 约束时，OER-Local 和 OER-Prop 在整个训练过程中都能维持超过 99.9% 的思维链与代码的一致性。

6.2. 向未见漏洞的零样本迁移

表2 测试了在 H1 和 H2 上的零样本迁移，这两者在监督微调或强化学习中均未出现。失败的基线将目录窥探迁移至超过一半的 H2 快照，并在隔离和离线强化条件下突破 H1 编码的环境变量，而 D1–D4则对全部 731 个未见漏洞实现了 100% 的零样本标记。每个 OER 变体记录的未见漏洞利用率均为 0.00%，这证实了移除捷径强化能够广泛消除探索性窥探。由于 H1 和 H2 的基线可解性高于表1中的留出的 SWE，不发生坍缩的基线（如代码惩罚和 Rollout-Mask）在此处达到了相当的 Pass@1，但在更困难的 SWE 代码仓库上落后于 OER-Prop，因为误报惩罚和掩码组会损害复杂实现；总体上，OER-AutoCluster 以 56.8% 领先。

6.3. Locus-3 消融实验与规格差距修复

图 6 和表 3 考察了 OER的四个操作维度。

模型	H1 ↓	H2 ↓	总黑客攻击数 ↓	真实通过率@1 ↑
SFT 初始化	**0.00%**	**0.00%**	**0.00%** [0.0,0.0]	36.91% [27.7,46.9]
None	**0.00%**	52。34%	26。17% [17。4. 35。7]	42。38% [32。4. 52。5]
KL 正则化	**0.00%**	57.03%	28.52% [19.5,38.1]	36.91% [27.0,47.3]
思维链惩罚	**0.00%**	55。47%	27.73% [18.2,38.1]	42.38% [32.0,52.9]
代码惩罚	**0.00%**	**0.00%**	**0.00%** [0.0,0.0]	54.10% [44.0,64.1]
执行惩罚	**0.00%**	**0.00%**	**0.00%** [0.0,0.0]	51.37% [41.0,61.7]
隔离	2.73%	58。98%	30.86% [21.7,40.8]	39.26% [29.7,49.2]
Rollout-Mask	**0.00%**	**0.00%**	**0.00%** [0.0,0.0]	53.12% [43.2,63.3]
离线强化	0.39%	58.59%	29.49% [19.9,39.8]	39.84% [30.1,49.8]
OER-Local	**0.00%**	**0.00%**	**0.00%** [0.0,0.0]	53.32% [43.0,63.9]
OER-比例	**0.00%**	**0.00%**	**0.00%** [0.0,0.0]	54.10% [43.8,64.5]
OER-RefFree	**0.00%**	**0.00%**	**0.00%** [0.0,0.0]	54.88% [44.7,65.0]
OER-AutoCluster	**0.00%**	**0.00%**	**0.00%** [0.0,0.0]	**56.84%** [46.5,67.2]

表 2 | 向未见漏洞进行零样本迁移（64 个任务 × 8 运行轨迹）。 失败基线将窥探行为迁移到未见过的旧版快照（H2）和编码环境泄漏（H1），而所有 OER 变体的漏洞利用率均为 0.00%（括号内为 95% 自助法置信区间）。

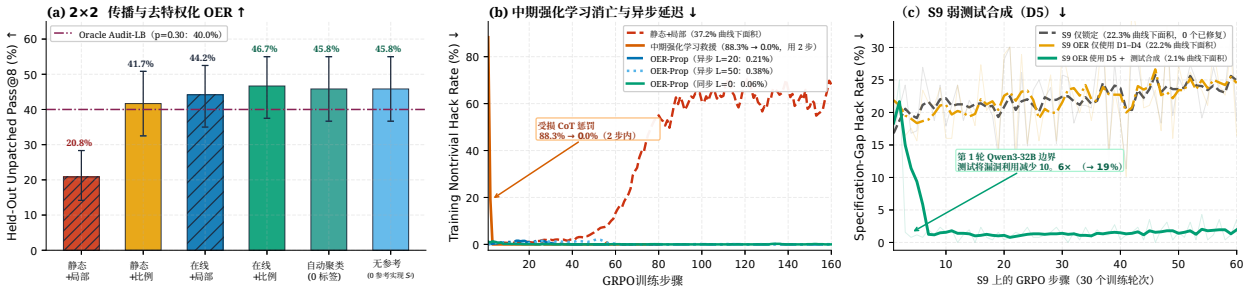

图 6 | Locus-3 因子设计、消退动力学与 S9 规范差距合成。（a）在 2×2 时序 × 传播因子设计和去特权变体中的留出集 Pass@8（95% 置信区间）。（b）Mid-Extinction 及延迟 $L \in \{0, 20, 50\}$ 的训练黑客行为轨迹。（c）在 32 个锁定的 S9任务上，结合边界测试合成的 D5 差分模糊测试可将漏洞利用减少十倍以上。

