PR #44552 完整报告

vllm-project/vllm

[Rust Frontend] Add seed_oss and step3p5 reasoning parsers

合并时间：2026-06-10 14:01

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/44552

执行摘要

本 PR 为 vLLM Rust 前端新增了 seed_oss 和 step3p5 两个推理解析器，对应 ByteDance-Seed/Seed-OSS-36B-Instruct 和 stepfun-ai/Step-3.5-Flash 模型。同时改进了共享的 DelimitedReasoningParser 的初始化逻辑，通过 initialize 方法根据 prompt 中的 token 确定初始状态，替代了运行时的标记跳过，使 DeepSeekR1 等现有解析器也受益。添加了模型名称自动检测模式和完整的测试覆盖（单元测试 + 工厂测试 + 端到端 roundtrip 测试）。

功能与动机

Rust 前端功能对等路线图 (issue #44280) 要求 Rust 前端具备与 Python 前端同等的推理解析能力，目前 DeepSeekR1、Qwen3 等已支持，但缺乏 Seed-OSS 和 Step-3.5 的支持。这两个模型分别使用 <seed:think>/</seed:think> 和 <think>/</think> 标签，后者还有在后与后换行框架的特殊处理。添加后，Rust 前端可以直接处理这些模型的推理输出，无需 fallback 到 Python 路径。

实现拆解

1. 新增解析器文件：创建 rust/src/reasoning-parser/src/seed_oss.rs 和 rust/src/reasoning-parser/src/step3p5.rs。 - SeedOssReasoningParser 直接委托 DelimitedReasoningParser，通过 initialize 检测 prompt 中是否包含 <seed:think> token 来设置初始状态。 - Step3p5ReasoningParser 在委托基础上增加了 process 方法，处理 </think> 前后的换行剥离：持有 reasoning 末尾的 \n 跨 push 缓存，在 </think> 到达时丢弃；同时丢弃退出推理后内容的首个前导 \n。
2. 扩展共享解析器：在 rust/src/reasoning-parser/src/delimited.rs 中添加 in_reasoning() 方法，返回当前是否处于推理模式，供 Step3p5 的 push 和 finish 判断状态转换。
3. 注册解析器与模式匹配：在 rust/src/chat/src/parser/reasoning/mod.rs 中：
   - 添加 SEED_OSS 和 STEP3P5 常量，注册对应解析器构造函数。
   - 添加模式：step-3p5、step3p5、step-3.5 映射到 STEP3P5（注意顺序，必须排在 step3 之前）；seed-oss、seedoss 映射到 SEED_OSS。
4. 测试覆盖：
   - 单元测试：每个解析器模块内覆盖无 prompt 标记、有 prompt 标记（起始 / 结束）、显式开始标签、流式分片、部分分隔符、未终止推理等场景。
   - 工厂测试：在 rust/src/chat/src/parser/reasoning/tests.rs 中验证 step-3p5 不被误路由到 step3，ByteDance-Seed/Seed-OSS-36B-Instruct 等模型名正确解析。

- 端到端测试：在 rust/src/chat/tests/roundtrip.rs 中增加 seed_oss 和 step3p5 两个 RoundtripCase，通过真实 HF tokenizer 和 chat template 验证推理 + 内容的分割输出。

rust/src/reasoning-parser/src/step3p5.rs

新增 Step3p5ReasoningParser，实现复杂的换行框架处理逻辑，是该 PR 的核心变更之一。

```
/// Step3p5 推理解析器核心处理逻辑：去除 `</think>` 前后的换行框架。
///
/// process 在每次 push 和 finish 时被调用，接收内部 DelimitedReasoningParser
/// 产生的 delta 以及转换前后的状态，负责处理换行缓存和丢弃。
fn process(
    &mut self,
    mut inner_delta: ReasoningDelta,
    was_in_reasoning: bool,
    now_in_reasoning: bool,
) -> ReasoningDelta {
    // 检测是否发生了推理→内容的转换（包括一次 push 内完成整个轮回的情况）
    let transitioned =
        !now_in_reasoning && (was_in_reasoning || inner_delta.reasoning.is_some());

    // 如果之前持有拖尾换行，现在需回放或丢弃
    if self.pending_reasoning_newline {
        if let Some(reasoning) = inner_delta.reasoning.as_mut() {
            // 有新的 reasoning 文本：将之前缓存的换行加回到文本前面
            reasoning.insert(0, '\n');
            self.pending_reasoning_newline = false;
        } else if transitioned {
            // 触发转换且无新 reasoning 文本：缓存的换行是 `</think>` 前的那个，丢弃
            self.pending_reasoning_newline = false;
        }
    }

    // 如果当前 reasoning 文本以换行结尾，则临时移除并缓存它，
    // 等待下一个 push 再决定是否保留（若之后是 `</think>` 则丢弃）
    if let Some(reasoning) = inner_delta.reasoning.as_mut()
        && reasoning.ends_with('\n')
    {
        reasoning.pop();
        if !transitioned {
            self.pending_reasoning_newline = true;
        }
    }

    // 如果紧跟在 `</think>` 之后的内容以换行开头，则去掉前导换行
    if let Some(content) = inner_delta.content.as_mut()
        && (transitioned || self.just_ended_reasoning)
        && content.starts_with('\n')
    {
        content.remove(0);
```

```
        }

        // 记录当前是否刚结束推理且未产生内容（以便下一个 push 处理前导换行）
        self.just_ended_reasoning = transitioned && inner_delta.content.is_none();

        // 移除空字符串，避免外部收到无意义 delta
        if inner_delta.reasoning.as_deref() == Some("") {
            inner_delta.reasoning = None;
        }
        if inner_delta.content.as_deref() == Some("") {
            inner_delta.content = None;
        }

        inner_delta
    }
```

