PR #2213 完整报告

THUDM/slime

Fix tau-bench token deltas for reasoning templates

合并时间：2026-08-12 13:51

原文链接：http://prhub.com.cn/THUDM/slime/pull/2213

执行摘要

- 一句话：修复 tau-bench 推理模板令牌增量与掩码错误
- 推荐动作：值得精读。该 PR 展示了处理『聊天模板重写历史』和『多轮生成前缀掩码』的完整思路，尤其是 fail-closed 设计、独立渲染单条消息并校验后缀、以及用掩码而不是删除来保留生成前缀的做法。对使用 Qwen3 推理模板的 tau-bench 类多轮训练场景具有直接参考价值。

功能与动机

PR body 指出根因：tau-bench 示例对每条新追加消息使用 curr[len(prev):]，但 Qwen3 官方聊天模板在新真实用户到达后会移除旧 assistant 推理内容，导致 curr 不再以 prev 开头，按旧长度切片会截断新用户；且首次 assistant 生成前缀在初始 prompt 中已存在，但后续前缀未追加到累积 GRPO token 流，导致后续回合遗漏 <lim_start|>assistant\n<think>。

实现拆解

1. 新增独立辅助模块 examples/tau-bench/token_delta.py：定义 get_token_delta(tokenizer, messages, *, include_generation_prompt=False)，集中处理两种分支：assistant 回合验证 curr 是否扩展 generation_prompt，并在 include_generation_prompt=True 时用掩码区分前缀（0）与续写（1）；非 assistant 回合优先走 append-only 快路径，若历史被重写则单独渲染最后一条 user 消息并校验其为渲染结果的精确后缀，否则 fail closed。
2. 改造 examples/tau-bench/trainable_agents.py 调用链：asolve 中调用 _get_token_delta 时传入 include_generation_prompt=bool(response_token_ids)，使只有存在已累积响应 token 时才在后续 assistant 回合附加生成前缀；_get_token_delta 方法体替换为直接委托 token_delta.get_token_delta，删除原手写两分支切片逻辑，统一入口避免逻辑漂移。
3. 新增测试文件 tests/test_tau_bench_token_delta.py（207 行）：以字节级编码的最小 tokenizer 模拟历史重写（HistoryRewritingTokenizer 会剥离旧 reasoning）和跨边界 BPE 合并（BoundaryMergingTokenizer 将 >\n 合并为单 token），覆盖新用户不截断、assistant 前缀保留、跨回合掩码正确性等回归场景；测试通过 importlib 动态加载 token_delta.py，避免将 examples 目录纳入包路径。

关键文件：

- examples/tau-bench/token_delta.py（模块 tau-bench 示例；类别 source；类型 core-logic；符号 get_token_delta）：新增核心 token delta 计算模块，集中处理历史重写、生成前缀掩码和 BPE 边界，是本次修复的主逻辑。
- examples/tau-bench/trainable_agents.py（模块 tau-bench 示例；类别 source；类型 core-logic；符号 _get_token_delta）：接入新 token delta 逻辑，按是否已有响应 token 决定是否附加生成前缀，并删除重复实现。
- tests/test_tau_bench_token_delta.py（模块 测试；类别 test；类型 test-coverage；符号 _load_get_token_delta, HistoryRewritingTokenizer, _strip_reasoning, BoundaryMergingTokenizer）：新增 207 行测试，用最小 tokenizer 模拟历史重写和 BPE 边界合并，覆盖本次修复的主要回归场景。

关键符号：get_token_delta, _get_token_delta, _load_get_token_delta, HistoryRewritingTokenizer.apply_chat_template, BoundaryMergingTokenizer.encode

