PR #2237 完整报告

THUDM/slime

fix: keep dataset order in filter_long_prompt for mixed multimodal data

合并时间：2026-08-12 13:35

原文链接：http://prhub.com.cn/THUDM/slime/pull/2237

执行摘要

- 一句话：修复 filter_long_prompt 混合多模态数据顺序错乱
- 推荐动作：值得精读：修复点集中、测试设计精巧（用替身与桩避免引入 transformers），是优化引入回归后最小代价恢复正确性的典型样例。数据管线与 VLM 训练相关同学建议细看 filter_long_prompt 的实现与测试写法。

功能与动机

PR body 指出：'With a processor configured, filter_long_prompt reorders the dataset — even when it filters nothing out.' 示例输入 p0..p5 输出为 p1,p3,p5,p0,p2,p4。由于 --rollout-shuffle 默认为 False，该顺序即训练顺序，导致 'A VLM run over a mixed text/image dataset trains every text-only prompt before it ever sees an image.' 这是 #1662 吞吐优化引入的回归，本 PR 旨在保留优化同时恢复正确顺序。

实现拆解

1. 在 slime/utils/data.py 的 filter_long_prompt 中，将分组循环改为 for position, sample in enumerate(origin_samples)，text_only 与 multimodal 列表元素变为 (position, sample) 元组，为后续恢复顺序保留原始位置信息。
2. 将原 filtered_samples 改名为 kept，两组分支中追加 (position, sample)；两组处理完后用 kept.sort(key=lambda position_and_sample: position_and_sample[0]) 按原始位置排序，再解包成 filtered_samples 返回。排序只在数据集构造时执行一次，不在热路径上。
3. 无 processor 的 else 分支天然按 origin_samples 顺序处理，无需修改。
4. 新增 tests/test_filter_long_prompt.py，使用 _Tokenizer/_Processor 替身和 stub_process_vision_info 桩模块，覆盖无过滤保持顺序、混合过滤保持顺序、单模态批量和无 processor 路径四类场景，其中两类在 main 上会失败。
5. 在 .github/workflows/pr-test.yml.j2 模板及生成的 pr-test.yml 中将 test_filter_long_prompt.py 注册到 cpu-unittest 作业（num_gpus: 0），确保 CI 覆盖。

关键文件：

- slime/utils/data.py（模块 数据过滤；类别 source；类型 core-logic；符号 filter_long_prompt）：核心修复文件：filter_long_prompt 在 processor 分支中通过携带原始位置并排序，恢复混合多模态数据集的原始顺序。

- tests/test_filter_long_prompt.py（模块 数据过滤；类别 test；类型 test-coverage；符号 stub_process_vision_info, _Tokenizer, _Processor, _encoded_length）：新增 CPU 单元测试，覆盖顺序保持的核心场景，锁定回归行为。
- .github/workflows/pr-test.yml（模块 持续集成；类别 infra；类型 infrastructure）：将新增测试注册到 cpu-unittest 作业矩阵，保证 CI 覆盖。
- .github/workflows/pr-test.yml.j2（模块 持续集成；类别 infra；类型 infrastructure）：workflow 模板同步添加测试条目，保证生成的 workflow 文件一致。

关键符号：filter_long_prompt, stub_process_vision_info, test_preserves_order_when_nothing_is_filtered, test_preserves_order_when_some_are_filtered, test_single_modality_batches_are_unchanged, test_no_processor_path_still_preserves_order

关键源码片段

slime/utils/data.py

核心修复文件：filter_long_prompt 在 processor 分支中通过携带原始位置并排序，恢复混合多模态数据集的原始顺序。

```
def filter_long_prompt(origin_samples: list[Sample], tokenizer, processor, max_length: int | None) -> list[Sample]:
    # max_length 为 None 时直接返回，由调用方决定是否过滤。
    if max_length is None:
        return origin_samples

    # prompt 不是字符串（如 list 形式）时跳过过滤，并提示配置 apply_chat_template。
    if not isinstance(origin_samples[0].prompt, str):
        logger.warning(
            'Skipping max_length check for list prompt. '
            'Set apply_chat_template=True to enable length filtering.'
        )
        return origin_samples

    if processor:
        # 为了吞吐量，把样本分为纯文本和多模态两组：
        # 纯文本用批量 tokenizer 一次打分，多模态用 processor 逐条打分。
        # 分组时同时记录原始 position，便于之后恢复数据集顺序。
        text_only = []
        multimodal = []
        for position, sample in enumerate(origin_samples):
            if sample.multimodal_inputs and any(v is not None for v in sample.multimodal_inputs.
            values()):
                multimodal.append((position, sample))
            else:
                text_only.append((position, sample))

        kept = []
```

```
        if text_only:
            # 纯文本组：一次调用 tokenizer，得到所有样本的 input_ids。
            prompts = [s.prompt for _, s in text_only]
            input_ids_list = tokenizer(prompts, add_special_tokens=False)['input_ids']
            for (position, sample), input_ids in zip(text_only, input_ids_list, strict=True):
                if len(input_ids) <= max_length:
                    kept.append((position, sample))

        if multimodal:
            # 多模态组：processor 需要按样本提取视觉信息，只能逐个处理。
            from slime.utils.processing_utils import process_vision_info
            for position, sample in multimodal:
                multimodal_inputs = process_vision_info(sample.prompt, processor)
                processor_output = processor(text=sample.prompt, **multimodal_inputs)
                input_ids = processor_output['input_ids'][0]
                if len(input_ids) <= max_length:
                    kept.append((position, sample))

        # 两组分开评分只是优化，必须按原始位置排序恢复数据集顺序；
        # 否则未开启 --rollout-shuffle 时，训练会先遍历纯文本，再遍历多模态。
        kept.sort(key=lambda position_and_sample: position_and_sample[0])
        filtered_samples = [sample for _, sample in kept]
    else:
        # 没有 processor 时所有样本都走批量 tokenizer，天然保持原始顺序。
        prompts = [sample.prompt for sample in origin_samples]
        input_ids_list = tokenizer(prompts, add_special_tokens=False)['input_ids']
        filtered_samples = [
            sample
            for sample, input_ids in zip(origin_samples, input_ids_list, strict=True)
            if len(input_ids) <= max_length
        ]

    logger.info(
        f'Filtered {len(origin_samples) - len(filtered_samples)} samples longer than max_length=
        {max_length}.'
    )
    return filtered_samples
```

