PR #47707 完整报告

vllm-project/vllm

[Bugfix][Rust Frontend] Detokenizer: avoid leaking prompt on zero-generated-token completions

合并时间：2026-07-16 17:11

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/47707

执行摘要

本 PR 修复了 Rust 前端分词器（DecodeStream）中一个边界条件 bug：当模型输出的唯一 token 是 EOS 并被引擎抑制时，flush() 会将整个 prompt 文本泄漏到输出结果中。修复通过检查 prefix_seeded 标志，确保只有在实际推送过 token 后才执行解码，否则返回空文本，与 Python V1 前端行为一致。

功能与动机

当请求以零生成 token 结束时——例如模型首 token 即为 EOS，引擎将其抑制——flush() 在 push_token() 从未被调用的情况下被触发。此时 DecodeStream 的 ids 中仍持有全部 prompt token id（作为解码的左侧上下文），而 prefix 为空，因为 seed_prefix() 仅在 push_token() 中调用。原代码在 flush() 中无条件解码 ids，导致整个 prompt 被追加到 cumulative_output，最终返回给用户。这与 Python V1 路径中 BaseIncrementalDetokenizer.update() 在 new_token_ids 为空时直接返回的行为不一致。

实现拆解

1. 问题定位：分析 DecodeStream::flush() 原实现：if !self.ids.is_empty() 时直接解码，但 ids 在零 token 场景下只包含 prompt，prefix_len 为 0，导致 prompt 全文被输出。
2. 修复核心：将条件改为 if self.prefix_seeded && !self.ids.is_empty()。prefix_seeded 仅在 push_token() 中设为 true，因此无 token 推送时不会进入解码分支。同时将 ids、prefix、prefix_index 的清除操作移到条件块之外，确保状态始终重置。
3. 测试覆盖：新增两个回归测试：
   - flush_without_push_token_does_not_leak_prompt：7001 token 的 prompt，验证返回空文本。
   - flush_without_push_token_does_not_leak_undecodable_prompt_tail：不完整 UTF-8 序列 [0xe4, 0xbd]，验证即使 seed_prefix() 无法建立有效 prefix，也不会泄漏。

rust/src/tokenizer/src/incremental.rs

包含 DecodeStream::flush() 方法的核心修复和两个新增的回归测试。

```
fn flush(&mut self, truncate_output_to: Option<usize>) -> Result<(Option<String>, String)> {
    // 如果 prefix 从未被种子化（即从未调用 push_token），
    // 则 ids 中只包含 prompt context —— 解码它会重复输出 prompt 文本。
    // 仅当 prefix_seeded 且 ids 非空时才执行解码；
    // 否则跳过解码，直接清理状态，返回空文本。
```

```
    if self.prefix_seeded && !self.ids.is_empty() {
        let string = self.tokenizer.decode(&self.ids, self.skip_special_tokens)?;
        let prefix_len = self.prefix.len();
        // 确保在 UTF-8 字符边界处切割。
        self.cumulative_output
            .push_str(&string[string.floor_char_boundary(prefix_len)..]);
    }
    self.ids.clear();
    self.prefix.clear();
    self.prefix_index = 0;
    self.prefix_seeded = true;
    if let Some(truncate_output_to) = truncate_output_to {
        self.cumulative_output.truncate(truncate_output_to);
    }
    let last_chunk = (self.output_index < self.cumulative_output.len())
        .then(|| self.cumulative_output[self.output_index..].to_string());
    self.output_index = 0;
    Ok((last_chunk, take(&mut self.cumulative_output)))
}

// 回归测试：验证零生成 token 时 flush 不泄漏 prompt
#[test]
fn flush_without_push_token_does_not_leak_prompt() {
    let backend = Utf8Backend;
    let prompt: Vec<u32> = b"The quick brown fox jumps over the lazy dog. "
        .iter()
        .cycle()
        .take(7001)
        .map(|&b| b as u32)
        .collect();
    let mut decoder = backend.create_decode_stream(&prompt, false, 0);
    let (last_chunk, full_text) = decoder.flush(None).unwrap();
    assert_eq!(last_chunk, None);
    assert_eq!(full_text, "");
}

// 回归测试：即使 prompt 尾部是不可解码的不完整 UTF-8，也不泄漏
#[test]
fn flush_without_push_token_does_not_leak_undecodable_prompt_tail() {
    let backend = Utf8Backend;
    let prompt = vec![0xe4, 0xbd];
    let mut decoder = backend.create_decode_stream(&prompt, false, 0);
    let (last_chunk, full_text) = decoder.flush(None).unwrap();
    assert_eq!(last_chunk, None);
    assert_eq!(full_text, "");
}
```

评论区精华

> BugenZhao: “Do we really need to seed the prefix here anyway? Can we simply do decode here only when if self.prefix_seeded && !self.ids.is_empty()?”
>
> xiaguan: “Thanks for pointing this out — you're right. If prefix_seeded is false at flush time, push_token() was never called and ids only holds prompt context, so seeding a prefix that gets discarded right after is wasted work.”

这一简洁的设计建议将修复从「先种子化再跳过」简化为「直接检查标志」，避免了不必要的操作，体现了良好的代码审视。

风险与影响

- 风险：低。修复仅改变 flush() 中解码的条件，对正常（有生成 token）的流程无影响。prefix_seeded 在 push_token() 后为 true，逻辑不变。
- 影响：仅限于 Rust 前端分词器在零生成 token 请求下的行为，消除 prompt 泄漏。与 Python V1 前端行为保持一致性。

关联脉络

此 PR 是 Rust 前端分词器维护的一部分，近期无直接关联的 PR（无重复函数或模块变更）。它独立修复了一个边界条件，确保了 Rust 前端的鲁棒性。