PR #51391 完整报告

vllm-project/vllm

[Bugfix][Parser] Prevent Inkling block-end leakage with tools

合并时间：2026-08-09 00:34

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/51391

执行摘要

- 一句话：修复 Inkling 工具启用时块结束 token 泄漏进 content
- 推荐动作：值得精读。核心设计决策是：语义事件（TOOL_CALL_START / TOOL_CALL_END）不能作为词法工具区域的边界依据，应改由状态转移结构推导；同时 transition is None 分支也要同步清理转发状态。这对任何使用共享 parser engine 新增 grammar 的开发者都有直接借鉴意义，也建议后续为嵌套 / 多 wrapper 工具区域补充更细粒度的测试。

功能与动机

issue #51387 报告：启用 --enable-auto-tool-choice --tool-call-parser inkling --reasoning-parser inkling 后，模型在 tool result 之后用纯文本回答（无 thinking 块、无 tool call）时，响应 content 以字面量 <|end_message|> 结尾，agent 框架在每一轮工具调用后的 synthesis 回合都会命中。issue 定位到两处互配根因：一是 StreamingParserEngine._on_terminal 的 skip_tool_parsing 分支刻意把块结束终端透传为 content 供下游 tool pass 消费，但无工具块时 tool pass 不启动，标记直达客户端；二是 DelegatingParser._extract_tool_calls 无工具调用分支返回 pre-tool-pass content，丢弃了 tool parser 已消费掉标记的版本。PR body 还明确本 PR 不重复 #49876（no-tools 纯文本模式）和 #50528（无 thinking 块时的 tool-call 交接）。

实现拆解

1. 变更入口：vllm/parser/engine/streaming_parser_engine.py 的 StreamingParserEngine 是 reasoning/tool 双 pass 共用的底层引擎；skip_tool_parsing=True 时由 reasoning pass 转发工具语法、tool pass 在下一阶段消费，泄漏发生在这条转发路径上。
2. 新增结构推导出口集合：在 __init__ 中新增 _tool_exit_terminals，从 config.transitions 提取所有“工具状态 → 非工具状态”转移对应的 terminal。相比用 TOOL_CALL_END 事件识别，结构推导能覆盖 MiniMax 这类语义事件与词法 wrapper 不一致的 grammar。
3. 新增转发 span 状态：_in_skipped_tool_span 表示 reasoning pass 当前是否处于需要逐字转发的词法工具区域；reset 时清零，保证 parser 跨请求复用不串扰。
4. 改造 _on_terminal 分支：对共享块结束 token，用 is_opener / is_exit / used_as_plain_closer 三段判断——工具区域之外作为普通文本 / 推理结束标记走正常 transition 被消费；区域之内打开 span、完整转发工具语法；对 transition is None 的

wrapper closer 也显式清除 span，避免 stale 状态使后续文本 closer 误走转发路径。

5. 测试与验证配套：tests/parser/engine/test_inkling.py 新增 TestDelegatingTwoPass（served 双 pass 路径，覆盖 streaming/non-streaming、纯文本、推理后文本、多工具往返、parser 复用、两种结束 token、无转移 closer）；tests/parser/engine/test_parser_engine.py 新增 TestSkipToolSpanForwarding（共享引擎层验证 span 转发与清理）。bbrowning 另行跑了 parser engine 全量单测与多个模型的 reasoning/tool parser 测试，共 4064 passed。

