PR #46875 完整报告

vllm-project/vllm

[Parser][Bugfix] Ensure tool call or other special tokens don't leak in non-streaming tool parsing

合并时间：2026-07-01 01:46

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/46875

执行摘要

- 一句话：自动发现特殊 token 并阻止泄漏，修复非流式工具解析 bug
- 推荐动作：值得精读。设计决策值得关注：自动发现 vs 手动配置的权衡，preserve_tokens 机制提供可扩展性，抑制工具调用与跳过语义的分离。建议在引入新解析器时参考 _build_drop_info 和 preserve_tokens 的设计。

功能与动机

从 PR body：特殊令牌仅通过 token ID 过滤，因此在引擎解析器的非流式文本路径上泄漏到响应内容中（skip_special_tokens=False）。每个模型手动维护 drop_tokens 集合和 adjust_request 覆盖来切换 skip_special_tokens 的做法脆弱且需要为每个新模型配置。用自动发现替换：从 tokenizer.all_special_tokens 构建 __DROP__终端，通过现有的 scanner/lexer/state-machine 管道路由，对流式（token ID）和非流式（文本）路径都有效，零每个模型配置。

实现拆解

1. 自动发现 drop 令牌：在 streaming_parser_engine.py 中新增 _build_drop_info 函数，读取 tokenizer.all_special_tokens 和 all_special_ids，与已有终端和 preserve_tokens 比对，未配置的令牌构建为 __DROP__ 终端，注入 LexerShape。移除旧的手动 drop_tokens 集合。
2. 抑制工具调用分离：将 skip_tool_parsing 的职责分离为 _suppress_tool_calls，使状态机正常解析工具（通过转换消耗终端），但过滤掉所有工具事件，确保工具 markup 不泄漏到内容且不返回工具调用对象。
3. 刷新 defer whitespace：在非流式 _single_pass_parse 中传递 finished=True，确保缓冲区中的空白内容被刷新，与流式行为一致。
4. 移除 Gemma4 的手动配置：移除 gemma4.py 中的 _GEMMA4_MODEL_DROP_TOKENS 和条件性的 adjust_request 覆盖，改为使用 preserve_tokens 保留 <|"|> 字符串定界符。测试同步更新。

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 _DropInfo, _build_drop_info）：核心变更文件，引入自动 drop 令牌的 _DropInfo 和 _build_drop_info，重构 __init__集成新机制，移除了旧的 drop_token_ids

逻辑。

- tests/parser/engine/test_parser_engine.py（模块 测试覆盖；类别 test；类型 test-coverage；符号 TestDropSpecialTokens, test_drops_special_token_by_id_from_content, test_drops_special_token_by_id_from_reasoning, test_drops_via_text_fallback_when_no_token_ids）：新增 TestDropSpecialTokens 类，全面测试自动 drop 机制在各种场景下的正确性，包括 token ID 路径、文本回退、保留配置、相邻 drop 等。
- vllm/parser/gemma4.py（模块 Gemma4 解析器；类别 source；类型 core-logic；符号 adjust_request）：移除了手动的 _GEMMA4_MODEL_DROP_TOKENS 和条件性的 adjust_request 覆盖，改为使用 preserve_tokens 保留 <|"|>，大幅简化 Gemma4 解析器配置。
- tests/parser/engine/test_replay.py（模块 重放测试；类别 test；类型 test-coverage；符号 _suppressed_expectations, _tool_suppression_expectations, TestSkipToolParsingReplay, TestToolCallFilteringReplay）：重构了测试夹具和期望计算，支持新的 tool suppression 语义，添加了 TestToolCallFilteringNonStreaming 等测试类，覆盖非流式路径。
- tests/parser/engine/replay_harness.py（模块 重放工具；类别 test；类型 test-coverage；符号 all_special_tokens, all_special_ids, parse_non_streaming）：增加了 MockTokenizer 的 all_special_tokens 和 all_special_ids 属性支持，新增 parse_non_streaming 辅助函数，为测试提供基础设施。
- vllm/parser/engine/token_id_scanner.py（模块 扫描器；类别 source；类型 core-logic）：移除了对 drop_token_ids 参数的依赖，因为 drop 逻辑已转移至 streaming_parser_engine 的自动发现机制。
- vllm/parser/engine/parser_engine.py（模块 解析引擎；类别 source；类型 core-logic）：调整了对 StreamingParserEngine 的交互，移除对旧 drop 机制的引用，适配新的 _suppress_tool_calls 标志。
- tests/parser/engine/test_gemma4_streaming_reasoning.py（模块 Gemma4 测试；类别 test；类型 test-coverage；符号 TestCommaInStringValueRegression, comma_tokenizer, comma_parser, multi_comma_tokenizer）：增加了 TestCommaInStringValueRegression 回归测试，确保 <|"|> 不会被自动 drop 导致逗号分割错误；同时更新 fixture 以支持 all_special_tokens。
- vllm/parser/engine/parser_engine_config.py（模块 配置；类别 source；类型 core-logic）：移除了 STRUCTURAL_DROP_TOKENS 常量，新增 preserve_tokens 配置项，移除 drop_tokens 参数。
- tests/parser/engine/test_token_id_scanner.py（模块 扫描器测试；类别 test；类型 test-coverage；符号 TestDropTokens, test_drop_token_with_holdback）：移除对 drop 机制的测试（因为 drop 逻辑已转移），删除 TestDropTokens 类。
- tests/parser/engine/test_delegating_replay.py（模块 委托测试；类别 test；类型 test-coverage；符号 test_delegating_parse_tool_choice_none）：新增 test_delegating_parse_tool_choice_none 测试，覆盖委托解析器的 tool_choice=none 场景。