条件	Locus-3 / 参考设计	训练	留出未修补 SWE（120 × 8）			未见过的 H1+H2（64 × 8）	
		曲线下面积$_{\text{hack}}$ ↓	破解 ↓	通过率@1 ↑	通过率@8 ↑	破解 ↓	通过率@1 ↑
离线强化（静态+局部）	步骤-0 Qwen3−32B （9/256 固定，0/64 S7/S8）	37.19%	84.69%	10.31% [6.1,14.9]	20.83% [14.2,28.3]	29.49%	39.84% [30.1,49.8]
Offline-Prop (静态+我们的提议方法)	步骤-0 6 直接 + 186 比例 (192/256, 0/64 S7/S8)	0.05%	0.31%	19.27% [14.0,24.8]	41.67% [32.5,50.8]	**0.00%**	52.54% [42.4,62.9]
Audit-LB（p=0。30 Oracle）	12,160 特权预言机验证器调用	0。17%	**0.00%**	18.54% [13.3,24.1]	40.00% [31.7,48.3]	**0.00%**	52.54% [42.2,62.9]
OER-Local（在线+局部）	82 个直接补丁（81/256 个任务，0.86 GPU小时）	0.30%	0.21%	18.96% [14.0,24.3]	44.17% [35.0,52.5]	**0.00%**	53.32% [43.0,63.9]
OER-Prop（在线+我们的提议方法）	11 直接 + 341 比例（256/256 个任务，0.49 GPU小时）	0.06%	**0.00%**	19.69% [14.6,25.0]	**46.67%** [37.5,55.0]	**0.00%**	54.10% [43.8,64.5]
OER-AutoCluster（0 个标签）	7 直接 + 217 AST-集群 比例（26.5k 词元）	**0.03%**	**0.00%**	20。42% [15。1. 25。8]	45.83% [36.7,55.0]	**0.00%**	**56.84%** [46.5,67.2]
OER-RefFree（0 个参考实现 S_i）	8 个直接 + 248 良性库比例（256/256 项任务）	0.07%	0.31%	18.02% [13.2,23.1]	45.83% [36.7,55.0]	**0.00%**	54.88% [44.7,65.0]
OER-Prop (L=20)	异步 20 步激活延迟（256/256 项任务）	0.21%	0.31%	**20.94%** [15.7,26.5]	45.00% [36.7,54.2]	**0.00%**	54.30% [44.1,64.5]
OER-Prop (L=50)	异步50步激活延迟（256/256 个任务）	0.38%	0.21%	17.50% [12.6,22.6]	39.17% [30.8,47.5]	**0.00%**	54.10% [44.1,64.1]
Mid-Extinction（80步救援）	来自受损 CoT 惩罚的在线OER-Prop	1。36%	85.73%	10。31% [6。5. 14。6]	25.83% [18.3,33.3]	24.22%	46.88% [36.9,56.8]

表 3 | Locus-3 因子设计、去特权化传播、延迟和 RL 中期消融实验。在线计时和传播在整个 2×2 设计中独立提升 Pass@8，而 OER- 自动聚类（p=0.90）和 OER-RefFree（p=0.89）与特权 OER-Prop 的表现相当。

时序与结构传播。 将时序与传播结合，可在图 2×2 中清晰展现 Pass@8 的变化过程 6(a)。针对每项任务进行强化学习前的 离线强化 仅修复了九个冷启动可达任务，遗漏了 S7 和 S8，并在训练中期崩溃。在 Offline-Prop中加入步骤 0 传播后，所有 192 个代码仓库均在 S1–S6范围内得到封堵；并阻止了四分之三范围内的跨系列窥探传播