关键源码片段

examples/tau-bench/token_delta.py

新增核心 token delta 计算模块，集中处理历史重写、生成前缀掩码和 BPE 边界，是本次修复的主逻辑。

```
# examples/tau-bench/token_delta.py
from typing import Any


def get_token_delta(
    tokenizer: Any,
    messages: list[dict[str, Any]],
    *,
    include_generation_prompt: bool = False,
) -> tuple[list[int], list[int]]:
    """Return the tokens and loss mask contributed by the last chat message."""
    if not messages:
        raise ValueError("Cannot calculate a token delta for an empty conversation")

    is_assistant = messages[-1]["role"] == "assistant"
    curr = tokenizer.apply_chat_template(messages, add_generation_prompt=False, tokenize=
    False)

    if is_assistant:
        # 计算去掉最后一条 assistant 消息前后的渲染结果，并验证
        # generation_prompt 与 curr 的扩展关系，防止模板意外重写历史。
        prev = tokenizer.apply_chat_template(messages[:-1], add_generation_prompt=False,
        tokenize=False)
        generation_prompt = tokenizer.apply_chat_template(messages[:-1], add_generation_
        prompt=True, tokenize=False)
        if not generation_prompt.startswith(prev):
            raise ValueError("Adding the assistant generation prompt rewrote the rendered
```

```
            conversation")
        if not curr.startswith(generation_prompt):
            raise ValueError("The assistant response does not extend its generation prompt")

        generation_prompt_text = generation_prompt[len(prev):]
        if not include_generation_prompt:
            # 不保留生成前缀时只取续写部分，全部作为训练 token（掩码 1）。
            new_text = curr[len(generation_prompt):]
            new_tokens = tokenizer.encode(new_text, add_special_tokens=False)
            return new_tokens, [1] * len(new_tokens)

        # 保留生成前缀：前缀用掩码 0 屏蔽（避免 loss 重复计算），续写用掩码 1。
        # 由于 BPE 可能跨前缀边界合并，这里按独立编码的前缀长度取 min，
        # 保证不会因边界合并超出实际 token 数。
        new_text = curr[len(prev):]
        new_tokens = tokenizer.encode(new_text, add_special_tokens=False)
        generation_prompt_length = len(tokenizer.encode(generation_prompt_text, add_special_
        tokens=False))
        masked_prefix_length = min(generation_prompt_length, len(new_tokens))
        loss_mask = [0] * masked_prefix_length
        loss_mask.extend([1] * (len(new_tokens) - masked_prefix_length))
        return new_tokens, loss_mask

    # 非 assistant（用户 / 工具 / 环境）回合：优先走 append-only 快路径。
    prev = tokenizer.apply_chat_template(messages[:-1], add_generation_prompt=False, tokenize=
    False)

    if curr.startswith(prev):
        new_text = curr[len(prev):]
    elif messages[-1]["role"] == "user":
        # Qwen3 等推理模板在新用户到来时会重写历史（隐藏旧的推理内容），
        # 此时不能按旧长度切片，而是独立渲染最后一条 user 消息，
        # 并确认它是完整渲染结果的精确后缀，否则 fail closed。
        new_text = tokenizer.apply_chat_template(
            [messages[-1]],
            add_generation_prompt=False,
            tokenize=False,
        )
        if not curr.endswith(new_text):
            raise ValueError("The latest user message is not a standalone suffix of the rendered
            conversation")
    else:
        raise ValueError("The chat template rewrote history while calculating a non-user token
        delta")

    new_tokens = tokenizer.encode(new_text, add_special_tokens=False)
    return new_tokens, [0] * len(new_tokens)
```

examples/tau-bench/trainable_agents.py

接入新 token delta 逻辑，按是否已有响应 token 决定是否附加生成前缀，并删除重复实现。

```
# examples/tau-bench/trainable_agents.py（关键改动片段）
from token_delta import get_token_delta

# ... 在 asolve 中，assistant 响应追加后：
# 只有已经累积了响应 token 时（bool(response_token_ids)），
# 才在后续 assistant 回合保留生成前缀，并用掩码 0 屏蔽前缀、掩码 1 标记续写。
messages.append({"role": "assistant", "content": response})
assistant_token_ids, assistant_loss_mask = self._get_token_delta(
    state.tokenizer,
    messages,
    include_generation_prompt=bool(response_token_ids),
)
response_token_ids.extend(assistant_token_ids)
loss_masks.extend(assistant_loss_mask)

# ... 原方法体替换为统一委托：
def _get_token_delta(
    self,
    tokenizer: AutoTokenizer,
    messages: list[dict],
    *,
    include_generation_prompt: bool = False,
) -> tuple[list[int], list[int]]:
    """Calculate token delta for multi-turn conversations."""
    return get_token_delta(
        tokenizer,
        messages,
        include_generation_prompt=include_generation_prompt,
    )
```