- tests/tool_use/test_gemma4_responses_adjust_request.py（模块 Gemma4 请求测试；类别 test；类型 test-coverage；符号 all_special_tokens, all_special_ids, test_gemma4_strips_special_tokens_when_nothing_to_preserve, test_gemma4_keeps_skip_special_tokens_false_when_nothing_to_preserve）：更新测试以适配新的 adjust_request 行为（不再设置 skip_special_tokens），验证自动 drop 生效。

关键符号：_build_drop_info, _DropInfo, adjust_request (gemma4), _suppress_tool_calls, parse_non_streaming

关键源码片段

vllm/parser/engine/streaming_parser_engine.py

核心变更文件，引入自动 drop 令牌的 _DropInfo 和 _build_drop_info，重构 __init__集成新机制，移除了旧的 drop_token_ids 逻辑。

```
# streaming_parser_engine.py — 自动构建 drop 终端

@dataclass(slots=True)
class _DropInfo:
    lexer_shape: LexerShape # 包含原始 + drop 终端的完整 LexerShape
    extra_token_ids: dict[int, str] # 映射 token ID -> "__DROP__"


def _build_drop_info(
    config: ParserEngineConfig,
    tokenizer,
) -> _DropInfo | None:
    """从 tokenizer 的特殊 token 中自动发现需要 drop 的 token。"""
    try:
        special_tokens: list[str] = list(tokenizer.all_special_tokens)
        special_ids: list[int] = list(tokenizer.all_special_ids)
    except (AttributeError, NotImplementedError):
        return None # 不支持则回退

    if not special_tokens:
        return None

    # 已有配置的终端文本（不应被 drop）
    configured_texts = (
        set(config.token_id_terminals.values())
        | set(config.terminals.values())
        | config.preserve_tokens # 通过 preserve_tokens 显式保留
    )

    extra_token_ids: dict[int, str] = {}
    drop_texts: set[str] = set()
    for text, tid in zip(special_tokens, special_ids):
```

```
            if text not in configured_texts:
                extra_token_ids[tid] = DROP_TERMINAL # 标记为 __DROP__
                drop_texts.add(text)

        if not drop_texts:
            return None

        # 构造每个 drop 文本的 TerminalDef（字面匹配）
        drop_terminal_defs = [
            TerminalDef(
                name=DROP_TERMINAL,
                pattern=re.compile(re.escape(text)),
                is_literal=True,
                literal=text,
            )
            for text in drop_texts
        ]

        # 合并到完整终端定义中，生成新的 LexerShape
        all_terminal_defs = list(config.terminal_defs) + drop_terminal_defs
        lexer_shape = LexerShape(all_terminal_defs)

        return _DropInfo(
            lexer_shape=lexer_shape,
            extra_token_ids=extra_token_ids,
        )

    # __init__ 中的集成

            ...
            drop_info: _DropInfo | None = None
            if tokenizer is not None:
                drop_info = _build_drop_info(config, tokenizer)

            lexer_shape = config.lexer_shape
            if drop_info is not None:
                resolved_token_ids.update(drop_info.extra_token_ids) # 注入 __DROP__ 终端
                lexer_shape = drop_info.lexer_shape # 使用扩展后的 LexerShape

            self._resolved_token_ids = resolved_token_ids
            self._has_drops = drop_info is not None
            self._scanner = TokenIDScanner(resolved_token_ids, tokenizer) # 移除了旧的 drop_token_
            ids 参数
```

