PR #31639 完整报告

sgl-project/sglang

[Fix] Account zero-logprob sequences correctly in chunked logprob stitching

合并时间：2026-07-18 13:33

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/31639

执行摘要

- 一句话：修复分块 logprob 拼接中零 logprob 行序列计数错误
- 推荐动作：该 PR 值得精读，尤其适合关注边界条件处理、分块算法和测试设计的开发者。四个修复点各有巧妙，测试方法（枚举所有组合对比）值得借鉴。建议在 review 时仔细理解每个修正的动机和实现。

功能与动机

PR body 明确指出修复 scheduler 崩溃：AssertionError: len(req.logprob.input_top_logprobs_val) == relevant_tokens_len 在 logprob_result_processor.py 中出现。零 logprob 行序列是正常输入，但 chunked walkers 处理不当导致计数漂移。

实现拆解

1. 空 chunk 不再跳过：原代码中 if chunk_indices.numel() == 0: continue 导致纯 sample-only 行的 chunk 完全不执行 per-sequence bookkeeping，现在移除该跳过，保证每个序列的占位条目都能被发射。
2. 修正 chunk slice 边界：token_to_seq_idx[end_idx] 属于下一个 chunk，导致零行序列恰好位于分块边界时被两个 chunk 重复计数；slice 改为在 chunk 内最后一行结束。
3. 修复发射守卫：walker 中的 if len(val) > 0 条件会丢弃拥有行但 token_ids=None 的序列条目（opt-out），且若该序列跨 chunk，会将其值扩展到前一个序列的条目上；现在改为仅当 split-continuation 时才 extend，否则强制发射（可能为空）。
4. 处理 None token_ids：get_token_ids_logprobs_raw 的 prefill 分支在混合 batch 中遇到 token_ids=None 时崩溃 TypeError: torch.tensor(None)；新增分支发射空条目同时步进行光标。
5. 配套测试：新增 test_logprob_chunk_stitching.py，使用 _build_batch 生成 (extend_len, start) 组合，配合异质 top_logprobs_num 和 token_ids_logprobs（包括 None/[]/ 列表），在多个 chunk_size 下对比 chunked 与非 chunked 输出。

关键文件：

- python/sglang/srt/layers/logprob_processor.py（模块 logprob 处理；类别 source；类型 core-logic；符号 get_token_ids_logprobs_raw, get_top_logprobs_chunk, get_token_ids_logprobs_chunk, process_input_logprobs_by_chunk）：修改了 4 个关键

函数，修复分块 logprob 拼接中零 logprob 行序列的计数错误

- test/registered/unit/layers/test_logprob_chunk_stitching.py（模块 测试套件；类别 test；类型 test-coverage；符号 _build_batch, _run, get_logits_fn, TestLogprobChunkStitching）：新增 143 行回归测试，枚举 11k+ batch shapes 验证分块拼接的正确性

关键符号：get_token_ids_logprobs_raw, get_top_logprobs_chunk, get_token_ids_logprobs_chunk, process_input_logprobs_by_chunk

