# PR #34237 完整报告

sgl-project/sglang

[Fix] lfm2 detector: recover tool calls dropped by common model-outpu···

合并时间：2026-08-23 18:09

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34237

## 执行摘要

- 一句话：修复 LFM2 检测器丢弃合法工具调用的问题
- 推荐动作：值得精读。核心看点： 1) 恢复阶梯的设计哲学——所有重写都是确定性的、保值的，并以 ast.parse 成功为唯一接受标准，“无法唯一判定就拒绝”的 fail-closed 原则避免了猜测性修复引入更坏的静默数据损坏； 2) “调用数不变量”防止晚闭合引号把兄弟调用吞进参数值，是本次最有洞察力的防护； 3) 19 个回归用例逐一验证修复前必失败，加上与 vLLM 的 42-payload 差分对齐，测试方法本身很有参考价值。只关心功能的读者可直接看 PR body 的三张 Before/After 表格，它本身就是一份优秀的 bug 分类清单。

## 功能与动机

PR body 明确说明动机来自真实负载：“Running real LFM2/LFM2.5 agent traces (SWE-agent / shell-command workloads) through Lfm2Detector turned up three distinct kinds of problem.” 三类问题按“是否解析器的错”分层：

1) 合法 Python 被误拒（limit=+7 只有 USub 分支、set 字面量、无占位符 f-string 都导致调用被丢弃）； 2) 发出错误数据——1e999 序列化成 Infinity 非法 JSON、位置参数 'Paris' 静默丢失、**kwargs 静默缺失、bytes 参数连坐整块调用、单个坏引号让整块 SyntaxError，“比不报告更糟，工具会在错误输入上执行”； 3) 模型确实写了非法 Python（裸换行、NUL 字节、前导零整数、嵌套引号）时解析器直接放弃。作者同时说明这是 vLLM 同格式修复的移植：“These mirror the fixes recently merged for the same tool-call format in vLLM (vllm-project/vllm#48171); this port is adapted to Lfm2Detector's structure (per-call salvage, buffered streaming) and touches no other detector.”

## 实现拆解

1. 变更入口与恢复阶梯：python/sglang/srt/function_call/lfm2_detector.py 的 _parse_pythonic_content 在 safe_ast_parse 失败后进入恢复循环，依次尝试 _recovery_candidates 组织的一组模块级纯函数重写，每个候选重新解析、任一成功即采用。这些 helper 包括：_escape_ctrl_chars_in_strings（把字符串内的裸换行 / 回车 / 制表符 /NUL 转义为 \n/\r/\t/\x00，同时规避 SyntaxError 与 ValueError 且保值）、_normalize_leading_zero_ints（剥离十进制整数字面量前导零，放过 0x/ 浮点 / 指数 / 全零等合法形态）、_rename_reserved_kwargs/_restore_reserved_kwarg_names（仅在关键字参数位置把 from=1 改成 from_pyreservedkw_=1，解析后精确还原，且不误伤 == 比较）、_escape_nested_quotes_in_strings（对 broken 字符串枚举所有候选闭合引号、转