tests/test_filter_long_prompt.py

新增 CPU 单元测试，覆盖顺序保持的核心场景，锁定回归行为。

```
# slime.utils.processing_utils 会拉起 transformers 和 PIL，
# CPU 测试镜像没有这些依赖；多模态分支只用到 process_vision_info，
# 所以这里用 types.ModuleType 构造一个假模块桩掉它。
@pytest.fixture
def stub_process_vision_info(monkeypatch):
    import sys
    import types
    module = types.ModuleType('slime.utils.processing_utils')
```

```
    module.process_vision_info = lambda prompt, processor: {'images': None}
    monkeypatch.setitem(sys.modules, 'slime.utils.processing_utils', module)


class _Tokenizer:
    # 批处理 tokenizer 替身：prompt 'pN:len' 编码为 len 个 id。
    def __call__(self, prompts, add_special_tokens=False):
        return {'input_ids': [[0] * _encoded_length(p) for p in prompts]}


class _Processor:
    # 逐样本 processor 替身，长度约定与 _Tokenizer 一致。
    def __call__(self, text=None, **kwargs):
        return {'input_ids': [[0] * _encoded_length(text)]}


def _encoded_length(prompt: str) -> int:
    # prompt 形如 'pN:len'，冒号后即编码长度，用整型解析出来。
    return int(prompt.split(':')[1])


@pytest.mark.unit
def test_preserves_order_when_some_are_filtered(stub_process_vision_info):
    # 混合纯文本与多模态样本，部分超长被过滤。
    # 规格为 ( 是否多模态 , 编码长度 )，max_length=100。
    specs = [
        (True, 5), # p0 多模态，保留
        (False, 500), # p1 纯文本，丢弃
        (False, 5), # p2 纯文本，保留
        (True, 500), # p3 多模态，丢弃
        (False, 5), # p4 纯文本，保留
        (True, 5), # p5 多模态，保留
    ]
    samples = _make_samples(specs) # _make_samples 按 specs 构造 Sample 列表

    kept = filter_long_prompt(samples, _Tokenizer(), _Processor(), max_length=100)

    # 关键断言：幸存样本必须保持原始数据集顺序，而不是先纯文本后多模态。
    assert [s.prompt for s in kept] == ['p0:5', 'p2:5', 'p4:5', 'p5:5']
```

评论区精华

该 PR 无公开 review 评论。PR body 中作者说明了设计权衡：分组评分是 #1662 引入的吞吐优化，不能因为修复顺序而放弃；因此采用携带原始位置 + 结束后排序的方式，既保留批量 tokenizer 的高吞吐，又让返回顺序与原始数据集一致。代码注释也强调 'The two groups are scored separately for throughput, so restore the dataset order here'。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 行为变更：修复会让依赖旧顺序（纯文本在前）的实验结果变化，这是修复目标，但对比实验需注意基线差异。
  2. 边界条件：filter_long_prompt 对空列表仍会因 origin_samples[0] 抛 IndexError，本次未处理，延续原有行为。
  3. 性能：排序 O(n log n) 仅在数据集构造时执行一次，可忽略；分组与 scoring 逻辑不变。
  4. 稳定性：kept.sort 使用稳定排序且 position 唯一，结果确定；无 processor 分支不受影响。 - 影响：影响面：所有配置了 processor 且数据混合文本与多模态的训练 run，训练顺序从纯文本全在前修正为原始数据顺序，更符合数据分布，利于 VLM 训练稳定。对纯文本数据集或未配置 processor 的场景零影响。接口、配置项、schema 无变化；CI 增加一个 CPU 测试，团队维护成本极小。 - 风险标记：训练顺序行为变更，空数据集边界未处理

关联脉络

- 暂无明显关联 PR