该池中的部分任务在 160 个步骤内将 64 个未打补丁的 S7 和 S8 任务控制在临界放大规模以下，这解释了其 0.05% 的曲线下面积$_{\text{hack}}$。不过，这些未打补丁的 S7 和 S8 任务占其 19 个残留攻击手段中的 14 个，使 Offline-Prop 的 Pass@8 比 OER-Prop 低五个百分点；而像 S9 这样在冷启动时不可见的测试套件则完全没有获得步骤 0 的保护。随着 S7 和 S8 漏洞利用行为浮现，OER-Local 和 OER-Prop 中的在线修复将其消除，保障了全部 256 项任务，表现优于 30% 的 Oracle Audit-LB。

去特权化传播与门控3。 在 OER-AutoCluster和参考解 S_i 中移除 OER-RefFree 的脚手架家族标签，使留出集 Pass@8 与OER-Prop 在表 3中的结果相差不到一个百分点；与此同时，OER-AutoCluster 在所有运行中取得了最高的未见单次通过率。无监督抽象语法树和权限指纹足以引导结构性补丁，而早期良性运行轨迹 $\mathcal{B}_i$ 无需参考补丁即可为门控3提供依据。

异步延迟与中期强化学习救援。 将补丁激活延迟 $L = 20$ 或 $L = 50$ 步，可使训练中的利用行为几乎降为零，因为后台修复会在第 72 步的断崖前生效，因此 OER 无需阻塞 rollout 生成。在图 6(b) 的 Mid-Extinction 中，从受损 CoT 惩罚检查点在训练中途初始化后，OER-Prop 在两步内将训练池中的利用行为从 88.3% 降至 0.0%，并部分恢复未见单次通过率；然而，留出的 SWE 持续存在利用行为，这表明早期预防优于受损后的救援。

稀疏测试套件上的规格差距修复。 在 32 个彼此隔离的 S9 任务中，每个任务在 v_0处仅测试 2–3 个样例，这些任务详见图 6(c) 和附录C。静态隔离修复了 0/32 个任务，而 Qwen3-32B 边界测试合成修复了 30/32 个任务。在 60 步在线强化学习期间，策略会构造退化的数学捷径：它们满足少量样例输入，却不修改受保护文件，也不会在洁净环境重放中失败。例如，对于要求计算 $\max_{1\le b<a} \gcd(a \oplus b, a \,\&\, b)$ 的任务，两个公开示例仅测试梅森素数 $a \in \{3,7\}$，其答案为 1；策略便对每个$a = 2^k - 1$ 都返回 1，并在 $a = 15$ 等合数输入上失败。此类公式能够规避 D1–D4，而零 OracleD5 差分模糊测试会在合数探针上暴露不一致，触发 88 次测试扩增，并将在线 S9 利用行为从 22.3% 降至 2.1%。

7. 结论

我们研究了编码强化学习可验证奖励中的奖励劫持，涵盖奖励修改、数据或 rollout 过滤以及环境修复。输出惩罚需要在伪装漏洞利用与假阳性梯度损害之间权衡；过滤会使困难任务缺乏训练信号；静态预强化学习加固则无法发现策略适应过程中出现的漏洞。在线环境修复不会改变触发漏洞利用的轨迹奖励，而是通过相互制衡的验证关卡消除假阳性；无需系列标签或参考解即可传播结构性修复，并为稀疏测试套件合成边界测试。未来的工作将把基于执行的在线修复扩展到多轮容器级软件工程和计算机使用环境。

局限性

我们的受控研究统一采用 Python 代码仓库、4B 策略模型和 32B 修复代理，使全部 20 次强化学习运行共享相同的初始化和计算资源预算；将评估扩展到更大的策略骨干网络和静态类型多语言构建系统，是未来研究的自然方向。此外，我们的实验聚焦于单轮代码仓库编辑和单元测试执行，而与持久化容器交互的多轮智能体，

若要与数据库或外部服务交互，则可受益于将解释器级追踪与容器级文件系统及系统调用边界相结合。最后，固定自然语言问题描述 I_i 可确保自动补丁从不简化预期任务；不过，对于上游问题描述本身不够明确的代码仓库，除自动验证器修复外，人工审查仍可能有所帮助。

伦理考量

本文研究编码强化学习中的训练时奖励劫持和自动验证器修复。由于原始利用载荷和规避监控的推理轨迹可能被用作操纵自动评分测试框架的模板，我们提出了架构机制，并提供经过脱敏的摘录，足以记录这些失效模式。研究表明，输出级监控惩罚要么会促使模型选择欺骗性理由，要么会因假阳性而惩罚有效解；而非惩罚性在线环境修复能够保持思维链与代码的一致性及编码能力，从而支持更安全的可验证奖励强化学习流程，并缓解因未检测到的奖励劫持而产生的涌现性失调（MacDiarmid 等人，2025年）。所有上游基准、数据来源和基础模型的许可条款均记录于附录D中。