义内部同风格引号并逐个 ast.parse 校验，恰好一个候选才恢复）。

2. 参数提取与单调用语义加固：_get_parameter_value 新增 UAdd（显式正号，且拒绝 -True）、Set（按源码顺序转 list）、全 Constant 的 JoinedStr（无占位符 f-string 折叠为字符串），并把 Constant 限定为 None/str/int/float，让 bytes/Ellipsis/ 复数在提取期就被拒绝。_parse_pythonic_call 改用 json.dumps(..., allow_nan=False)，序列化失败只跳过当前调用；** 解包的字面量 dict 按 later-binding-wins 合并，非 dict 操作数拒绝调用；含位置参数的调用直接拒绝，不再静默丢失位置参数值后照常交付。
3. 块级抢救与流式继承：当整块候选全部无法解析时，用 _split_top_level_calls 按顶层逗号把块切成独立段，每段单独走同一恢复阶梯，保证 [read(path='/x'), bash(command='grep 'e' a.log')] 里合法的 read 存活；切分同时跑字符串感知与纯括号计数两遍扫描，避免坏引号隐藏分隔符。流式路径 parse_streaming_increment 缓冲完整块后委托给 detect_and_parse，自动继承全部恢复逻辑，无需单独实现。
4. PythonicDetector 兄弟修复：python/sglang/srt/function_call/pythonic_detector.py 移植其中两个可观测缺陷——按调用粒度捕获 ValueError/TypeError（不再连坐整块），并 allow_nan=False 序列化；其正则门槛本来就拒绝位置参数与 ** 解包，故这两个类无需移植。
5. 测试配套：test/registered/unit/function_call/test_function_call_parser.py 新增 19 个回归用例（17 个在 TestLfm2Detector，其余覆盖 PythonicDetector 的同类缺陷），每个都验证了“修复前必失败”，覆盖恢复正确分支与歧义拒绝的负向分支（如 test_ambiguous_nested_quotes_not_guessed、test_swallowing_reading_rejected）。另用 42 个 payload（vLLM 测试常量、对抗性引号 / 转义用例、真实 SWE-agent 轨迹）做差分测试，确认与 vLLM 合并版解析器输出完全一致。

关键文件：

- python/sglang/srt/function_call/lfm2_detector.py（模块 调用检测；类别 source；类型 core-logic；符号 _rename_reserved_kwargs, _restore_reserved_kwarg_names, _is_escaped, _escape_nested_quotes_in_strings）：本 PR 的核心修复文件：新增 4 组模块级恢复重写 helper、恢复阶梯 _recovery_candidates 与 _parse_pythonic_content 中的重试循环，并强化 _get_parameter_value 与 _parse_pythonic_call。所有恢复逻辑只在 ast.parse 失败后触发，正常路径零开销。
- test/registered/unit/function_call/test_function_call_parser.py（模块 解析测试；类别 test；类型 test-coverage；符号 test_non_finite_argument_never_emits_invalid_json, test_unconvertible_argument_skips_only_that_call, test_multiline_string_argument_recovered, test_nul_byte_in_string_argument_recovered）：19 个回归用例全部覆盖“修复前必失败”的场景，其中 test_swallowing_reading_rejected 锁定了“晚闭合读法吞掉兄弟调用”这一最关键不变量，test_ambiguous_nested_quotes_not_guessed 锁定歧义拒绝的负向分支；PR body 声明这些用例都先在旧实现上验证失败。
- python/sglang/srt/function_call/pythonic_detector.py（模块 调用检测；类别 source；类型 core-logic；符号 detect_and_parse, _get_parameter_value）：移植 LFM2 修复中在 PythonicDetector 可观测的两个缺陷：非有限浮点序列化为 Infinity、不可转换参数连坐整块调用；改为按调用粒度捕获并跳过，_get_parameter_value 同时限定 Constant 为

JSON 可表示类型。

关键符号：_rename_reserved_kwargs, _restore_reserved_kwarg_names, _is_escaped, _escape_nested_quotes_in_strings, _escape_ctrl_chars_in_strings, _normalize_leading_zero_ints, _recovery_candidates, _split_top_level_calls, _parse_pythonic_call, _parse_pythonic_content, _get_parameter_value