关键文件：

- vllm/parser/engine/streaming_parser_engine.py（模块 解析引擎；类别 source；类型 core-logic；符号 StreamingParserEngine.init, StreamingParserEngine.reset, StreamingParserEngine._on_terminal）：共享 parser engine 核心逻辑：新增 _tool_exit_terminals 结构推导、_in_skipped_tool_span 状态与 _on_terminal 分支改造，是本次修复的主体。
- tests/parser/engine/test_inkling.py（模块 解析器；类别 test；类型 test-coverage；符号 _function_tool, _delegating, _stream_delegating, TestDelegatingTwoPass）：新增 TestDelegatingTwoPass 提供 served 双 pass 路径的回归覆盖，直接复现 issue 的 streaming/non-streaming 泄漏场景。
- tests/parser/engine/test_parser_engine.py（模块 引擎测试；类别 test；类型 test-coverage；符号 TestSkipToolSpanForwarding, _skip_engine, test_tool_syntax_forwarded_and_span_cleared, test_span_not_stale_across_two_tool_calls）：新增 TestSkipToolSpanForwarding 在共享引擎层验证 span 转发、跨两次工具调用不 stale、以及 transitionless closer 后的内容消费契约。

关键符号：StreamingParserEngine.init, StreamingParserEngine.reset, StreamingParserEngine._on_terminal, TestDelegatingTwoPass.test_plain_text_non_streaming, TestSkipToolSpanForwarding.test_content_after_transitionless_closer_observable

关键源码片段

vllm/parser/engine/streaming_parser_engine.py

共享 parser engine 核心逻辑：新增 _tool_exit_terminals 结构推导、_in_skipped_tool_span 状态与 _on_terminal 分支改造，是本次修复的主体。

```
# 在 __init__ 中基于状态转移结构推导词法工具区域的出口集合。
# 不用 TOOL_CALL_END 语义事件的原因是：MiniMax 的 </invoke> 会触发
# TOOL_CALL_END，但外层 <minimax:tool_call> wrapper 的关闭没有任何事件；
# 因此统一用“工具状态 -> 非工具状态”的转移来识别出口。
self._tool_exit_terminals: frozenset[str] = frozenset(
    terminal
    for (state, terminal), tr in config.transitions.items()
    if state in self._TOOL_STATES and tr.next_state not in self._TOOL_STATES
)


# reset 时清除上一请求遗留的转发 span，避免 parser 复用时状态串扰。
```

```
self._in_skipped_tool_span = False


def _on_terminal(self, terminal: str, value: str) -> list[SemanticEvent]:
    key = (self.state, terminal)
    transition = self.config.transitions.get(key)

    if transition is None:
        # skip pass 投影后的状态可能没有该 terminal 的转移
        #  (Qwen3 / DeepSeek / GLM 4.7 MoE / Nemotron v3 的 wrapper closer
        # 都只在工具状态定义）。此时依然要清除 span，否则它会在请求
        # 剩余阶段保持 True，后续普通文本 closer 会被误转发并泄漏。
        if self.skip_tool_parsing and terminal in self._tool_exit_terminals:
            self._in_skipped_tool_span = False
        return self._emit_for_state(value)

    if self.skip_tool_parsing and terminal in self._tool_terminals:
        # Inkling 复用同一个块结束 token 作为工具、文本、推理三种出口，
        # 因此不能只按 terminal 身份分类。规则是：词法工具区域之外按正常
        # 转移消费 closer；区域之内则完整转发工具语法（含 wrapper closers）。
        is_opener = transition.next_state in self._TOOL_STATES
        is_exit = terminal in self._tool_exit_terminals
        used_as_plain_closer = (
            is_exit and not is_opener and not self._in_skipped_tool_span
        )
        if not used_as_plain_closer:
            if is_opener:
                self._in_skipped_tool_span = True
            elif is_exit:
                self._in_skipped_tool_span = False
            if self.state == ParserState.MESSAGE_HEADER:
                self.state = ParserState.CONTENT
                self._message_header_buffer = ""
                return [
                    SemanticEvent(
                        EventType.TEXT_CHUNK,
                        value=value,
                        tool_index=self.tool_index,
                    )
                ]
            if EventType.REASONING_END in transition.events:
                self.state = ParserState.CONTENT
                return [
                    SemanticEvent(
                        EventType.REASONING_END,
                        value=value,
                        tool_index=self.tool_index,
                    ),
                    SemanticEvent(
```

```
                    EventType.TEXT_CHUNK,
                    value=value,
                    tool_index=self.tool_index,
                ),
            ]
        content_type = self.config.content_events.get(self.state)
        if content_type is not None:
            return [
                SemanticEvent(
                    content_type,
                    value=value,
                    tool_index=self.tool_index,
                )
            ]
        return []
    # used_as_plain_closer 为 True：这是纯文本 / 推理的结束标记，
    # 落到下方原有路径，按正常转移消费，不再透传给调用方。
# 原有：普通转移、skip_in_token_id_mode 与 _apply_transition 处理保持不变。
```

