Coding Agent 训练数据质量：为什么测试通过还不够？

239.AI · 239.AI · 2026年11月12日

测试通过不等于代码正确。本文从一个简短的代码例子出发，分析 Coding Agent 任务中的需求遗漏、Verifier 误判、奖励偏差与训练数据质量。

在 Coding Agent 的评测中，测试通过通常意味着任务被判定为完成。在部分采用可执行反馈的强化学习（RL）设置中，这个判定还会进一步变成奖励。相比人工逐份审查补丁，自动化测试有明确的优势：运行成本可控，结果可重复，也能检查真实的软件行为。

问题是，**测试接受了一份补丁，不等于这份补丁满足了任务的全部要求**。如果两者之间的偏差没有被识别，评测会高估模型能力，训练则可能奖励并未真正完成任务的代码。

这不是“测试有没有写够”的单一问题。它还涉及需求是否明确、测试是否错把实现细节当成要求、执行失败有没有被误当成代码失败，以及模型在反复获得反馈后会生成怎样的候选补丁。

一个很小的错误：先截断，还是先筛选？

假设一个任务查询接口需要支持按来源筛选，并最多返回 `limit` 条结果。需求约定是**先按来源筛选，再截断结果**。下面用 `239.AI` 作为虚构测试数据中的来源标签；这不是 239 的实际代码。

```
def select(rows, source, limit=1):
    return [r for r in rows[:limit] if r["source"] == source]

assert select([{"source": "239.AI"}], "239.AI")  # 通过
assert select([{"source": "other"}, {"source": "239.AI"}], "239.AI")  # 失败
```

第一条断言通过，第二条失败。函数没有先筛选，而是先从原始记录中取一条，再对这一条过滤。只要匹配项不在原始记录的前 `limit` 条里，就会被漏掉。

如果验证器只包含第一条测试，这份错误实现会得到 PASS。测试没有撒谎：它所检查的输入确实返回了匹配结果。遗漏的是**筛选与截断的组合语义**。

真实仓库里，这类遗漏往往更难发现。新增一个 API 参数可能改变默认行为；修改缓存逻辑可能破坏并发条件下的一致性；修复异常处理可能影响调用方依赖的错误类型。局部功能正常，并不代表修改后的系统符合完整要求。

测试通过，到底证明了什么？

一份软件工程任务至少有三件不同的东西：任务要求、候选实现、验证测试。测试执行的是有限输入上的断言；正确性要讨论的则是实现是否满足适用的行为契约。

自动程序修复研究中常用 *plausible patch* 指通过给定测试的补丁，用 *correct patch* 指真正符合预期行为的补丁。前者是测试观察结果，后者是更强的语义判断。两者不能直接画等号。

代码覆盖率也不能填平这道差距。即使相关分支全部执行过，断言仍可能没有检查操作顺序、状态保持或跨模块交互。测试名称叫“验证分页”，也不代表它真的检查了分页的边界。**执行覆盖、断言有效性和需求覆盖，是三回事。**

进一步说，测试不只是可能“漏测”，还可能检查了不该检查的东西。

以刚才的例子为例，如果任务只写“支持按来源筛选并限制结果数量”，原有接口和文档也没有交代组合顺序，那么“先筛选再截断”是否是必须遵守的行为，就缺少明确依据。此时，隐藏测试坚持某种顺序，不能自动证明另一种实现有错。

但这不意味着题面必须列出每个边界条件。阅读仓库、理解已有接口和兼容约定，本来就是 Coding Agent 的工程工作。问题的边界在于：**用于拒绝答案的要求，能否从任务描述和解题者实际可访问的上下文中合理确定。** 只有验证器知道的私有预期，不应悄悄成为题目要求。

Verifier 的错误不止一种

讨论 Verifier（验证器）可靠性时，容易只关注错误实现被放过。实际上还要关注另一面：正确实现被拒绝。

误判	实际情况	验证器的结果	影响
错误接受（False Acceptance）	实现违反任务要求	PASS	错误代码获得正向信号

误判	实际情况	验证器的结果	影响
错误拒绝（False Rejection）	实现满足任务要求	FAIL	有效解法失去正向信号

在仓库级任务中，错误拒绝的一种常见诱因是：同一功能可以有多种正确实现，但测试却绑定了特定私有函数、内部调用顺序或参考补丁的代码结构。

例如，同样是“筛选后限制结果数”，可以在内存中操作，也可以交由数据库查询完成。只要任务没有限制实现机制，两者都应按对外行为判断。若隐藏测试要求必须调用某个私有 helper，测试实际上就把参考实现的选择当成了正确性的定义。

历史 PR 的补丁是重要的参考实现，却不是完整的规格说明。它可以证明某种方案曾被采用，也可以用于寻找值得调查的行为差异；**不同于参考补丁，不等于错误**。反过来，参考补丁通过测试，也不能证明测试能够识别所有错误实现。

因此，一套验证器是否可靠，需要同时考察它能否拒绝有明确需求违例的候选，以及能否接受不同但正确的实现。前提是“错误”与“正确”的判断有独立于当前测试结果的依据，而不是先看 PASS/FAIL，再倒推代码语义。

为什么“能拒绝 95% 的错误代码”仍不足以说明 PASS 可信？

这里有一个经常被忽略的统计问题：**错误拒绝率高，不等于被接受的补丁大多正确**。最终 PASS 的可信度还取决于候选补丁中正确实现的比例。

假设某批候选有 1,000 份，其中真正正确的只有 100 份，另外 900 份存在功能错误。验证器接受了 90 份正确补丁，也错误接受了 45 份错误补丁。它拒绝了 855/900，即 95% 的错误补丁；但 135 份 PASS 中，只有 90 份真正正确。

