

PR #56366 完整报告

vllm-project/vllm

[Bugfix][Rust Frontend][Multimodal] Align DeepSeek V4.1 and Kimi K3 media with rendered placeholders

合并时间: 2026-09-14 18:38

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/56366>

[Bugfix][Rust Frontend][Multimodal] DeepSeek V4.1 与 Kimi K3 媒体顺序与渲染占位符对齐

执行摘要

本 PR 修复 Rust 前端在多模态 chat 请求中「渲染顺序」与「媒体提取顺序」各自独立推导导致的错位问题: DeepSeek V4.1 / Kimi K3 会按 assistant tool-call 顺序重排工具结果、并可能丢弃历史 developer 消息, 而 `extract_media_parts` 仍按原始请求顺序扫描, 图片因此被静默关联到错误的工具结果, 或提取到没有占位符的图片而使占位符扩展失败。方案是在渲染器写出占位符的同一处记录 `MediaPartSource`, 通过 `RenderedPrompt.media_order` 传给 chat 层按序解析, 不重排的渲染器返回 `None` 保持原行为。改动集中在 `rust/src/chat`, 共 18 个文件、+415/-85, 含 3 个新增测试; 由作者 reidliu41 提交、维护者 BugenZhao 追加 2 个重构 commit 后合入。

功能与动机

PR body 把问题定义为「两个独立顺序来源」的冲突:

Rust chat rendering and multimodal extraction previously determined media order independently. DeepSeek V4.1 and Kimi K3 can reorder tool results to match the preceding assistant tool-call order.

具体两个失败模式:

1. 工具结果乱序到达: 渲染出的图片占位符跟随 tool-call 顺序, 媒体提取仍在请求顺序, 于是「images could be silently associated with the wrong tool results」——错误是静默的, 用户只会看到模型对错图作答。
2. 历史 developer 消息被丢弃: DeepSeek V4.1 会省略这类消息, 但其中的图片仍被提取, 「causing multimodal placeholder expansion to fail」——请求直接失败。

目标因此很明确: 让渲染器报告媒体来源位置, 提取端按渲染顺序解析; 同时不能破坏 HF/Jinja 等按请求顺序渲染的既有渲染器。

实现拆解

1. 建立「渲染媒体来源」契约 (`rust/src/chat/src/renderer/mod.rs`, `rust/src/chat/src/lib.rs`) - 新增 `MediaPartSource { message_index,`

- `content_part_index` }, 指向源 `ChatRequest.messages` 中的具体 `part`。 -
- `RenderedPrompt` 增加 `media_order: Option<Vec<MediaPartSource>>`: `None` = 沿用请求顺序 (Jinja 路径), `Some` = 严格使用列表 (空列表表示媒体已被全部丢弃)。 -
- `deepseek_v4/v32`、`harmony`、`hf`、`inkling` 只填 `media_order: None`, 零行为变化。
2. DeepSeek 渲染侧边写边记 (`renderer/deepseek.rs`、`renderer/deepseek_v41/mod.rs`)
- `render_request` 拆为 `render_request_with_media_order`, 返回 (`String`, `Vec<MediaPartSource>`)。 - `message_index` 与 `&mut media_order` 贯穿 `render_system_message` / `render_developer_message` / `render_user_content_block` / `write_tool_result` / `write_chat_content`。 - 只有 V4.1 方言在写入 `< | deepseek_image | >` 时 `push` 来源; 工具结果传入重排后的真实下标 `sorted_index`, 保证记录顺序就是渲染顺序。 - `DeepSeekV41ChatRenderer::render` 返回 `media_order: Some(...)`。
3. Kimi K3 渲染器同构改造 (`renderer/kimi_k3/encoding.rs`、`renderer/kimi_k3/mod.rs`)
- `K3TokenWriter` 增加 `media_order` 字段与 `image(message_index, content_part_index)` 方法; `finish()` 改为 (`Vec<u32>`, `Vec<MediaPartSource>`)。 - `write_content` / `write_role_message` / `write_tool_message` 增加 `message_index`; `parts` 循环用 `enumerate()` 取 `content_part_index`。 - `flush_tool_run` 用的是 `pending_tool_run` 里保存的原始消息下标, 而不是重排后的 `position`。
4. 提取端按渲染顺序解析 (`rust/src/chat/src/multimodal.rs`) -
- `extract_media_parts(request, media_order)` 用 `itertools::Either` 统一双分支: `Some` 精确回溯并校验下标 (缺失时报 `renderer reported missing chat message/content part ...`) ; `None` 保持旧的请求顺序扫描并用 `is_multimodal()` 过滤。 - `finalize_rendered_prompt` 改为先取 `media_parts` 再判空, 媒体为空直接返回文本 `Prompt`。
5. 入口空判定与测试配套 (`rust/src/chat/src/lib.rs`、各 `tests.rs`) -
- `ChatRequestProcessor::finalize_rendered_prompt` 引入 `media_is_empty: Some` 看列表是否为空, `None` 退回 `!request.has_multimodal()`, 避免被丢弃媒体的请求误报 `UnsupportedMultimodalRenderer`。 - 新增测试: `extract_media_parts_uses_request_order_or_explicit_selection` (`multimodal.rs`)、`media_order_follows_reordered_tool_results` 与 `media_order_excludes_dropped_historical_developer_content` (`deepseek_v41/tests.rs`)、`media_order_follows_reordered_tool_results` (`kimi_k3/tests.rs`) ; `rust/src/server` 与 `rust/src/chat/tests/chat.rs` 同步补齐 `RenderedPrompt` 新字段。 - 无配置、`schema`、部署改动。