tests/parser/engine/test_parser_engine.py

新增 TestDropSpecialTokens 类，全面测试自动 drop 机制在各种场景下的正确性，包括 token ID 路径、文本回退、保留配置、相邻 drop 等。

```
    # test_parser_engine.py — TestDropSpecialTokens 类（部分）
```

```
class TestDropSpecialTokens:
    """特殊 token 自动 drop 的单元测试。"""

    def test_drops_special_token_by_id_from_content(self):
        """特殊 token 以其真实 token ID 到达时，应从 content 中移除。"""
        config = ParserEngineConfig(
            name="drop_content_test",
            terminals={},
            token_id_terminals={},
            transitions={},
            initial_state=ParserState.CONTENT,
            content_events={ParserState.CONTENT: EventType.TEXT_CHUNK},
        )
        engine = _make_engine(
            config=config,
            vocab=_DROP_VOCAB, # 包含 <bos>, <eos>
            special_tokens=list(_DROP_VOCAB.keys()),
        )
        engine._engine.reset()
        # 同时提供文本和 token ID，模拟流式输入
        events = engine._engine.feed("hello<bos>world", [72, 204, 73])
        delta = engine._events_to_delta(events)
        assert delta is not None
        assert "<bos>" not in delta.content # <bos> 应被移除
        assert delta.content == "helloworld" # 剩余内容拼接

    def test_drops_via_text_fallback_when_no_token_ids(self):
        """即使没有 token ID，文本 lexer 也能作为回退捕获 drop token。"""
        engine = _make_engine(
            vocab=_DROP_VOCAB,
            special_tokens=list(_DROP_VOCAB.keys()),
        )
        engine._engine.reset()
        events = engine._engine.feed("hello<bos>world", []) # 无 token ID
        delta = engine._events_to_delta(events)
        assert delta is not None
        assert "<bos>" not in delta.content

    def test_configured_terminal_not_treated_as_drop(self):
        """已配置为终端的特殊 token 不应被自动 drop。"""
        # 配置 TOOL_START 使用 <|tool_call|>
        config = _hermes_config()
        # <|tool_call|> 同时出现在 vocab 和 config.terminals 中
        engine = _make_engine(config=config, vocab=_DROP_VOCAB,
                              special_tokens=list(_DROP_VOCAB.keys()))
        engine._engine.reset()
        events = engine._engine.feed("<|tool_call|>content", [202, 65])
        delta = engine._events_to_delta(events)
        assert delta is not None
```

```
        assert delta.content is not None
        # 工具调用终端应被正常解析，不会变成 __DROP__
```

vllm/parser/gemma4.py

移除了手动的 _GEMMA4_MODEL_DROP_TOKENS 和条件性的 adjust_request 覆盖，改为使用 preserve_tokens 保留 <|"|>，大幅简化 Gemma4 解析器配置。

```
# gemma4.py — 移除手动 drop 配置后的 gemma4_config

@functools.cache
def gemma4_config() -> ParserEngineConfig:
    return ParserEngineConfig(
        name="gemma4",
        initial_state=ParserState.CONTENT,
        # ... 其他配置项保持不变 ...
        # drop_tokens 参数已移除，交由自动发现处理
        preserve_tokens=frozenset({STRING_DELIM}), # 保留 <|"|> 避免被自动 drop
    )
```

之前 gemma4.py 还包含 adjust_request 方法，根据 enable_thinking 和 tools 状态有条件地设置 skip_special_tokens=True。该逻辑已移除，因为现在解析器引擎自动处理所有特殊 token 的过滤和保留。

评论区精华

claude[bot] 在审核中指出：“解析器引擎的架构更改（通过 tokenizer 自动发现 drop，新的工具调用抑制路径，移除 per-model drop_token 列表）——这需要具有 parser-engine 上下文的人类审核者来签收。” SFeng33 随后批准了 PR。此外，PR 作者在升级主线时发现 Kimi K2 解析器的工具调用章节结构不同，调整了测试期望，说明跨解析器验证的重要性。

- 需要人类审核解析器引擎架构变更 (design): PR 后来被 sfeng33 批准，但架构变更的风险被记录。

风险与影响

- 风险：
  1. 自动发现依赖 tokenizer.all_special_tokens 属性，若某些 tokenizer 不暴露此属性（_build_drop_info 已捕获 AttributeError/NotImplementedError 并返回 None，回退到旧行为）。
  2. 新的抑制工具调用机制改变了 tool_choice='none' 的行为，可能影响依赖旧行为的客户端（如 Gemma4 之前通过 adjust_request 设置 skip_special_tokens=True 走文本路径，现在完全依赖解析器引擎）。
  3. 非流式路径的空白刷新可能改变某些场景的响应尾部空白（但作者认为外部服务层不可见）。
  4. 这些变更在 Gemma4、Kimi K2 等解析器上通过大量单元测试和 e2e 测试验证。 - 影响：对用户：修复了工具调用特殊 token 泄漏的关键 bug，提升了响应质量。对系统：移除 per-model 配置，降低了新模型集成解析器的维护成本。对团队：需要确保所有使用

parser engine 的模型在升级后测试通过（尤其是 tool_choice 和推理模式组合）。影响范围：所有启用工具调用的模型（Gemma4、Kimi、Hermes 等）在非流式路径都会受益。影响程度中等到高。 - 风险标记：核心路径变更，tokenizer 兼容性，需跨模型验证

关联脉络

- 暂无明显关联 PR