关键源码片段

python/sglang/srt/layers/logprob_processor.py

修改了 4 个关键函数，修复分块 logprob 拼接中零 logprob 行序列的计数错误

```
def get_top_logprobs_chunk(
    logprobs: torch.Tensor,
    logits_metadata: LogitsMetadata,
    top_k_nums: List[int],
    pruned_lens: List[int],
    input_top_logprobs_val: List,
    input_top_logprobs_idx: List,
    split_pruned_len: int,
) -> int:
    """Get top-k logprobs for each sequence in the chunk. 返回下一个chunk的剩余token数。"""
    # 修复 1: 删除原有 `if logprobs.shape[0] == 0: return 0`,
    # 确保零行 chunk 也能执行后续循环，为每条序列发射空占位条目。
    max_k = max(logits_metadata.top_logprobs_nums)
    ret = logprobs.topk(max_k, dim=1)
    values = ret.values.tolist()
    indices = ret.indices.tolist()

    pt = 0
    next_split_pruned_len = 0
    for n, (k, pruned_len) in enumerate(zip(top_k_nums, pruned_lens)):
        if n == 0:
            pruned_len -= split_pruned_len
        else:
            split_pruned_len = 0

        if pruned_len <= 0:
            input_top_logprobs_val.append([])
            input_top_logprobs_idx.append([])
            continue

        val = []
        idx = []
        for j in range(pruned_len):
            if pt + j >= len(values):
```

```
                next_split_pruned_len = split_pruned_len + j
                break
            val.append(values[pt + j][:k])
            idx.append(indices[pt + j][:k])

        # 修复 2: 将 `if len(val) > 0:` 改为直接基于 split_pruned_len 判断，
        # 即使 val 为空（零 logprob 行序列）也发射空条目，避免被跳过。
        if split_pruned_len > 0:
            input_top_logprobs_val[-1].extend(val)
            input_top_logprobs_idx[-1].extend(idx)
        else:
            input_top_logprobs_val.append(val)
            input_top_logprobs_idx.append(idx)

        pt += pruned_len
    return next_split_pruned_len

# get_token_ids_logprobs_raw 的 prefill 分支（部分）
else: # prefill
    pt = 0
    for i, (token_ids, pruned_len) in enumerate(
        zip(token_ids_logprobs_list, extend_logprob_pruned_lens_cpu)
    ):
        if pruned_len <= 0:
            vals.append([])
            idxs.append([])
            continue
        # 修复 3: 添加 token_ids 为 None 的处理，避免 `torch.tensor(None)` 崩溃。
        # 序列的行仍然占据 logprobs 内存，因此需要步进 pt。
        if token_ids is None:
            vals.append([])
            idxs.append([])
            pt += pruned_len
            continue
        token_ids_tensor = torch.tensor(token_ids, dtype=torch.long).to(
            logprobs.device, non_blocking=True
        )
        pos_logprobs = logprobs[pt : pt + pruned_len, token_ids_tensor]
        vals.append(pos_logprobs if no_copy_to_cpu else pos_logprobs.tolist())
        idxs.append([token_ids for _ in range(pruned_len)])
        pt += pruned_len
```

评论区精华

本 PR 由作者自行审查并合并，无公开 review 评论。作者通过多次 CI rerun（如 /rerun-test）运行了新增单元测试、端到端测试以及其他 logprob 相关测试（test_original_logprobs、test_lora_hf_sgl_logprob_diff 等），全部通过，验证了修复的正确性。

- CI 验证测试覆盖 (testing): 修复通过验证

风险与影响

- 风险：核心风险在于对分块 logprob 拼接中边界条件的调整可能影响正常 logprob 路径的计数逻辑。但新增的测试覆盖 11k+ 种组合（包括零行、None token_ids、空 token_ids 等），并对比非分块参考输出，大大降低了回归风险。此外，删除了 if logprobs.shape[0] == 0: return 0 的早期返回，使得空 chunk 仍执行后续循环，可能带来微小性能开销，但鉴于空 chunk 本身不常有，且操作简单，影响可以忽略。
- 影响：影响面主要限于使用 logprob 功能并且在混合 batch 中包含 logprob opt-out 或分块预填充的用户。修复后避免了 scheduler 崩溃，提高了系统稳定性。对于未使用 logprob 或不分块的用户无影响。
- 风险标记：核心路径变更，边界条件复杂，测试覆盖依赖

关联脉络

- PR #20071 refactor logprob processor layer: 同一文件 logprob_processor.py 的重构，涉及 logprob 处理逻辑的基础调整，与本 PR 的输入 logprob 修复相关
- PR #31624 [Refactor] Move output logprob processing into the logprob_processor layer: 将输出 logprob 处理移入 logprob_processor 层，与本 PR 的输入 logprob 修复有所关联