关注点	变更
数据结构	<code>MediaPartSource</code> 、 <code>RenderedPrompt.media_order</code>
顺序生产者	<code>write_chat_content</code> (DS)、 <code>K3TokenWriter::image</code> (K3)
顺序消费者	<code>extract_media_parts</code> → <code>prepare_multimodal</code>
默认路径	<code>None</code> 保持请求顺序, HF/Jinja 不受影响

`rust/src/chat/src/renderer/mod.rs`

引入 `MediaPartSource` 数据结构并给 `RenderedPrompt` 增加 `media_order`:
`Option<Vec<MediaPartSource>>`, 这是整个 PR 的接口契约所在: `None` 表示沿用请求顺序, `Some` 表示严格按列表来源, 空列表表示媒体已被丢弃。

```
/// 某个多模态 content part 在源 chat 请求中的位置。
#[derive(Debug, Clone, Copy, PartialEq, Eq)]
pub struct MediaPartSource {
    /// 源请求中的零基消息下标。
    pub message_index: usize,
    /// 该消息内部零基的 content part 下标。
    pub content_part_index: usize,
}

/// 提交给文本后端的已渲染 chat prompt。
#[derive(Debug, Clone, PartialEq)]
pub struct RenderedPrompt {
    /// 渲染后的 prompt, 可能是文本, 也可能已经 tokenize 完成。
    pub prompt: Prompt,
    /// 按渲染后占位符顺序排列的媒体来源。
    ///
    /// `None` 表示沿用请求中的消息与 content-part 顺序 (例如 Jinja 模板路径);
    /// `Some` 表示严格使用列出的来源, 空列表意味着渲染器丢弃了全部媒体。
    /// 渲染器只要会重排工具结果或丢弃消息, 就必须填写该字段,
    /// 否则媒体提取会与占位符顺序脱钩。
    pub media_order: Option<Vec<MediaPartSource>>,
    /// 应用服务端默认值、请求覆盖与推理控制后, 对渲染器可见的模板参数。
    pub effective_template_kwargs: HashMap<String, Value>,
}
```