## 关键源码片段

### python/sglang/srt/function_call/lfm2_detector.py

本 PR 的核心修复文件：新增 4 组模块级恢复重写 helper、恢复阶梯 _recovery_candidates 与 _parse_pythonic_content 中的重试循环，并强化 _get_parameter_value 与 _parse_pythonic_call。所有恢复逻辑只在 ast.parse 失败后触发，正常路径零开销。

```
def _escape_nested_quotes_in_strings(text: str) -> Tuple[str, bool]:
    """恢复被嵌套引号破坏的字符串参数（仅在歧义唯一时）。

    模型输出的 shell 命令常把同风格引号嵌套进字符串参数，例如
    ``command='sed -n '360,450p' f.py'``，Python 会把 `'sed -n '` 与
    数字 `360` 视作并列表达式（juxtaposition）报 `SyntaxError`，导致
    整块调用被丢弃。这里把"第一个未转义引号无法在语法上闭合该字符串"
    视为 broken，再枚举每个语法上可行的闭合引号，转义中间的同风格
    引号并做 `ast.parse` 校验；恰好一个候选可解析才恢复，零个或多个
    候选视为真正歧义，原样返回、绝不猜测。
    """

    def unescaped_quotes(start: int, quote: str) -> List[int]:
        # 收集从 start 起的未转义同风格引号位置，遇到反斜杠成对跳过
        positions = []
        j = start
        while j < len(text):
            if text[j] == "\\":
                j += 2
                continue
            if text[j] == quote:
                positions.append(j)
            j += 1
        return positions

    def is_closer(pos: int) -> bool:
        # 引号后（允许空白）紧跟 ,)]}: 之一才是语法上可行的闭合点；
        # 否则说明字符串在中间被打断，处于 broken 状态
        k = pos + 1
        while k < len(text) and text[k].isspace():
            k += 1
        return k < len(text) and text[k] in _QUOTE_FOLLOWERS

    # 关键不变量：晚闭合的候选可能把兄弟调用吞进字符串值里，例如
    # ``f(a='x 'y'), g(...)`` 会被读成单个调用、`g(...)` 被塞进 `a`，
```

```
    # 工具将拿到被污染的参数执行——比丢弃更糟。括号计数不受坏引号
    # 影响，因此用顶层调用数作为候选的合法性门槛。
    expected_calls = len(_split_top_level_calls(text, respect_strings=False))
    prefix: List[str] = []
    index = 0
    while index < len(text):
        char = text[index]
        if char not in {"'", '"'}:
            prefix.append(char)
            index += 1
            continue
        quotes = unescaped_quotes(index + 1, char)
        if not quotes:
            return text, False
        if is_closer(quotes[0]):
            # 第一个引号就能正常闭合，按原样消费这一段
            prefix.append(text[index : quotes[0] + 1])
            index = quotes[0] + 1
            continue
        winners = []
        for close in (j for j in quotes if is_closer(j)):
            interior: List[str] = []
            for j in range(index + 1, close):
                if text[j] == char and not _is_escaped(text, j):
                    interior.append("\\")
                interior.append(text[j])
            candidate = "".join(
                ["".join(prefix), char, "".join(interior), char, text[close + 1 :]]
            )
            try:
                module = safe_ast_parse(_escape_ctrl_chars_in_strings(candidate))
            except (SyntaxError, ValueError):
                continue
            if expected_calls > 1 and _top_level_call_count(module) < expected_calls:
                continue
            winners.append(candidate)
        if len(winners) == 1:
            return winners[0], True
        return text, False
    return text, False
```

## python/sglang/srt/function_call/pythonic_detector.py

移植 LFM2 修复中在 PythonicDetector 可观测的两个缺陷：非有限浮点序列化为 Infinity、不可转换参数连坐整块调用；改为按调用粒度捕获并跳过，`_get_parameter_value` 同时限定 Constant 为 JSON 可表示类型。

```
        for call_index, call in enumerate(parsed.elts):
            if not isinstance(call.func, ast.Name):
                continue
```

```
            function_name = call.func.id
            # 工具未注册时按 SGLANG_FORWARD_UNKNOWN_TOOLS 决定跳过或放行
            if function_name not in tool_indices:
                logger.warning(
                    f"Model attempted to call undefined function: {function_name}"
                )
                if not envs.SGLANG_FORWARD_UNKNOWN_TOOLS.get():
                    continue
            # 每个 Call 独立转换：不可转换的参数以前会逃逸到外层 except，
            # 把块内所有可解析的兄弟调用一并丢弃
            try:
                arguments = {}
                for keyword in call.keywords:
                    arguments[keyword.arg] = self._get_parameter_value(keyword.value)
                # allow_nan=False：非有限浮点（如字面量 1e999 溢出为 inf）
                # 以前会被序列化成 Infinity，下游 JSON 解析器无法接受
                parameters = json.dumps(arguments, ensure_ascii=False, allow_nan=False)
            except (ValueError, TypeError) as e:
                logger.warning(f"Skipping tool call {function_name}: {e}")
                continue
            calls.append(
                ToolCallItem(
                    tool_index=call_index,
                    name=function_name,
                    parameters=parameters,
                )
            )
        return StreamingParseResult(normal_text=normal_text, calls=calls)
```