tests/test_tau_bench_token_delta.py

新增 207 行测试，用最小 tokenizer 模拟历史重写和 BPE 边界合并，覆盖本次修复的主要回归场景。

```
# tests/test_tau_bench_token_delta.py（关键测试结构）
class HistoryRewritingTokenizer:
    """Minimal chat template that hides old reasoning after a new user turn."""

    @staticmethod
    def _strip_reasoning(content: str) -> str:
        # 模拟 Qwen3 模板：新用户到来后剥离旧 assistant 的 <think> 推理。
        return re.sub(r"<think>.*?</think>", "", content, flags=re.DOTALL)

    def apply_chat_template(self, messages, *, add_generation_prompt, tokenize):
        assert tokenize is False
        last_user = max((i for i, message in enumerate(messages) if message["role"] == "user"),
        default=-1)
```

```
        rendered = []
        for i, message in enumerate(messages):
            content = message["content"]
            if message["role"] == "assistant" and i < last_user:
                content = self._strip_reasoning(content)
            rendered.append(f'<{message["role"]}>{content}</{message["role"]}>')
        if add_generation_prompt:
            rendered.append("<assistant>")
        return "".join(rendered)

    @staticmethod
    def encode(text, *, add_special_tokens):
        assert add_special_tokens is False
        return list(text.encode())

    @staticmethod
    def decode(token_ids):
        return bytes(token_ids).decode()


class BoundaryMergingTokenizer(HistoryRewritingTokenizer):
    """Tokenizer where the generation-prefix tail merges with a leading newline."""

    @staticmethod
    def encode(text, *, add_special_tokens):
        assert add_special_tokens is False
        raw = text.encode()
        token_ids = []
        index = 0
        while index < len(raw):
            # 将 ">\n" 合并为单 token，模拟 BPE 跨前缀边界合并。
            if raw[index : index + 2] == b">\n":
                token_ids.append(1000)
                index += 2
            else:
                token_ids.append(raw[index])
                index += 1
        return token_ids
```

评论区精华

该 PR 无 review 评论和讨论线程。实现通过 PR body 和提交信息自带验证说明：5 个 pytest 用例通过，并对比了 Qwen3-4B-Instruct-2507 与严格 Epoch-4 Qwen3-4B Thinking checkpoint 的 token ID 和掩码，与 Slime 的 MultiTurnLossMaskGenerator(..., tokenizer_type="qwen3") 完全一致。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. fail-closed 行为：新逻辑在历史重写且非 user 消息时直接抛 ValueError，若真实 tokenizer 存在未预料的模板重写模式（如工具消息触发重写）会导致 rollout 中断。
  2. 掩码语义耦合：min(generation_prompt_length, len(new_tokens)) 依赖前缀 token 数等于生成前缀文本独立编码的 token 数，当 BPE 跨边界合并时该近似仍可能掩盖部分续写 token，测试中的 BoundaryMergingTokenizer 仅覆盖了 >\n 一种合并形态。
  3. 变更范围隔离：修复仅作用于 examples/tau-bench 下的 tau-bench agent 训练路径，不影响 slime 主 rollout 管线，但 token_delta.py 通过相对导入（from token_delta import ...）依赖运行目录，若示例被以包方式导入可能失败。 - 影响：影响范围限定在 examples/tau-bench 的 tau-bench 示例 agent 训练：修复后累计的 GRPO 训练 token 流在每个真实用户回合都保留完整用户输入，并在每个 assistant 回合保留生成前缀，前辍 token 掩码为 0（不参与 loss），确保 RL 训练对多轮推理模板的 token 和掩码与 rollouts 完全对齐。对使用 Qwen3 等会重写历史模板的模型训练质量有直接改进，对普通模板（append-only）行为保持兼容。团队维护者可借此参考更健壮的 token delta 计算模式。 - 风险标记：fail-closed 异常可能中断 rollout, BPE 边界掩码近似，示例目录导入路径依赖，缺少真实 tokenizer 集成测试

关联脉络

- PR #2184 sync source_names: 同属 rollout 数据管线改进，涉及 slime/ray/rollout.py 与多轮对话数据处理，本 PR 修复 tau-bench 示例中的 token 流问题，与数据源头追踪同属训练数据质量保障。
- PR #2220 Optimize update weight: 涉及 MoE 权重更新与 rollout 侧 token 处理，类似地关注多轮 rollout 中 token 流与掩码的精确性，同属训练管线正确性维护。
- PR #2250 Add lightweight rollout hooks and sampling controls: 新增 rollout 采样钩子与动态采样回退，与 tau-bench 的 token delta 修复同属 rollout 数据质量和训练稳定性方向，且都影响累积 token 流的正确性。