评论区精华

- 词法区域 vs 语义事件（bbrowning，CHANGES_REQUESTED）：TOOL_CALL_START / TOOL_CALL_END 描述的是语义调用，不一定是 skip pass 必须保留的词法 wrapper。MiniMax 是典型案例：<minimax:tool_call> 打开词法区域时没有 TOOL_CALL_START，</minimax:tool_call> 关闭时也没有 TOOL_CALL_END，因此建议边界改为结构转移推导。
- transitionless closer 的 stale 状态（bbrowning）：Qwen3、DeepSeek v3.2/v4、GLM 4.7 MoE、Nemotron v3 的 wrapper closer 只在工具状态定义，skip pass 位于 CONTENT 时 transition is None 提前返回，会让 passthrough 标志在请求剩余阶段保持 True，普通输出测试可能漏掉。
- 独立验证（bbrowning，APPROVED）：跑了 parser engine 全量单测 + deepseek/gemma4/glm4_moe/minimax_m2/qwen3 等模型的 reasoning/tool parser 测试，4064 passed；live server 复现脚本从 2 pass/4 fail/2 inconclusive 提升到 5 pass/1 fail/2 inconclusive，剩余失败与 #49876、#50528 相关，由作者们协调落地。
- 词法工具区域边界应以结构转移而非语义事件定义 (design): 最终实现采纳结构性边界，新增 _tool_exit_terminals 推导与 _in_skipped_tool_span 跟踪。
- transitionless wrapper closer 会导致 span 状态 stale (correctness): 在 transition is None 分支补上 _tool_exit_terminals 检查并清除 span，新增对应回归测试。
- 回归验证范围 (testing): 已批准；剩余失败与 #49876、#50528 相关，由作者们协调后续落地。

风险与影响

- 风险：streaming_parser_engine.py 是共享引擎，qwen3/deepseek/glm47_moe/nemotron_v3 等所有 engine-backed parser 都经过该路径，本次改动虽经 4064 个单测回归，但 live 环境仍存在 1 个失败用例（与

#49876/#50528 相关），说明 Inkling 整体解析尚不稳定。_in_skipped_tool_span 是新增可变状态，任何提前 return 路径若未正确清理，都可能造成同类泄漏或反向误消费；当前测试覆盖主要场景，但嵌套工具区域等边缘未穷尽。本 PR 未运行实时模型评估，仅 CPU-only 确定性测试，后续接入真实 Inkling 权重时建议再做一次端到端回归。

- 影响：用户侧：启用 Inkling tool-calling 的请求（尤其是 agent 循环中 tool result 之后的 synthesis 回合）不再收到尾部脏标记，流式与非流式同时修复，content 与 reasoning 的划分更符合 OpenAI 兼容协议预期。系统侧：共享 parser engine 的行为变化影响所有 engine-backed parser，但工具区域内转发、区域外消费的语义被保留，回归面集中在新的 span 状态清理逻辑；对推理性能和 GPU 路径无影响。团队侧：与 #49876、#50528 构成 Inkling 解析修复序列，三个 PR 互补，需要协同合入并统一做一次带真实模型的端到端验证。
- 风险标记：共享引擎核心路径变更，无实时模型验证，新增状态变量需跨请求清理

关联脉络

- PR #49876 [Bugfix][Parser] Confirm reasoning end when an Inkling content block opens: 同一 Inkling 解析问题线；PR body 明确说明本 PR 不重复 #49876（no-tools 纯文本模式），二者需要协同合入。