PR #6313 完整报告

verl-project/verl

[tool] fix: tool response truncate side

合并时间：2026-05-14 21:45

原文链接：http://prhub.com.cn/verl-project/verl/pull/6313

执行摘要

- 一句话：修复 tool 响应截断方向错误
- 推荐动作：建议精读此 PR，特别是测试用例的设计方式，其清晰展示了如何通过 Mock 模拟复杂的 agent loop 行为并进行断言。核心逻辑变更虽小，但体现了对配置语义一致性的细致追求，值得借鉴。后续可考虑补充 max_tool_response_length=0 边界情况的测试和修复。

功能与动机

PR body 指出原有截断逻辑将 'left' 与 'right' 的实现恰好颠倒：'left' 保留了头部而丢弃了尾部，'right' 保留了尾部。这会导致多轮工具调用中，模型收到不完整的工具响应，丢失关键的尾部结论（如 Final answer: Paris），从而造成静默的 prompt 损坏和训练信号错误。

实现拆解

1. 截断逻辑修正：在 verl/experimental/agent_loop/tool_agent_loop.py 的 _call_tool 方法中，交换了 'left' 和 'right' 分支的字符串切片操作。原 'left' 分支使用 tool_response_text[:self.max_tool_response_length] + '...(truncated)'，现改为 '(truncated)...' + tool_response_text[-self.max_tool_response_length:]；原 'right' 分支对应互换。
2. 测试辅助增强：在 tests/experimental/agent_loop/test_call_tool_on_cpu.py 中新增 FakeLongResponseTool 类，用于模拟返回长文本的工具；_make_tool_agent_loop 函数增加 max_tool_response_length 和 tool_response_truncate_side 参数，使测试能灵活控制截断配置。
3. 新增单元测试：添加 test_left_truncation_keeps_response_tail 和 test_right_truncation_keeps_response_head 两个测试用例，分别验证 'left' 截断后响应以 '(truncated)...' 开头并以尾部关键文本结尾，而 'right' 截断后响应以头部关键文本开头。
4. 运行验证：通过 pytest 执行全部 8 个测试用例通过，并通过 ruff check 确保代码风格合规。

关键文件：

- verl/experimental/agent_loop/tool_agent_loop.py（模块 Agent 循环；类别 source；类型 core-logic）：核心修复文件，交换了截断分支的切片逻辑，使语义与 veRL 其余部分一致。

- tests/experimental/agent_loop/test_call_tool_on_cpu.py（模块 Agent 测试；类别 test；类型 test-coverage；符号 _make_tool_agent_loop, FakeLongResponseTool, init, execute）：新增的测试扩充分了覆盖范围，包括长响应截断场景和参数化配置，确保修复正确且不破坏原有功能。

关键符号：ToolAgentLoop._call_tool, FakeLongResponseTool.execute, _make_tool_agent_loop, test_left_truncation_keeps_response_tail, test_right_truncation_keeps_response_head

关键源码片段

verl/experimental/agent_loop/tool_agent_loop.py

核心修复文件，交换了截断分支的切片逻辑，使语义与 veRL 其余部分一致。

```
# 截断逻辑片段（修复后）
tool_response_text = tool_execution_response.text
if tool_response_text and len(tool_response_text) > self.max_tool_response_length:
    if self.tool_response_truncate_side == "left":
        # 左侧截断：保留尾部内容，丢弃头部
        tool_response_text = "(truncated)..." + tool_response_text[-self.max_tool_response_length :
        ]
    elif self.tool_response_truncate_side == "right":
        # 右侧截断：保留头部内容，丢弃尾部
        tool_response_text = tool_response_text[: self.max_tool_response_length] + "...(truncated)
        "
    else:
        # 中心截断：保留头部和尾部，丢弃中间
        length = self.max_tool_response_length // 2
        tool_response_text = tool_response_text[:length] + "...(truncated)..." + tool_response_text[-
        length:]
```