换句话说，在这个**纯假设**的分布里，PASS 的正确比例仅约 67%。这不是行业缺陷率，也不能推广到任何特定模型或数据集。它说明：判断验证质量时，不能只给出错误补丁的检出率，还要关心被接受样本的正确性，以及计算这些比例时采用了什么候选分布。

而候选分布恰恰不是恒定的。不同模型、难度和采样策略会产生不同的补丁；同一模型在训练前后，也不一定犯同一种错误。

当 PASS 变成 RL 奖励，验证盲区会怎样改变问题？

在采用结果奖励（outcome reward）的某些 SWE RL 设置中，系统可能将验证器的 PASS 映射为正奖励。训练希望模型学会完成软件工程任务，但算法直接优化的是实际提供的奖励信号。如果验证器接受了违反需求的补丁，两者就发生偏离。

最初这看起来像普通标签噪声，但 Coding Agent 有一个关键特点：它会阅读代码、执行命令、观察反馈、修改方案。**模型生成什么补丁，会随着能力和探索策略改变；验证器面对的错误分布，也可能随之改变。**

能力较弱的 Agent 可能经常提交无法运行的代码；更强的 Agent 可能完成主要功能，却在兼容性、边界条件或跨模块交互处留下细微缺陷。后者更容易通过只覆盖常规路径的测试。由简单候选得出的“验证器足够可靠”，不能直接外推到后续更强的候选。

如果优化持续偏向某些可获得奖励、但没有完成任务的实现，验证盲区就可能从偶发误判变成可被策略反复触及的路径。不过，这是一种有条件的风险，不意味着单次错误奖励必然导致训练退化，也不意味着所有错误补丁都属于 Reward Hacking。

这里最好区分三件事：Agent 无意写错、实现针对已知测试样例过拟合，以及候选代码主动干扰测试执行或结果生成。它们都可能导致不可信奖励，但成因不同：补更多业务断言，不能自动防止结果被篡改；加强执行隔离，也不能自动补齐业务语义。

比“让参考答案通过”更有价值的验证证据

正确实现通过、未修改的起点无法满足新要求，是有用的基本证据，却不能证明测试能够识别部分正确的补丁。前面的“先截断再筛选”就是一个具体反例：它能运行，也能满足简单样例，只在组合条件下违反任务要求。

因此，比单独看参考答案 PASS 更有信息量的是：**一个已确认违反要求、但其他部分尽可能正常的实现，会不会被同一套测试拒绝？** 失败还应当归因于目标行为，而不是语法错误、环境崩溃或测试根本没运行。

另一侧也要检查：一个采用不同机制、但已确认满足要求的实现，是否被正常接受？这能暴露验证器对参考实现的过度绑定。

这些证据依然是有限的。拒绝一个错误补丁，不代表所有错误补丁都会失败；接受一个替代正确实现，也不能证明不存在误拒。可靠的技术结论应当说明自己覆盖了哪些要求和错误类型，而不是将有限验证说成绝对正确。

FAIL 和 ERROR 不能混在一起

一份代码失败，可能是功能不符合要求，也可能是依赖没有装好、测试未被收集、进程超时、报告缺失，或者不同运行之间出现不稳定结果。

只有在验证正常完成、断言确实检查了预定行为时，FAIL 才能作为相应检查的有效负面结果。如果测试根本没执行，就无法由退出码推断代码语义。在需要形成可靠训练反馈的场景中，至少应区分：

- **PASS：** 完整执行的既定检查接受了候选；
- **FAIL：** 完整执行的既定检查发现候选未满足某些断言；
- **ERROR：** 缺少可信、完整的执行结果，不能作出正常的功能判断。

PASS 和 FAIL 仍然相对于那套测试成立，不是无限范围的正确性真值。把 ERROR 直接压成 Reward 0，会让环境故障和能力失败混为一谈；仅凭一个报告文件宣称 PASS，而不确认本次实际执行，也可能错误地相信被污染的结果。

可靠的验证，还不是训练数据质量的全部

即使一项任务需求明确、验证合理、执行可信，也不一定适合某个模型的训练目标。

大量只需修改一行条件的任务，可能有非常准确的奖励，却未必能训练复杂仓库理解和跨模块修改能力。反过来，目标模型几乎无法探索出有效解法的任务，即便失败判断准确，也未必提供高效的学习信号。难度是相对于模型、工具和执行预算而言的，不能只用改动行数定义。

数据分布也不可忽视。高度重复的任务可能覆盖很窄的能力空间；公开仓库中的任务还可能与模型已有训练材料或评测题目重叠，影响对能力增益的判断。

如果样本包含完整 Agent 操作轨迹，还应区分最终补丁的正确性与轨迹的教学价值：一个最终成功的过程可能包含大量无效搜索和偶然试错，不能因此认为每一步都值得模仿。

因此，判断 Coding Agent 数据质量至少要分清三个层次：**任务是否成立、反馈是否可信、样本是否适合目标训练**。前两者可以通过需求、候选行为和执行证据逐步建立；最后一项还需要结合模型表现、数据分布和训练后的独立评测。

自动化测试仍然不可替代。需要避免的是，把一次 PASS 从有限的运行观察，升级成对代码正确性、奖励可靠性乃至训练效果的全面保证。对训练数据而言，真正重要的不是标签有多确定，而是**这份确定性来自哪些可检查的证据，又在哪些地方停止成立**。

原文出处: https://239.ai/zh/posts/coding-agent-training-data-quality/

239.AI

欢迎转载本站文章，无需另行申请；请注明作者，并保留原文链接。