PR #21723 完整报告

sgl-project/sglang

[BugFix] Fix EAGLE speculative decoding missing grammar-based finish ⋯

合并时间：2026-04-15 03:43

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/21723

执行摘要

- 一句话：修复 EAGLE 推测解码中基于语法的请求完成检测缺失导致的调度错误。
- 推荐动作：该 PR 值得精读，因为它揭示了推测解码中语法完成检测与请求状态同步的关键设计问题。关注点包括：验证循环内状态更新的顺序重要性，以及不同推测算法（EAGLE vs ngram）在数据结构设计上的差异如何影响错误修复范围。

功能与动机

根据 PR body 描述，当使用 EAGLE 推测解码（speculative-algo NEXTN）并启用结构化输出（如 JSON 模式）时，批处理请求会失败并抛出错误："ValueError: length of new_indices: 1 != length of topk_p: 2, this should not happen"。根本原因是 EagleVerifyInput.verify 函数中，在检查 req.finished() 之前调用 grammar.accept_token()，且语法状态仅在请求未完成时更新。这导致当语法推进到完成状态时，请求未被正确标记为已完成，从而在后续调度中引发数据不一致。特别是在 ignore_eos 设置为 true 时，EOS 令牌不再触发早期终止，语法成为检测请求完成的主要机制，显著增加了触发此错误的概率。

实现拆解

1. 核心逻辑调整：修改 python/sglang/srt/speculative/eagle_info.py 中的 verify 函数。将 grammar.accept_token(id) 的调用从 else 分支移至 req.finished() 检查之前，并添加额外的 req.check_finished() 调用，以确保语法驱动的完成检测能及时更新请求状态。
2. 控制流重构：移除原有的 else 分支结构，改为在 req.check_finished() 后直接检查 if not req.finished() and req.grammar is not None，然后执行语法接受和再次完成检查。这样保证了无论请求是否通过语法完成，其状态都能在循环内被正确设置。
3. 影响范围确认：在 review 讨论中，确认了 ngram 推测解码不存在相同问题，因为 ngram 不依赖 unfinished_accept_index 等数据结构，下游逻辑由掩码驱动且已在循环内截断，因此无需类似修复。

关键文件：

- python/sglang/srt/speculative/eagle_info.py（模块 推测解码；类别 source；类型 core-logic；符号 verify）：这是修复的核心文件，包含 EAGLE 推测解码的验证逻辑，直接修改了状态检测顺序以解决语法完成缺失问题。

关键符号：verify

关键源码片段

python/sglang/srt/speculative/eagle_info.py

这是修复的核心文件，包含 EAGLE 推测解码的验证逻辑，直接修改了状态检测顺序以解决语法完成缺失问题。

```
# 在验证循环中，针对每个接受的令牌进行处理
for j, idx in enumerate(accept_index_row):
    if idx == -1:
        break
    num_accepted += 1
    id = predict_cpu[idx]
    req.output_ids.append(id)
    if req.require_reasoning and think_end_id is not None:
        req.update_reasoning_tokens(id, think_end_id)
    req.check_finished() # 首先检查请求是否已完成（例如通过EOS）

    # 关键修复：如果请求未完成且有语法约束，则接受令牌并再次检查完成状态
    if not req.finished() and req.grammar is not None:
        try:
            req.grammar.accept_token(id) # 更新语法状态，可能使请求完成
        except ValueError as e:
            logger.info(f"{i=}, {req=}\n{accept_index=}\n{predict=}")
            raise e
        req.check_finished() # 再次检查，确保语法驱动的完成被捕获

    # 检查请求是否已完成（无论是通过EOS还是语法）
    if req.finished():
        has_finished = True
        accept_index[i, j + 1 :] = -1 # 将后续令牌标记为无效
        break
```

评论区精华

reviewer Qiaolin-Yu 提问："Nice fix. Does ngram have the same issue? ... If so, could you also update that? Thanks!" 作者 mingyue300 回复："Thanks. I think ngram does not need the same fix. Unlike EAGLE, ngram doesn't read 'req.finished()' after the loop to build any data that is consumed by the next verify round..." 结论：确认 ngram 不存在相同问题，因此修复仅针对 EAGLE 模块。

- ngram 是否存在相同问题 (correctness): 作者解释 ngram 不依赖 unfinished_accept_index 等数据结构，下游逻辑由掩码驱动且已在循环内截断，因此无需类似修复。

风险与影响

- 风险：1. 回归风险：修改了核心验证逻辑，如果新逻辑顺序有误，可能导致其他完成检测路径（如 EOS 令牌）被错误处理，但变更较小且聚焦于语法完成场景，风险可控。 2. 性能影

响：添加了额外的 req.check_finished() 调用，可能引入微小开销，但在验证循环内，影响可忽略。 3. 兼容性：仅影响启用 EAGLE 推测解码和结构化输出的场景，对默认配置无影响。

- 影响：1. 用户影响：修复了使用 EAGLE 推测解码和结构化输出（如 JSON 模式）时的批处理失败问题，提升功能稳定性和用户体验。 2. 系统影响：解决了调度器中的数据不一致错误，避免因异常导致的请求中断，提高系统可靠性。 3. 团队影响：明确了 EAGLE 与 ngram 在完成检测机制上的差异，为后续推测解码模块的维护提供参考。
- 风险标记：核心路径变更，状态同步风险

关联脉络

- PR #22753 Fix streaming session busy-check double-counting via active_pool_idxs: 同样涉及调度器中的数据一致性问题修复，但针对流式会话内存统计，可对比学习状态同步机制。
- PR #22525 fix: EPLB dispatch OOB when shared experts fusion enabled under DeepEP: 同为 bugfix 类型，涉及索引越界问题，反映仓库在复杂交互场景下的错误处理模式。