PR #51478 完整报告

vllm-project/vllm

[Frontend] Add content_parts to /inference/v1/generate for raw multim···

合并时间：2026-08-11 10:00

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/51478

执行摘要

- 一句话：为 /generate 新增 content_parts 原始多模态输入
- 推荐动作：值得精读，尤其是评审中 SSRF 问题的发现与修复过程：它示范了「新入口复用时必须继承既有安全配置」的审查要点。建议关注三点：_check_mm_fields_exclusive 的互斥设计、AsyncMultiModalItemTracker 复用带来的防护一致性、以及 flat MediaContentPart schema 相对 RFC 原提案 multi_modal_data 的取舍。使用 /generate 做多模态推理的 RL 工程团队应直接采用该字段，并显式配置 --allowed-media-domains。

功能与动机

RFC #51472 指出 RL 框架（如 prime-rl）需要 token 级多模态推理，现有三条路径都不理想：/v1/completions 在 OpenAI 规范中不支持多模态；/v1/chat/completions 要求 messages + chat template，而 RL 调用方已有 token_ids；/render → /generate 虽可行但迫使序列化后的 pixel_values（单图约 MB 级）在线传输。理想形态是「token_ids + 原始媒体单次请求、服务端解析」，本 PR 以 content_parts 字段落地该需求。值得注意的是：RFC 正文原本论证 multi_modal_data 更优（避免 chat 层概念泄漏、天然支持 UUID 缓存），实现最终却选择了 OpenAI 风格 content parts 的 flat schema，以复用 Rust 侧 llm_multimodal::MediaContentPart 并消除转换层。

实现拆解

1. 协议层契约：Python vllm/entrypoints/scale_out/token_in_token_out/protocol.py 的 GenerateRequest 新增可选字段 content_parts，并通过 model_validator(mode="after") 增加 _check_mm_fields_exclusive，拒绝与预计算特征 features 同时传入，避免同一请求出现两套媒体来源；Rust rust/src/server/src/routes/inference/generate/types.rs 同步新增 content_parts: Option<Vec<MediaContentPart>>（flat schema，形如 {"type": "image_url", "url": "..."}）。
2. Python 服务端解析：vllm/entrypoints/scale_out/token_in_token_out/serving.py 的 serve_tokens 新增第一个分支：构造 AsyncMultiModalItemTracker(self.model_config) 与 parser，按类型分派 parse_image / parse_audio / parse_video，resolve_items 完成媒体抓取与预处理，拼装 TokensPrompt（multi_modal_data + multi_modal_uuids）后经 render_cmpl_async 走与 chat 相同的 renderer 管线；features 与纯文本两条既有分支保持不变。

3. Rust 服务端解析：rust/src/server/src/routes/inference/generate.rs 的 generate handler 中 take() 出 content_parts，调用 ChatLlm.prepare_media(parts, &mut body.token_ids) 得到 MmFeatures，失败映射为 invalid_request；convert.rs 的 prepare_generate_request 增加 mm_features 参数并写入 TextRequest.mm_features；render.rs 为 lower_render_request 构造的 GenerateRequest 补 content_parts: None（修复新增字段导致的编译错误）。
4. 测试配套：tests/entrypoints/scale_out/token_in_token_out/test_serving_multimodal_tokens.py 新增 test_content_parts_generates_tokens 与 test_content_parts_streaming 两个集成测试，先经 /render 获取 token_ids，再带 content_parts 调 /generate 验证普通与流式响应；按评审合并进既有多模态测试文件以复用 server fixture，避免 CI 再起一台 Qwen3-VL 实例。
5. 演进修正：初版使用嵌套 OpenAI chat 格式与 content_parts_to_media_parts 转换函数，评审后改为 flat MediaContentPart schema 并删除转换层；初版直接用 MEDIA_CONNECTOR_REGISTRY.load()（未带 SSRF 参数），评审后切回 AsyncMultiModalItemTracker；初版 skip_mm_cache=True 被移除，让 RL 多 epoch rollout 受益于多模态缓存。