rust/src/reasoning-parser/src/seed_oss.rs

新增 SeedOssReasoningParser，处理 / 标签，是该 PR 的另一个核心新增。

```
/// SeedOSS 模型的推理解析器，使用 `<seed:think>`/`</seed:think>` 标签。
///
/// 该解析器直接委托给 `DelimitedReasoningParser`，并通过 `initialize`
/// 根据 prompt 中是否包含起始/结束 token 确定初始推理状态，无需运行时跳过标记。
pub struct SeedOssReasoningParser {
    inner: DelimitedReasoningParser,
}

impl SeedOssReasoningParser {
    /// 创建内部解析器，指定自定义标签
    pub fn new(tokenizer: DynTokenizer) -> Result<Self> {
        Ok(Self {
            inner: DelimitedReasoningParser::new(
                tokenizer,
                "<seed:think>",
                "</seed:think>",
                false, // default_in_reasoning: false，由 initialize 决定
            )?,
        })
    }
}

impl ReasoningParser for SeedOssReasoningParser {
    fn initialize(&mut self, prompt_token_ids: &[u32]) -> Result<()> {
        self.inner.initialize(prompt_token_ids); // 解析 prompt 中是否包含标记
        Ok(())
    }

    fn push(&mut self, delta: &str) -> Result<ReasoningDelta> {
        Ok(self.inner.push(delta)) // 直接委托
```

```
    }

    fn finish(&mut self) -> Result<ReasoningDelta> {
        Ok(self.inner.finish())
    }
}

// 单元测试：验证 initialize 从 prompt token 推断初始状态
#[cfg(test)]
mod tests {
    use std::sync::Arc;
    use super::SeedOssReasoningParser;
    use crate::{ReasoningParser, tests::FakeTokenizer};

    #[test]
    fn picks_up_prompt_start_boundary() {
        let tokenizer = Arc::new(FakeTokenizer);
        let mut parser = SeedOssReasoningParser::new(tokenizer).unwrap();
        // Prompt 中预填了 `<seed:think>` (token id 10)，所以初始状态应为推理中
        parser.initialize(&[10]).unwrap();

        let delta = parser.push("reason</seed:think>answer").unwrap();
        assert_eq!(delta.reasoning.as_deref(), Some("reason"));
        assert_eq!(delta.content.as_deref(), Some("answer"));
    }
}
```

评论区精华

> BugenZhao: "Shall we move these model-specific parser tests into their own modules?" yzhan1: "Moved them to corresponding modules as requested." 结论：测试已移至各自模块内，保持模块内聚。
>
> BugenZhao(issue 评论中): "I don't think this is the right direction. We'll detect there's already start/end token prefilled by checking the prompt tokens on initialize, so we're clear what the initial state is..." yzhan1: 根据建议重构了 initialize 逻辑，不再在运行时跳过标记。 结论：采纳设计方向，使用 prompt token 初始化状态，简化了运行时行为。

风险与影响

风险

- 模式匹配顺序：step3p5 模式必须在 step3 之前注册，否则 step-3p5-instruct 会被误路由。当前通过代码顺序保证，但未来维护时需警惕。
- 影响现有解析器：DelimitedReasoningParser 的初始化逻辑变更影响所有使用它的解析器（DeepSeekR1、SeedOSS、Step3p5）。单元测试已覆盖边界，但生产环境流式行为需进一步验证。

- 新逻辑复杂：Step3p5 的换行缓存状态机涉及跨 push 的状态维护，边缘情况（如极端长内容、特殊 Unicode 换行、空 push）需要更多实测。

影响

- 用户：Rust 前端现在可直接解析 Seed-OSS 和 Step-3.5 的推理输出，无需 Python fallback，提升响应速度和一致性。
- 团队：该 PR 树立了添加新推理解析器的标准模式：创建解析器文件、注册名称和模式、添加单元测试和 roundtrip 测试。未来扩展可参照此模式。

关联脉络

- 本 PR 是 Rust 前端功能对等路线图 issue #44280 的一部分，填补了 seed_oss 和 step3p5 两个解析器空白。
- 与 #44596 (Mistral 解析器重构) 属于同一功能线，共享相同的 ReasoningParserFactory 注册模式和 DelimitedReasoningParser 基础设施。
- 后续可继续为更多模型（如 Kimi K2、Minimax M2.5 等）添加 Rust 端解析器。