

PR #51450 完整报告

vllm-project/vllm

[Structured Output] Keep invalid structured-output requests from stopping the engine

合并时间: 2026-09-09 23:32

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/51450>

执行摘要

- 一句话: 修复投机解码 -1 填充 token 导致结构化输出引擎崩溃的关键安全问题。
- 推荐动作: 这是一个重要的安全修复, 值得精读。核心设计决策——将填充剥离逻辑从单一后端移至调度器共享路径——是正确且具有远见的, 它确保了修复的全面性和持久性。评审过程 (@arpera 的深入分析和测试要求) 显著提升了修复的质量和覆盖范围。建议关注以下方面: `strip_speculative_padding` 的实现细节及其在调度器两个不同入口点的应用方式; 新增测试如何模拟崩溃场景 (通过 `_RecordingGrammar` 和 `_Matcher` mock); 以及修复如何同时解决了功能性问题 (崩溃) 和用户体验问题 (`xgrammar` 的嘈杂警告)。

功能与动机

本 PR 解决了一个影响服务可用性的安全问题。根据 PR body, `-1` 值仅用于填充投机解码 token 行, 但被传递给了结构化输出语法器, 后者将每个条目视为真实 token ID。这可能导致受影响的后端崩溃。问题具有普遍性, 因为任何允许请求结构化输出的客户端都可以通过普通请求触发此路径, 而无需特殊访问权限。影响是服务中断, 而非代码执行或数据访问。

实现拆解

1. 新增通用填充移除函数: 在 `vllm/v1/structured_output/utils.py` 中新增 `strip_speculative_padding` 函数。该函数遍历 token ID 列表, 遇到第一个负数 (`-1` 哨兵) 即截断并返回其前缀。此设计将修复逻辑集中于单一、可复用的位置。
2. 在调度器关键路径集成移除逻辑: 在 `vllm/v1/core/sched/scheduler.py` 的 `update_draft_token_ids` 和 `update_draft_token_ids_in_output` 方法中, 在调用 `metadata.grammar.validate_tokens(spec_token_ids)` 之前, 先调用 `strip_speculative_padding(spec_token_ids)`。这确保了所有后端在验证前都收到干净的 token 列表。
3. 在 Guidance 后端添加防御性双重保障: 在 `vllm/v1/structured_output/backend_guidance.py` 的 `GuidanceGrammar.validate_tokens` 方法中, 在调用底层 `ll_matcher.validate_tokens` 之前, 也调用 `strip_speculative_padding`。这是为了防止存在未通过调度器路径的调用, 并作为一个独立的防御层。同时添加了空列表检查, 确保移除填充后若为空则直接返回。
4. 全面补充测试用例: 新增并修改了三个测试文件。`tests/v1/structured_output/test_scheduler_speculative_padding.py` 测试新工具函数的各种边界情况。`tests/v1/core/test_scheduler.py` 中添加了两个测试, 验证调度器在

update_draft_token_ids 和 update_draft_token_ids_in_output 路径上正确剥离 -1 填充，并验证语法器从未接收到负数。tests/v1/structured_output/test_guidance_negative_draft_tokens.py 专门测试 Guidance 后端在收到 -1 填充时不再引发异常。

关键文件：

- vllm/v1/structured_output/utils.py（模块 结构化输出；类别 source；类型 core-logic；符号 strip_speculative_padding）：包含核心修复逻辑 strip_speculative_padding 函数，是本次变更的基础工具。
- vllm/v1/core/sched/scheduler.py（模块 调度器；类别 source；类型 dependency-wiring；符号 update_draft_token_ids, update_draft_token_ids_in_output）：调度器是草稿 token 处理的入口，在此集成修复确保了所有后端的通用性。
- vllm/v1/structured_output/backend_guidance.py（模块 结构化输出；类别 source；类型 dependency-wiring；符号 GuidanceGrammar.validate_tokens）：为 Guidance 后端添加防御性保护，作为调度器修复的补充，确保即使存在直接调用也是安全的。
- tests/v1/core/test_scheduler.py（模块 调度器；类别 test；类型 test-coverage；符号 _RecordingGrammar, test_update_draft_token_ids_strips_ngram_padding, test_update_draft_token_ids_in_output_strips_padding）：验证调度器核心路径的修复是否有效，使用 mock 语法器模拟崩溃场景。
- tests/v1/structured_output/test_guidance_negative_draft_tokens.py（模块 结构化输出；类别 test；类型 test-coverage；符号 test_validate_tokens_stops_at_trailing_padding, test_validate_tokens_single_negative_does_not_raise）：专门测试 Guidance 后端对 -1 填充的防御性保护。
- tests/v1/structured_output/test_scheduler_speculative_padding.py（模块 结构化输出；类别 test；类型 test-coverage；符号 test_strips_trailing_padding, test_empty_when_leading_padding, test_passthrough_without_padding, test_truncates_at_first_sentinel）：单元测试新提取的 strip_speculative_padding 工具函数。

关键符号：strip_speculative_padding, Scheduler.update_draft_token_ids, Scheduler.update_draft_token_ids_in_output, GuidanceGrammar.validate_tokens

关键源码片段

vllm/v1/core/sched/scheduler.py

调度器是草稿 token 处理的入口，在此集成修复确保了所有后端的通用性。

```
def update_draft_token_ids(self, draft_token_ids: DraftTokenIds) -> None:
    # ... 省略部分代码 ...
    if self.structured_output_manager.should_advance(request):
        metadata = request.structured_output_request
        # 在 validate_tokens 前剥离投机解码的 -1 填充，防止语法器崩溃或警告
        spec_token_ids = strip_speculative_padding(spec_token_ids)
        spec_token_ids = metadata.grammar.validate_tokens(spec_token_ids)
        request.spec_token_ids = spec_token_ids

def update_draft_token_ids_in_output(
```

```

self, draft_token_ids: DraftTokenIds, scheduler_output: SchedulerOutput
) -> None:
    # ... 省略部分代码 ...
    if self.structured_output_manager.should_advance(request):
        metadata = request.structured_output_request
        # 同样在 validate_tokens 前剥离填充, 保持路径一致
        spec_token_ids = strip_speculative_padding(spec_token_ids)
        spec_token_ids = metadata.grammar.validate_tokens(spec_token_ids)
    # ... 省略部分代码, 包括后续的 -1 回填逻辑 ...