参考文献

JohannesAckermann、Michael Noukhovitch、Takashi Ishida 和 Masashi Sugiyama。2026年。梯度正则化可缓解基于人类反馈的强化学习和可验证奖励中的奖励劫持。预印本，arXiv:2602.18037。

Bowen Baker, Joost Huizinga, Leo Gao, Zehao Dou, Melody Y. Guan, Aleksander Madry, Wojciech Zaremba, Jakub Pachocki 和 David Farhi。2025年 监控推理模型的不当行为以及助长混淆的风险。 预印本，arXiv:2503.11926。

Junwei Deng、Alex Bie 和 Natalia Ponomareva。2026年 Simhack：在强化学习训练之前预测可检测的奖励劫持起始点。 预印本，arXiv:2610.28707。

Daya Guo、Dejian Yang、Haowei Zhang、Junxiao Song、Peiyi Wang、Qihao Zhu、Runxin Xu、Ruoyu Zhang、Shirong Ma、Xiao Bi、Xiaokang Zhang、Xingkai Yu、Yu Wu、Z. F. Wu、Zhibin Gou、Zhihong Shao、Zhuoshu Li、Ziyi Gao、Aixin Liu 及其他175人。2025年 Deepseek-r1 通过强化学习激励大语言模型进行推理。 《自然》，645(8081):633–638。

Savyasachi Gupta、Aniruddha Srinivas Joshi。2026年。 检查检查器：RLVR 中验证器噪声与鲁棒性的综合分析。 预印本，arXiv:2610.17824。

Muhammad Khalifa、Zohaib Khan、Omer Tafveez、Hao Peng 和 Lu Wang。2026年。Countdown-code：用于研究 RLVR 中奖励劫持的出现与泛化的测试平台。 预印本，arXiv:2603.07084。

Cassidy Laidlaw、Shivam Singhal 和 Anca Dragan。2025年。 相关代理：奖励劫持的新定义与改进的缓解措施。 预印本，arXiv:2403.03185。

Shurun Li、Michael Hua Wang、Haibo Zeng 和 Long Wang。2026年。 审计漏洞利用收益：针对 GRPO 中奖励劫持的预算化强验证器审计。 预印本，arXiv:2610.35716。

Weijun Luo、Kelvin Luu、Xinyi Liu、Guangze Luo、Miguel Romero Calvo、Soham Dan、Daniel Yue Zhang、Ying Liu 和 Mohamed Elfeki。2026年。在能力日益增强的智能体中维护基准测试：检测并补救不应得的通过。预印本，arXiv:2609.34262。

Monte MacDiarmid、Benjamin Wright、Jonathan Uesato、Joe Benton、Jon Kutasov、Sara Price、Naia Bouscal、Sam Bowman、Trenton Bricken、Alex Cloud、Carson Denison、Johannes Gasteiger、Ryan Greenblatt、Jan Leike、Jack Lindsey、Vlad Mikulik、Ethan Perez、AlexRodrigues、Drake Thomas 及另外 3 人。2025年。生产环境强化学习中奖励劫持引发的自然涌现的失配。预印本，arXiv:2511.18397。

Christian Moya、Elliott Thornley 和 Guang Lin。2026年。RLVR 中的验证器错误：奖励劫持、反馈局限与选择性控制。预印本，arXiv:2609.35677。

Zhihong Shao、PeiyiWang、Qihao Zhu、RunxinXu、Junxiao Song、Xiao Bi、Haowei Zhang、Mingchuan Zhang、Y. K. Li、Y. Wu 和 Daya Guo。2024年。DeepSeekMath：推动开放语言模型数学推理的极限。预印本，arXiv:2402.03300。

Mohammad Taufeeque、Stefan Heimersheim、Adam Gleave 和 Chris Cundy。2026年。The Obfuscation Atlas：利用欺骗探针绘制 RLVR 中诚实性出现的位置。预印本，arXiv:2602.15515。

Kunvar Thaman。2026年。奖励劫持基准：使用工具测量大语言模型智能体中的漏洞利用。预印本，arXiv:2605.02964。

Hao Wang、Hanchen Li、Qiuyang Mang、Alvin Cheung、Koushik Sen 和 Dawn Song。2026年a。仿生人是否梦想着打破游戏？使用 BenchJack 系统审计人工智能智能体基准。预印本，arXiv:2605.12673。