关键文件：

- vllm/entrypoints/scale_out/token_in_token_out/serving.py（模块 生成服务；类别 source；类型 core-logic；符号 serve_tokens）：Python 前端核心改动：serve_tokens 新增 content_parts 分支，用 AsyncMultiModalItemTracker 完成媒体解析并送入 renderer 管线，是 Python 侧功能落点，也是 SSRF 防护修复的发生地。
- vllm/entrypoints/scale_out/token_in_token_out/protocol.py（模块 协议层；类别 source；类型 core-logic；符号 GenerateRequest, _check_mm_fields_exclusive）：协议层新增 content_parts 字段与 _check_mm_fields_exclusive 互斥校验，定义了对外 API 契约并防止与 features 预计算路径混用。
- tests/entrypoints/scale_out/token_in_token_out/test_serving_multimodal_tokens.py（模块 多模态令牌；类别 test；类型 test-coverage；符号 test_content_parts_generates_tokens, test_content_parts_streaming）：新增两个集成测试覆盖普通与流式两种调用方式，复用既有 Qwen3-VL server fixture，验证 content_parts 端到端行为。
- rust/src/server/src/routes/inference/generate.rs（模块 Rust 路由；类别 source；类型 entrypoint；符号 generate）：Rust 路由入口：在 handler 中调用 ChatLlm.prepare_media 把 content_parts 解析为 MmFeatures，并提供错误映射，是 Rust 侧功能落点。
- rust/src/server/src/routes/inference/generate/convert.rs（模块 Rust 路由；类别 source；类型 entrypoint；符号 prepare_generate_request）：prepare_generate_request 增加 mm_features 参数并透传到 TextRequest.mm_features，打通 Rust 侧数据链；相关单测同步补 None 参数。
- rust/src/server/src/routes/inference/generate/types.rs（模块 Rust 路由；类别 source；类型 entrypoint；符号 GenerateRequest）：GenerateRequest 增加 content_parts: Option>，定义 Rust 侧请求 schema。

- rust/src/server/src/routes/render.rs（模块 Rust 路由；类别 source；类型 entrypoint；符号 lower_render_request）：为 lower_render_request 构造的 GenerateRequest 补 content_parts: None，修复新增字段导致的 Rust CI 编译错误。

关键符号：_check_mm_fields_exclusive, serve_tokens, prepare_generate_request, generate, lower_render_request, test_content_parts_generates_tokens, test_content_parts_streaming

关键源码片段

vllm/entrypoints/scale_out/token_in_token_out/serving.py

Python 前端核心改动：serve_tokens 新增 content_parts 分支，用 AsyncMultiModalItemTracker 完成媒体解析并送入 renderer 管线，是 Python 侧功能落点，也是 SSRF 防护修复的发生地。

```
# ServingTokens.serve_tokens 中媒体输入解析的主分支（本次改动新增）
# content_parts 携带 OpenAI 风格原始媒体部件，服务端负责抓取与预处理，
# RL 框架只需传 token_ids + 媒体 URL，无需序列化传输像素数据。
engine_input: EngineInput
if request.content_parts:
    # 复用 chat/completions 的解析器：AsyncMultiModalItemTracker 内部
    # 使用同一套 MediaConnector 配置，从而继承 --allowed-media-domains
    # 等 SSRF 防护参数（评审中从裸 connector 切换而来）。
    tracker = AsyncMultiModalItemTracker(self.model_config)
    mm_parser = tracker.create_parser()
    for part in request.content_parts:
        ptype = part.get("type", "")
        url = part.get("url")
        uuid = part.get("uuid")
        # flat schema，形如 {"type": "image_url", "url": "https://..."}
        if ptype == "image_url":
            mm_parser.parse_image(url, uuid)
        elif ptype == "audio_url":
            mm_parser.parse_audio(url, uuid)
        elif ptype == "video_url":
            mm_parser.parse_video(url, uuid)
    # 完成媒体抓取与预处理；mm_uuids 为内容寻址缓存键，
    # 让 RL 多轮 rollout 复用已解析的媒体特征（评审后移除 skip_mm_cache）。
    mm_data, mm_uuids = await tracker.resolve_items()

    prompt = TokensPrompt(prompt_token_ids=request.token_ids)
    if mm_data:
        prompt["multi_modal_data"] = mm_data
    if mm_uuids:
        prompt["multi_modal_uuids"] = mm_uuids

    # 与 /render 共用 renderer 管线，保证特征计算与预计算路径一致
    (engine_input,) = await self.online_renderer.renderer.render_cmpl_async([prompt])
# elif features := request.features: 既有预计算特征路径，保持不变
```