```

vllm/v1/structured_output/backend_guidance.py

为 Guidance 后端添加防御性保护, 作为调度器修复的补充, 确保即使存在直接调用也是安全的。

```

def validate_tokens(self, tokens: list[int]) -> list[int]:
    # ... 省略部分代码 ...
    if self.ll_matcher.is_stopped():
        return []

    # 防御性剥离投机解码填充, 即使调度器已处理也保证此处安全
    tokens = strip_speculative_padding(tokens)
    if len(tokens) == 0:
        return []

    num_tokens = self.ll_matcher.validate_tokens(tokens)

    self.check_error()

    return tokens[:num_tokens]

```

评论区精华

核心讨论由维护者 @arpera 发起, 他指出了初始实现中的几个关键问题并推动了重构:

1. 修复位置不当: 初始修复位于 `update_from_output` 方法, 但 @arpera 通过实际复现测试 (`e_ngram_gpu_repro.py`) 确认, 实际导致崩溃的代码路径是 `update_draft_token_ids` 和 `update_draft_token_ids_in_output` 中对 `grammar.validate_tokens` 的调用。他要求将修复移至这两处。
2. 修复应后端无关: @arpera 指出, 修复不应仅限于 Guidance 后端, 而应在调度器层面实现, 以确保对任何未来后端都有效。他提到 `xgrammar` 后端虽然能处理 `-1`, 但会输出大量嘈杂的警告, 调度器层面的修复可以避免此问题。
3. 测试覆盖不足: @arpera 要求补充能在修复前复现错误、修复后通过的测试, 并具体建议测试 `GuidanceGrammar.validate_tokens([-1])` 不会抛出异常。
4. 消除冗余代码: 评审指出 Guidance 后端内原有的循环逻辑与新提取的 `strip_speculative_padding` 函数重复, 要求改为复用该共享工具函数。

最终, PR 根据这些反馈进行了重大调整, 将核心修复逻辑从单一后端移至调度器的共享路径, 并添加了全面的测试套件。

- 修复位置与范围 (design): 将 `strip_speculative_padding` 调用集成到调度器的两个方法中, 并在 Guidance 后端添加防御性调用, 形成双层保护。
- 测试覆盖要求 (testing): 作者新增了三个测试文件: 测试工具函数本身、测试 Guidance 后端、以及测试调度器两个方法路径, 全面覆盖了修复前后行为。
- 消除代码重复 (style): 作者重构代码, 将 Guidance 后端中的循环替换为对共享工具函数 `strip_speculative_padding` 的调用。
- 移除冗余注释 (style): 作者移除了两处注释, 保持代码简洁。

风险与影响

- 风险:
 1. 回归风险: 修改了调度器处理投机解码草稿 token 的核心路径 (`update_draft_token_ids` 和 `update_draft_token_ids_in_output`)。虽然剥离填充后逻辑不变, 但增加了函数调用, 需确保不影响正常流程的性能或正确性。测试覆盖了主要场景, 但未涉及与不同投机解码器的集成测试。
 2. 兼容性风险: 修复依赖于 `-1` 作为投机解码填充的哨兵值。如果未来有新的投机解码器使用其他负值或不同机制, 此修复可能不完整。不过, `strip_speculative_padding` 的逻辑 (遇到任何负数即截断) 具有较强的适应性。
 3. 安全修复完整性: 作为安全修复, 其主要风险在于是否覆盖了所有可能的攻击路径。当前修复在调度器层面拦截, 理论上覆盖了所有通过调度器处理的草稿 token。但 `GuidanceGrammar.validate_tokens` 中的防御性添加, 表明团队考虑了可能存在绕过调度器的调用点, 这是一种纵深防御策略。
- 影响:
 1. 用户影响: 直接影响所有使用结构化输出 (JSON Schema、正则表达式等) 并启用投机解码的用户。修复前, 恶意或意外的请求可能导致服务崩溃 (拒绝服务)。修复后, 这些请求将被安全处理, 服务稳定性提高。
 2. 系统影响: 修复位于调度器这一关键组件, 影响所有使用结构化输出的请求的草稿 token 验证流程。由于修复点通用, 未来新增的结构化输出后端也将自动受益。
 3. 团队影响: 此 PR 展示了安全漏洞修复的良好实践: 识别根本原因 (投机解码填充与语法验证的不兼容)、将修复提升到架构层面 (调度器)、并添加防御层。它为处理类似跨组件数据不一致问题提供了模板。 - 风险标记: 核心路径变更, 安全修复, 投机解码集成

关联脉络

- PR #54821 [Structured Output] Handle empty values from Rust frontend in structured output: 该 PR 是从本 PR (#51450) 中拆分出来的, 处理 Rust 前端在结构化输出中产生空值的问题 (PTP-VLLM-043), 是同一安全审计 (GHSA-85xf-c7hm-whqw) 的另一项修复。