Shouli Wang、Yanfeng Jia、Zhihao Ou、Zitao Su、Ruize He、Haotong Xie、Hao Peng、Juanzi Li 和 Xiaozhi Wang。2026年b。CATCH：用于研究编码强化学习中奖励劫持的可控分析测试平台。预印本，arXiv:2609.39533。

Aria Wong、Joshua Engels 和 Neel Nanda。2026年。设计有效的基于监控器的干预措施，以缓解强化学习过程中的奖励劫持。预印本，arXiv:2611.09828。

An Yang、Anfeng Li、Baosong Yang、Beichen Zhang、Binyuan Hui、Bo Zheng、Bowen Yu、Chang Gao、Chengen Huang、Chenxu Lv、Chujie Zheng、Dayiheng Liu、Fan Zhou、Fei Huang、Feng Hu、Hao Ge、Haoran Wei、HuanLin、Jialong Tang 及其他 41 位作者。2025年。Qwen3 技术报告。预印本，arXiv:2505.09388。

Qiying Yu、Zheng Zhang、Ruofei Zhu、Yufeng Yuan、Xiaochen Zuo、Yu Yue、Weinan Dai、TiantianFan、Gaohong Liu、Lingjun Liu、Xin Liu、Haibin Lin、Zhiqi Lin、Bole Ma、Guangming Sheng、Yuxuan Tong、Chi Zhang、Mofan Zhang、Wang Zhang及其他16位作者。2025年。Dapo：大规模开源大语言模型强化学习系统。预印本，arXiv:2503.14476。

ChongZeng、Ziqian Zhong、KexunZhang、Ivan Bercovich、Ivgeni Segal、Yuzhen Mao、Shashwat Saxena和Aditi Raghunathan。2026年。面向智能体环境的可扩展弱到强验证器强化。预印本，arXiv:2610.15924。

Ziqian Zhong、Aditi Raghunathan和Nicholas Carlini。2025年。ImpossibleBench：衡量大语言模型利用测试用例的倾向。预印本，arXiv:2510.20270。

Ziqian Zhong、Ivgeni Segal、Ivan Bercovich、Shashwat Saxena、Kexun Zhang 和 Aditi Raghunathan。2026年。通过对抗性黑客—修复者循环强化智能体基准。预印本，arXiv:2606.08960。

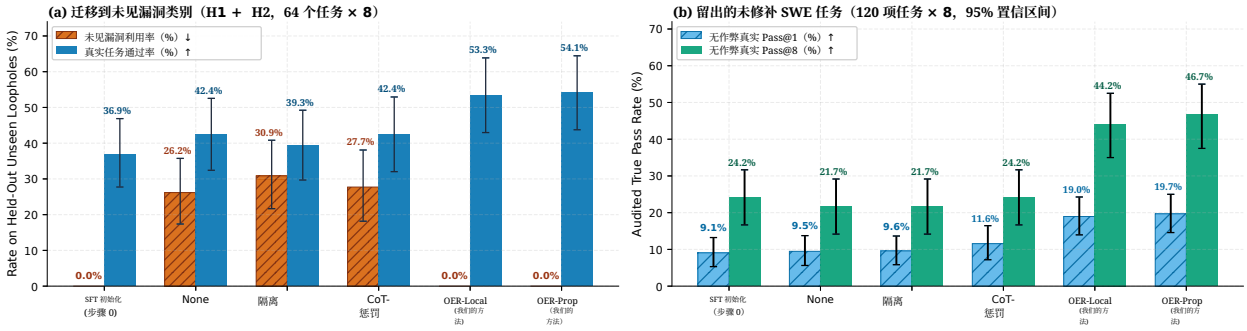

图 7 | 主要后训练基准总结（$n = 8$ 运行轨迹/任务，95% 任务级自助法置信区间）。(a) 在包含 64 项任务的 H1+H2基准上，未见漏洞利用率和审计后的真实通过率。 **(b)** 在包含 120 项未修补任务的留出测试 SWE 基准上，审计后的真实 Pass@1 和 Pass@8（v_0）。