rust/src/chat/src/renderer/deepseek.rs

DeepSeek V4/V4.1 共享渲染逻辑: `render_request` 拆出 `render_request_with_media_order`, 并把 `message_index` 与 `&mut media_order` 贯穿到 `render_user_content_block / write_tool_result / write_chat_content`, 是「占位符与来源同源产生」的关键实现。

```
/// 渲染一条 `<tool_result>` 内容, 并传入原始请求中的消息下标,
/// 供 `write_chat_content` 记录媒体来源使用。
fn write_tool_result(
    out: &mut String,
    content: &ChatContent,
    dialect: DsDialect,
    message_index: usize,
    media_order: &mut Vec<MediaPartSource>,
) -> Result<()> {
    out.push_str("<tool_result>");
    write_chat_content(out, content, dialect, message_index, media_order)?;
    out.push_str("</tool_result>");
    Ok(())
}
```

```

/// V4.1 方言在每个 image part 的位置无条件内联图片占位符
/// （对应 Python 编码里的 `IMAGE_PLACEHOLDER`）；其他方言拒绝多模态 part。
///
/// 这里的 `index` 与 `message_index` 组合出的 `MediaPartSource`
/// 必须与渲染顺序严格一致，chat 层才能把媒体还原到正确的工具结果上。
fn write_chat_content(
    out: &mut String,
    content: &ChatContent,
    dialect: DsDialect,
    message_index: usize,
    media_order: &mut Vec<MediaPartSource>,
) -> Result<()> {
    match content {
        ChatContent::Text(text) => out.push_str(text),
        ChatContent::Parts(parts) => {
            for (index, part) in parts.iter().enumerate() {
                match part {
                    ChatContentPart::ImageUrl { .. } if dialect == DsDialect::V41 => {
                        out.push_str(DEEPSEEK_V41_IMAGE_PLACEHOLDER);
                        // 写占位符与记录来源在同一处发生，杜绝两套顺序再次分叉
                        media_order.push(MediaPartSource {
                            message_index,
                            content_part_index: index,
                        });
                    }
                    _ => out.push_str(part.as_text()?),
                }
            }
        }
    }
    Ok(())
}

```

rust/src/chat/src/renderer/kimi_k3/encoding.rs

Kimi K3 的 K3TokenWriter 增加 `media_order` 字段与 `image()` 辅助方法，`finish()` 改为返回 `(Vec<u32>, Vec<MediaPartSource>)`，并在 `write_content/write_role_message/write_tool_message` 中补齐 `message_index`，是工具结果重排后顺序仍正确的关键。

```

/// K3 prompt 编码器：保持 Python 的分段 tokenize 边界，同时累积媒体来源。
pub(super) struct K3TokenWriter<'a> {
    tokenizer: &'a dyn Tokenizer,
    token_ids: Vec<u32>,
    /// 按占位符写出顺序记录的媒体来源，交由 `RenderedPrompt.media_order` 上报
    media_order: Vec<MediaPartSource>,
}

```

```

impl<'a> K3TokenWriter<'a> {
    /// 写出一个图片占位符并记录其来源，保证两者不会分叉。
    fn image(&mut self, message_index: usize, content_part_index: usize) -> Result<()> {

```

```

        self.control(IMAGE_PLACEHOLDER)?;
        self.media_order.push(MediaPartSource {
            message_index,
            content_part_index,
        });
        Ok(())
    }

    pub(super) fn finish(self) -> (Vec<u32>, Vec<MediaPartSource>) {
        (self.token_ids, self.media_order)
    }
}

/// 逐 part 写入消息内容; `message_index` 一路透传,
/// `content_part_index` 由 `enumerate()` 得到, 正好是 chat 层回溯所需的下标。
fn write_content(
    out: &mut K3TokenWriter<'_>,
    message_index: usize,
    content: &ChatContent,
) -> Result<()> {
    match content {
        ChatContent::Text(text) => write_text_with_images(out, text),
        ChatContent::Parts(parts) => {
            for (content_part_index, part) in parts.iter().enumerate() {
                match part {
                    ChatContentPart::Text { text } => write_text_with_images(out, text)?,
                    ChatContentPart::ImageUrl { .. } => {
                        out.image(message_index, content_part_index)?;
                    }
                    ChatContentPart::VideoUrl { .. } => {
                        return Err(Error::UnsupportedMultimodalContent("video_url"));
                    }
                    // ...其余 part 类型沿用既有处理
                }
            }
        }
        Ok(())
    }
}
}

```

评论区精华

BugenZhao (APPROVED) : Thanks @reidliu41 this is a good catch! I've added a commit to put `media_order` into `RenderedPrompt` and use `None` for default ordering.

