PR #31860 完整报告

sgl-project/sglang

Fix dropped tool calls when a stream delta carries several

合并时间：2026-07-21 18:07

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/31860

执行摘要

- 一句话：修复流式 delta 中多个 tool call 被丢弃的问题
- 推荐动作：值得快速合入并关注相关模块（其他 Detector 如 xgrammar 的流式解析是否存在类似模式）。提取 progress-flag 循环的设计可复用。

功能与动机

当 stream-interval 较大或多 token 解码步骤导致单个 delta 包含多个完整 tool call 时，原始实现仅输出第一个调用，其余被静默丢弃。非流式路径（detect_and_parse）已正确迭代所有匹配，这是流式路径独有的漏洞。

实现拆解

1. 从原 parse_streaming_increment 中提取单步处理逻辑为新方法 _parse_buffered_increment，返回 (StreamingParseResult, bool) 二元组，bool 表示本次处理是否推进了缓冲区。
2. 在 parse_streaming_increment 内改用 while True 循环持续调用 _parse_buffered_increment，直到 made_progress 为 False，从而在一个 delta 内排空所有完整调用。同时累积 normal_text 和 calls 并在最后返回合并结果。
3. 调整原方法中的直接 return 语句：在 partial JSON、非 Mapping 载荷等终止路径返回 (StreamingParseResult(), False) 以指示无进展；在正常文本前缀已被剥离、调用现在位于缓冲区头部时返回 (..., True) 以继续循环。
4. 新增三个回归测试覆盖：同一 delta 中两个完整调用、文本后跟完整调用、被拒绝调用后跟有效调用，验证所有调用均正确发出。

关键文件：

- python/sglang/srt/function_call/inkling_detector.py（模块 解析器；类别 source；类型 core-logic；符号 _parse_buffered_increment, parse_streaming_increment）：核心修复文件：提取 _parse_buffered_increment 方法并重构 parse_streaming_increment 为 drain 循环。
- test/registered/unit/function_call/test_function_call_parser.py（模块 测试；类别 test；类型 test-coverage；符号 test_streaming_two_complete_tool_calls_in_one_delta_both_emit, test_streaming_text_then_tool_call_in_one_delta_emits_both, test_streaming_rejected_middle_call_keeps_later_valid_call）：新增三个回归测试，覆盖

修复的主要场景。

关键符号：_parse_buffered_increment, parse_streaming_increment

关键源码片段

python/sglang/srt/function_call/inkling_detector.py

核心修复文件：提取 _parse_buffered_increment 方法并重构 parse_streaming_increment 为 drain 循环。

```
# python/sglang/srt/function_call/inkling_detector.py

def parse_streaming_increment(
    self, new_text: str, tools: List[Tool]
) -> StreamingParseResult:
    # Drain every complete call in the delta: this detector has no
    # stream-end flush, so anything left in self._buffer is lost.
    self._buffer += new_text
    all_calls: list[ToolCallItem] = []
    normal_parts: list[str] = []
    while True:
        result, made_progress = self._parse_buffered_increment(tools)
        if result.normal_text:
            normal_parts.append(result.normal_text)
        if result.calls:
            all_calls.extend(result.calls)
        if not made_progress:
            break
    return StreamingParseResult(
        normal_text="".join(normal_parts),
        calls=all_calls,
    )

def _parse_buffered_increment(
    self, tools: List[Tool]
) -> tuple[StreamingParseResult, bool]:
    # One drain step: emit a text run or one complete call; the bool is
    # whether the buffer advanced (the caller loops while it does).
    current_text = self._buffer

    if self.bot_token not in current_text:
        # ... control token handling, returns (result, False) as before
        ...

    bot_pos = current_text.find(self.bot_token)
    if bot_pos > 0:
        normal_text, self._current_header_name = self._split_trailing_tool_header(
            current_text[:bot_pos]
        )
```

```
            self._buffer = current_text[bot_pos:]
            normal_text = self._clean_normal_text(normal_text)
            if normal_text:
                # prefix stripped, call now at buffer head -> keep draining
                return StreamingParseResult(normal_text=normal_text), True
            current_text = self._buffer

        # ... partial JSON parsing and tool call extraction
        try:
            payload, end_idx = _partial_json_loads(current_text[start_idx:], flags)
        except (MalformedJSON, json.JSONDecodeError):
            return StreamingParseResult(), False
        if not isinstance(payload, Mapping):
            return StreamingParseResult(), False

        # ... build call, check _is_complete_json, etc.
        if not _is_complete_json(json_text):
            return StreamingParseResult(calls=calls), False

        call = self._tool_call_item(payload, tools, self.current_tool_id, ...)
        if call is not None:
            calls.append(call)
        # ... update buffer after consuming complete call
        self._buffer = current_text[start_idx + end_idx:]
        return StreamingParseResult(calls=calls), True
```

评论区精华

该 PR 无公开 review 评论，作者独立完成设计与实现。PR body 清晰描述了问题复现方式（--stream-interval 1000 从 1 个 call 变为 4 个）。

- 暂无高价值评论线程

风险与影响

- 风险：改动集中在 inkling_detector.py 单个文件，影响范围限于 Inkling 协议流式 tool call 解析。循环逻辑确保每次调用消耗缓冲区且终止条件明确，不会出现无限循环。对外接口 parse_streaming_increment 返回值类型不变，调用方无需修改。风险较低。
- 影响：用户体验：修复了多 tool call 响应在流式场景下被截断的问题，使流式输出与非流式输出一致。系统影响：每个 delta 处理中增加少量循环开销，但循环次数通常与调用数量相当，可忽略。测试覆盖增强，降低未来回归风险。
- 风险标记：核心路径变更，缺乏更广泛测试（其他 detector 类似模式）

关联脉络

- 暂无明显关联 PR