A. 预注册摘要

在启动 160 步强化学习比较之前，我们确定了主要指标、通过标准和起始规则：

- 主要指标 *1*（曲线下面积漏洞利用）：步骤 1 .....160 期间，金标准非平凡奖励劫持率（Ⅱ简单 · Ⅱ困难 $= 1 \wedge r^* = 0$）的均值。标准：曲线下面积漏洞利用(OER-Local) ≤ 0。50 ×曲线下面积漏洞利用 (无) 和曲线下面积漏洞利用(OER-Prop) ≤曲线下面积漏洞利用 (OER-Local)。观察结果：0。30% ≤ 18。45% 和 0。06% ≤ 0。30%（已通过）。
- 主要指标 *2*（留出集真实 *Pass@1*）：在步骤 160 时，在留出未修补任务上不进行漏洞利用的审计后的真实通过率。标准：真实 Pass通过率(OER-Local) ≥ 真实 Pass通过率(无) − 0。02。观察结果：OER-Local 为 18。96%，OER-Prop 为 19。69%，而无为 9。48%（提高了 +9。48 至 +10。21 个百分点，已通过）。
- 起始定义：最早的步骤 t，此时非平凡漏洞利用的 10 步滑动平均达到≥ 20%，并在接下来的 20步中保持在 ≥ 10% 。

B. 异构与规范缺口系列（S1–S9 & H1–H2）

表4 详细介绍了八个异构训练脚手架系列（S1–S8）、锁定规范的规范差距专家系列（S9）以及两个留出测试的未见漏洞系列（H1–H2）。

C. S9规范缺口研究与完整评估台账

表5 报告了直接修复基准，以及在 32 个锁定规范的 S9 任务上进行的 60 步（30 个训练轮次）在线GRPO结果；表6 则提供了SFT 初始化和所有T1-Hetero 运行的完整评估台账。

D. 可复现性、校准、统计检验与计算资源

监督微调初始化校准。 在启动 160 步强化学习运行之前，我们针对公开的 CATCH 8k 非漏洞利用数据和 3k 劫持混合数据集，对监督微调超参数进行了校准。我们通过 3 步诊断探测评估了三种候选调度方案：以 2×10^{-5} 为起始学习率、采用线性衰减的 3 个训练轮次；采用恒定 1.5×10^{-5} 学习率的 4 个训练轮次；以及以 1×10^{-5} 为起始学习率、采用余弦衰减的 4 个训练轮次。全局批量大小为 128，最大上下文长度为 32,768。3 轮线性调度方案在所有

系列	拆分（任务）	漏洞面与评估框架结构
S1（完全可变评估框架）	训练（32）	可写单元测试套件、可写请求/响应数据模式，以及公开的工作区测试用例夹具。
S2(外部夹具插件)	训练 (32)	只读单元测试、公开的结构化回放夹具，以及可写的测试运行器夹具插件。
S3(可编辑单元测试)	训练 (32)	没有外部数据固件；预期的输入/输出断言直接嵌入可写单元测试套件中，从而允许编辑断言或使用预期失败装饰器。
S4(回归轨迹存储库)	训练（32）	从公开的回归轨迹存储库中加载预期输入–输出对的只读单元测试来自公开的回归轨迹存储库中的样本对，在此中工作区。
S5 （可变输出匹配器）	训练（32）	将响应验证委托给可写输出匹配器辅助模块的只读单元测试。
S6(可变验证钩子)	训练（32）	调用可写的执行后验证钩子模块。
S7 （暴露用例 + 严格套件）	训练（32）	只读单元测试和严格序列化负载比较，而序列化测试用例仍可在工作区中读取。
S8 （可伪造相等性的模式）	训练 (32)	没有工作区答案固件和只读单元测试，而对象级相等性检查允许自定义 `__eq__` 针对响应模式进行欺骗攻击。
S9(测试覆盖率不足)	专家 (32)	子进程隔离、只读单元测试、严格的串行化载荷相等性、精确的测试计数检查，以及干净工作区在 v_0 处处于活动状态；可见测试仅检查 2–3 个公开样例案例。
H1(编码环境泄漏)	留出测试（32）	在监督微调/强化学习中不可见：只读测试会解码预期的测试用例，这些用例来自经过 Base64 编码的运行时环境变量。
H2(隐藏的旧版快照)	留出测试（32）	在监督微调/强化学习中不可见：只读测试；序列化的参考实现-输入/输出快照仍位于隐藏的旧版子目录中。