## 评论区精华

该 PR 没有行内 review 评论，唯一审核是 APPROVED（JustinTong0323：“LGTM”）；3 条 issue 评论均为 CI 指令（/tag-and-rerun-ci、两次 /rerun-failed-ci）。真正的设计讨论记录在 PR body 与提交演进中：

1) 作者把第三类“模型写了非法 Python 时的恢复重写”明确标记为 “the debatable group; I am happy to drop them if you would rather keep the pressure on the model”，即恢复可能降低对模型输出质量的纠正压力，最终该组随 PR 合并，接受标准是“确定性、保值、ast 校验通过、歧义即拒绝”； 2) 合并者 JustinTong0323 用最后一个提交（753024d）补上了一处审查暴露的作用域缺陷：_restore_reserved_kwarg_names 原本无条件执行，工具真实声明 in_pyreservedkw_ 参数会被静默改回 in，修复后 _recovery_candidates 对经过改名处理的候选打标记、还原只作用于这些候选，并把 rename 与 requote 组合以覆盖 from='sed -n '1,5p' f.py' 这类叠加缺陷。

- 恢复型重写与严格拒绝的边界取舍 (design): 该组恢复最终随 PR 合并；接受标准是确定性、保值、ast 校验通过，且多个候选都能解析的歧义场景拒绝而非猜测。
- 保留字参数恢复的作用域缺陷修复 (correctness): _recovery_candidates 为经过 rename 的候选打标记，还原仅作用于这些候选的解码结果；rename 与 requote 按序组合执行。

# 风险与影响

- 风险： 1) lfm2_detector.py 的 4 个重写 helper 都是字符级状态机扫描（引号、反斜杠转义、标识符 / 数字边界），对三重引号字符串、f-string 内部花括号、注释等未覆盖语法存在误判可能；缓解措施是所有候选必须通过 safe_ast_parse 校验，且只在原解析失败后触发，正常路径零开销。 2) _escape_nested_quotes_in_strings 依赖 _split_top_level_calls 的括号计数作为“不吞兄弟调用”的不变量，若块内出现未配对括号，计数可能失真，恢复判定会变保守或失效。 3) pythonic_detector.py 行为变更：bytes/ 复数等参数从“整块丢弃”变为“仅跳过该调用”，非有限浮点从“输出非法 JSON”变为“跳过调用”，对既有调用方是可见但合理的语义收紧。 4) 测试覆盖：19 个回归用例大多走非流式 detect_and_parse，流式缓冲路径虽声明继承同一逻辑，但缺少针对恢复的流式专项用例。 5) 性能：恢复阶梯只在解析失败后执行，最坏是对同一文本做多次线性扫描，O(n) 量级，风险低。
  - 影响：影响范围集中在 sglang/srt/function_call 包：Lfm2Detector 的合法调用不再被误丢、非法参数不再以“成功”状态输出、坏块中的好调用得以存活；PythonicDetector 也同步收紧了两处缺陷。对用户而言，LFM2/LFM2.5 的 SWE-agent / shell-command 场景工具调用成功率与参数可信度提升；对系统无 API、schema 或部署变更；对团队而言，这套“确定性、保值、ast 校验、歧义即拒绝”的恢复模式与 42-payload 差分对比 vLLM 的验证方法，可作为后续其他 pythonic 系检测器（llama4/olmo3）同类修复的模板。
  - 风险标记：引号状态机复杂度高，嵌套引号恢复依赖调用数不变量，Pythonic 检测器行为变更，流式恢复路径测试覆盖偏弱

# 关联脉络

- 暂无明显关联 PR