这条评论交代了本 PR 最重要的设计决策: 媒体顺序不通过渲染器额外返回值旁路传递, 而是成为 `RenderedPrompt` 的一等字段, 并用 `Option` 把「默认请求顺序」编码进类型里。对应 commit 2 「Refactor rendered media order into `RenderedPrompt`」 (Co-authored-by: Codex) 与 commit 3 「Use match for rendered media emptiness check」。

claude[bot]: This pull request is from a fork — automated review is disabled. A repository maintainer can comment [@claude review](#) to run a one-time review.

fork PR 未触发自动评审，因此没有逐行讨论；唯一的评审证据是维护者批准 + 自行补交 commit，说明该修复方向被认可，但实现形态由维护者重写了一遍。

风险与影响

技术风险

- 渲染器与请求下标强耦合：MediaPartSource 直接指向 request.messages 的下标。DeepSeek 的 render_user_content_block 通过独立计算的 sorted_tool_response_indices 重排工具结果，本 PR 显式传入重排后的 sorted_index 才正确；未来若再新增「过滤 / 重排」规则而忘记同步索引，会触发 renderer reported missing content part ... 一类运行期错误（可观测，但会变成用户侧失败请求）。
- 静默降级为纯文本：finalize_rendered_prompt 现在只要 media_parts.is_empty() 就返回文本 Prompt。对 DeepSeek 丢弃历史 developer 图片是预期行为，但若某个渲染器误报空 media_order，请求会在缺图的情况下继续推理，且现有测试只覆盖了 DeepSeek 的丢弃场景。
- crate API 破坏性变更：RenderedPrompt 新增必填字段，workspace 内所有构造点已同步（含 rust/src/server/src/routes/tests.rs、http_client_tests.rs、grpc/tests.rs、rust/src/chat/tests/chat.rs），仓库外依赖需同步升级。
- 待验证项：Kimi K3 的 ChatContent::Text 路径走 write_text_with_images，不记录 MediaPartSource；若字符串内容格式中夹带图片占位符语义，其顺序一致性缺少测试佐证。
- 验证证据不足：PR body 的 Test Result 段为空，只能从 Issue 评论确认 CI #88787 被触发，没有最终绿灯结论；覆盖以 Rust 单元测试为主。

影响范围

- 用户：Rust OpenAI/http frontend 在「多工具调用 + 工具结果带图」与「历史 developer 消息带图」的 DeepSeek V4.1 / Kimi K3 请求上，不再出现图片错配或占位符扩展失败；HF/Jinja 路径行为不变。
- 系统：仅影响 vllm-chat 的渲染与媒体准备阶段，未触及 tokenizer、engine core 协议、KV 传输。
- 团队：确立了「会重排 / 丢弃消息的渲染器必须上报 media_order」的统一契约，后续新增渲染器可复用同一接口。

关联脉络

- 与 PR#55176（scale-out CLI 化）在 rust/src/server/src/routes/tests.rs 交汇：本 PR 需在该文件补 media_order: None，说明 Rust 前端多处改动共享同一批测试构造点。
- 与 PR#56338（Rust 前端透传按请求水印控制）属于同一演进方向：把请求级语义从协议层准确传递到渲染 / 引擎层，本 PR 传递的是「媒体顺序」这一新语义。
- 与 PR#55071（统一多模态 LoRA token 计数钩子）、PR#56786（EPD 保留媒体处理选项）共同反映仓库正在收敛多模态接口：Python 侧统一钩子、媒体选项透传，Rust 侧统一「媒体来源顺序」契约。

- 更大的趋势是 Rust 前端在快速补齐与 Python 前端等价的模型特定渲染语义（工具结果重排、历史消息丢弃、占位符位置），这类「渲染顺序 = 媒体顺序」的不变量未来可能需要在渲染器测试模板中固化，避免新增模型时重复踩坑。