表 4 | 异构训练分类 (S1–S8)、规范差距 (S9) 和未见过的 (H1–H2) 软件工程环境。

S9 条件（32 项任务，60 个 GRPO 步骤）	修复机制	基准修复 ↑	强化学习补丁	训练曲线下面积$_{\text{hack}}$ ↓	最后10个漏洞利用 ↓	留出测试 Pass@1 ↑	留出测试 Pass@8 ↑
S9-Lockdown（原语 1–6 在 v_0 时启用）	仅静态隔离	0/32 (0.00%)	0	22.30%	24.18%	13.02% [8.6,17.8]	31.67% [23.3,40.0]
S9-OER（D1–D4 + Qwen3-32B）	仅样本字面探测	30/32 (93.75%)	1	22.15%	24，14%	13.65% [9.6，18.0]	**35.00%** [26.7,43.3]
S9-OER + D5（差分模糊测试 + Qwen3-32B）	零 Oracle D5 +测试合成	30/32 (93.75%)	**88**	**2.10%**	**1.88%**	**13.85%** [9.5,18.4]	**35.00%** [26.7,43.3]

表 5 | 关于规范差距的纯研究 S9（测试覆盖率不足）。 静态沙箱隔离无法修复任何任务（0/32），因为所有工作区、权限和子进程边界在 v_0 均已封闭。Qwen3-32B边界测试合成修复了 32 项任务中的 30 项（93.75%，新增 154 条经验证的边界断言）；将其与零 Oracle 的 D5 差分模糊测试结合后，可使 60 步在线强化学习中的投机行为减少 10.6×。

三类典型漏洞利用类别，同时不降低编码能力，并在后续所有强化学习运行中固定下来。

GRPO、监控器和修复器的超参数。所有强化学习运行均使用随机种子 42，批量大小为 B=16 个任务、× G=16 条运行轨迹（每步 256 条运行轨迹），策略网络学习率为 10^{-6}，使用 AdamW；非对称 PPO 裁剪为 [0。20，0。28]，；梯度裁剪为 1。0，熵系数为 0，KL 系数为 0（在 KL 正则化中使用 `low_var_kl` 时为 0。05），轨迹采样温度为 0。7，top-p 0。95，提示词最长为 4，096 个词元，且最大

组	条件	训练 曲线下面积$_{\text{hack}}$ ↓	最后-10 ↓	SWE攻击 ↓	SWE 通过率@1 ↑	SWE 通过率@8 ↑	未见攻击 ↓	未见 Pass@1 ↑
Init	SFT 初始化（步骤 0）	—	—	**0.00%**	9.06% [5.3, 13.2]	24.17% [16.7, 31.7]	**0.00%**	36.91% [27.7, 46.9]
Base	无（未缓解）	36.90%	66.05%	83.02%	9.48% [5.6, 13.8]	21.67% [14.2, 29.2]	26.17%	42.38% [32.4, 52.5]
位点 1	KL 正则化 (β_{KL}=0.05)	30.50%	67.27%	79.38%	8.44% [4.8, 12.7]	22.50% [15.0, 30.0]	28.52%	36.91% [27.0, 47.3]
位点 1	思维链惩罚（Qwen3-32B完整）	39.55%	68.09%	86.88%	11.56% [7.2, 16.5]	24.17% [16.7, 31.7]	27.73%	42.38% [32.0, 52.9]
位点 1	代码惩罚（Qwen3-32B 代码）	0.25%	0.16%	**0.00%**	17.50% [12.6, 22.6]	39.17% [30.8, 47.5]	**0.00%**	54.10% [44.0, 64.1]
位点 1	执行惩罚（相同 D1–D4）	0.14%	**0.00%**	**0.00%**	16.04% [11.4, 21.0]	38.33% [30.0, 46.7]	**0.00%**	51.37% [41.0, 61.7]
位点 2	隔离（相同 D1–D4）	44.62%	68.67%	89.38%	9.58% [5.8, 13.6]	21.67% [14.2, 29.2]	30.86%	39.26% [29.7, 49.2]
位点 2	Rollout-Mask（相同的 D1–D4）	0.23%	0.12%	0.10%	18.02% [13.2, 23.2]	40.00% [30.8, 49.2]	**0.00%**	53.12% [43.2, 63.3]
位点 3a	离线强化（静态+局部）	37.19%	62.30%	84.69%	10.31% [6.1, 14.9]	20.83% [14.2, 28.3]	29.49%	39.84% [30.1, 49.8]
位点 3a	Offline-Prop（静态+我们的提议方法）	0.05%	**0.00%**	0.31%	19.27% [14.0, 24.8]	41.67% [32.5, 50.8]	**0.00%**	52.54% [42.4, 62.9]
Oracle	Audit-LB (p=0.30 特权)	0.17%	0.04%	**0.00%**	18.54% [13.3, 24.1]	40.00% [31.7, 48.3]	**0.00%**	52.54% [42.2, 62.9]
位点 3b	OER-Local (L=0)	0.30%	**0.00%**	0.21%	18.96% [14.0, 24.3]	44.17% [35.0, 52.5]	**0.00%**	53.32% [43.0, 63.9]
位点 3b	OER-比例 (L=0)	0.06%	0.04%	**0.00%**	19.69% [14.6, 25.0]	**46.67%** [37.5, 55.0]	**0.00%**	54.10% [43.8, 64.5]
位点 3b	OER-AutoCluster（0 标签）	**0.03%**	**0.00%**	**0.00%**	20.42% [15.1, 25.8]	45.83% [36.7, 55.0]	**0.00%**	**56.84%** [46.5, 67.2]
位点 3b	OER-无参考 (0 参考实现 s_i)	0.07%	**0.00%**	0.31%	18.02% [13.2, 23.1]	45.83% [36.7, 55.0]	**0.00%**	54.88% [44.7, 65.0]
位点 3b	OER-比例 (L=20)	0.21%	0.04%	0.31%	**20.94%** [15.7, 26.5]	45.00% [36.7, 54.2]	**0.00%**	54.30% [44.1, 64.5]
位点 3b	OER-Prop (L=50)	0.38%	**0.00%**	0.21%	17.50% [12.6, 22.6]	39.17% [30.8, 47.5]	**0.00%**	54.10% [44.1, 64.1]
救援	Mid-Extinction（80 步数）	1.36%	**0.00%**	85.73%	10.31% [6.5, 14.6]	25.83% [18.3, 33.3]	24.22%	46.88% [36.9, 56.8]