注意：当 self.max_tool_response_length 为 0 时，text[-0:] 会返回整个字符串而非空字符串（Python 特性），这是一个已知的未修复边界问题。

tests/experimental/agent_loop/test_call_tool_on_cpu.py

新增的测试扩充分了覆盖范围，包括长响应截断场景和参数化配置，确保修复正确且不破坏原有功能。

```
class FakeLongResponseTool(FakeTool):
    """返回长文本的工具，用于测试截断逻辑。"""

    def __init__(self, name: str, text: str):
        super().__init__(name)
        self.text = text # 工具返回的预定义长文本

    async def execute(self, instance_id, parameters, **kwargs):
        return ToolResponse(text=self.text), 1.0, {}

def _make_tool_agent_loop(
```

```
    tools: dict[str, Any],
    max_tool_response_length: int = 10000,
    tool_response_truncate_side: str = "left",
):
    """创建一个最小化的 ToolAgentLoop Mock，支持自定义截断配置。"""
    from verl.experimental.agent_loop.tool_agent_loop import ToolAgentLoop

    mock = MagicMock(spec=ToolAgentLoop)
    mock.tools = tools
    mock.max_tool_response_length = max_tool_response_length
    mock.tool_response_truncate_side = tool_response_truncate_side
    mock._call_tool = ToolAgentLoop._call_tool.__get__(mock, ToolAgentLoop)
    return mock


async def test_left_truncation_keeps_response_tail(self):
    # 构造一个长工具响应，最后一行是关键结论
    tool_response = (
        "Search results for capital of France:\n"
        "1. Lyon is a major city with a long Roman history.\n"
        "2. Marseille is a large port city in southern France.\n"
        "3. The final retrieved snippet says the capital is Paris.\n"
        "Final answer: Paris"
    )
    tools = {"search": FakeLongResponseTool("search", tool_response)}
    loop = _make_tool_agent_loop(tools, max_tool_response_length=19, tool_response_truncate_
    side="left")
    tool_call = FakeFunctionCall(name="search", arguments="{}")
    response, reward, _ = await loop._call_tool(tool_call, {}, self.agent_data)
    assert reward == 1.0
    assert response.text.startswith("(truncated)...")
    # 确保尾部关键结论保留
    assert response.text.endswith("Final answer: Paris")
```

评论区精华

仅有一条来自 gemini-code-assist[bot] 的评论，指出当 self.max_tool_response_length 为 0 时，Python 负索引切片 text[-0:] 会返回整个字符串而非空字符串，可能导致截断失效。建议改用 text[len(text) - n:] 以正确处理零长度边界情况。该评论未被采纳或进一步讨论，PR 随后由 wuxibin89 批准合并。

- 左截断在 max_tool_response_length=0 时的边界行为 (correctness): 未采纳该建议，PR 已合并。

风险与影响

- 风险:

1. 边界条件风险：当 max_tool_response_length 设为 0 时，当前使用 text[-0:] 会返回完整字符串而非空串，截断标记 '(truncated)...' 会错误地拼接到完整文本前。虽然后续可通过其他 PR 修复，但当前版本在极端配置下仍存在潜在问题。
2. 回归风险低：变更仅涉及两行代码的交换，且测试已覆盖两种截断模式，回归风险较低。
3. 兼容性影响：无 API 变更，已有 YAML 配置无需修改即可获得正确语义。 - 影响：影响范围：所有使用多轮工具调用且配置了 tool_response_truncate_side 的训练 / 推理流程。影响程度：中。对于已配置 left 截断的用户，修复前会静默丢失尾部关键工具结果，导致模型行为异常和训练信号错误；修复后行为正确，但可能改变历史 rollouts 的重放结果（需注意兼容性）。测试覆盖：新增的单元测试在 CPU 上运行，覆盖了 'left' 和 'right' 两种模式，但未测试中心截断（'center' 分支）和 max_tool_response_length=0 的边界情况。

- 风险标记：边界条件未覆盖（max_tool_response_length=0），核心路径变更

关联脉络

- PR #4918 [agent] char-count vs token-count truncation: PR body 提及该 Issue 讨论字符数与 token 数截断的差异，但本 PR 仅专注于修复截断方向语义。