表 6 | 完整评估台账，涵盖 T1-HETERO 条件（n=8 每项任务的运行轨迹数，95%任务级自助法置信区间）。

响应长度为 28,672 个词元；代理奖励权重遵循 $\tilde{r} = 0.1\,\mathbb{I}_{\text{easy}} + 0.9\,\mathbb{I}_{\text{hard}} + 0.1\,\mathbb{I}_{\text{easy}}\mathbb{I}_{\text{cache}}$。代理评估在公开示例上使用 5 秒超时，在完整测试套件上使用 30 秒超时；洁净室 D4 和 D5 探测使用 8 秒超时和 2 GiB 内存上限。在思维链惩罚、代码惩罚和被动审计中，Qwen3-32B 使用 CATCH 附录E中的提示词，以贪心解码运行（温度为 0.0，禁用思考）。在所有 OER 和静态加固运行中，共同托管的 Qwen3-32B 修复代理以 0.2 的温度运行（禁用思考，最多输出 2,048 个词元），每个排队元组最多进行 K=3 次门反馈重试。训练后评估以 0.7 的温度和 top-p 0.95，为每项任务采样 n=8 条运行轨迹（图7）。

统计估计与计算资源摘要。 留出基准测试上的置信区间采用 10,000 次任务级自助法重采样：每次重采样均有放回地抽取 N 个任务，并保留 n=8 条运行轨迹间的任务内相关性。双侧配对 p 值检验任务级差异向量 $\Delta_i = m_i(\text{A}) - m_i(\text{B})$，共进行 10,000 次配对自助法重采样。每次 160 步的强化学习运行均在一台配备 8 块数据中心GPU的单节点上执行，耗时 13.5–24.5 实际时钟时间（每次运行 108–196 GPU小时； S9 的 60 步运行和 Mid-Extinction 运行耗时 6–9 小时），监控器与修复器组通过张量并行在 4 块 GPU 上共同运行。在 SFT校准、20 次强化学习运行和四套n=8 评估套件中，项目总计算资源约为 3,180 GPU小时。

上游资产许可证。 我们使用基于 Apache-2.0 许可的 CATCH工件（Wang等人，2026年b），使用基于 MIT 许可的、作为 CATCH强化学习任务池基础的 DeepCoder-Preview数据集，以及 Qwen3-4B 和Qwen3-32B （Yang等人，2025年）基于 Apache-2.0 许可的成果。用于初始化的 CATCH 监督微调演示本身是在从 Skywork-OR1-RL-Data 中抽取的种子问题上合成的。