vllm/entrypoints/scale_out/token_in_token_out/protocol.py

协议层新增 content_parts 字段与 _check_mm_fields_exclusive 互斥校验，定义了对外 API 契约并防止与 features 预计算路径混用。

```
# GenerateRequest 新增的原始媒体字段与互斥校验（protocol.py）
features: MultiModalFeatures | None = None
"""Multimodal hashes and placeholder positions (populated for MM inputs)."""

content_parts: list[dict[str, Any]] | None = None
"""OpenAI 风格原始媒体部件（image_url / audio_url / video_url 等），
由服务端解析媒体；与预计算特征 features 互斥。"""

# 模型级校验：禁止同时携带原始媒体与预计算特征，避免同一条请求
# 出现两套相互冲突的媒体来源（例如既给 URL 又给已处理的张量）。
@model_validator(mode="after")
def _check_mm_fields_exclusive(self) -> "GenerateRequest":
    if self.content_parts and self.features:
        raise ValueError("content_parts and features are mutually exclusive")
    return self
```

tests/entrypoints/scale_out/token_in_token_out/test_serving_multimodal_tokens.py

新增两个集成测试覆盖普通与流式两种调用方式，复用既有 Qwen3-VL server fixture，验证 content_parts 端到端行为。

```
@pytest.mark.asyncio
async def test_content_parts_generates_tokens(client, test_image):
    """content_parts 携带原始媒体应能产出输出 token。"""
    data_url = encode_image_url(test_image, format="PNG")

    # 先用 render 接口拿到渲染后的 token_ids（含图片占位 token）
    render_resp = await client.post(
        RENDER_ENDPOINT,
        json={
            "model": MODEL_NAME,
            "messages": [
                {
                    "role": "user",
                    "content": [
                        {"type": "image_url", "image_url": {"url": data_url}},
                        {"type": "text", "text": "What color is this?"},
                    ],
                }
            ],
        },
    )
    render_resp.raise_for_status()
    token_ids = render_resp.json()["token_ids"]
```

```
    # 再以 token_ids + 原始媒体部件直接请求 generate，不再传输像素数据
    gen_resp = await client.post(
        GEN_ENDPOINT,
        json={
            "token_ids": token_ids,
            "content_parts": [{"type": "image_url", "url": data_url}],
            "sampling_params": {"max_tokens": 10, "temperature": 0.0},
        },
    )
    gen_resp.raise_for_status()
    gen_data = gen_resp.json()

    assert "choices" in gen_data
    choice = gen_data["choices"][0]
    assert "token_ids" in choice
    assert len(choice["token_ids"]) > 0
```

评论区精华

评审中最关键的交锋围绕 SSRF 防护与 API schema 选择：

1. SSRF 高危（depthfirst-app[bot]）：直接 MEDIA_CONNECTOR_REGISTRY.load(envs.VLLM_MEDIA_CONNECTOR) 未传 allowed_media_domains / allowed_local_media_path / media_io_kwargs，会绕过运维配置的 --allowed-media-domains，攻击者可指向 http://169.254.169.254/... 触达内网；作者以「fix」回应并切换为 AsyncMultiModalItemTracker（与 chat 端点共用连接器与防护配置）。
2. ChatContentPart 与 MediaContentPart 重复（Isotr0py）：两种结构基本一致；作者删除中间类型与转换函数，直接采用 flat MediaContentPart schema。
3. 互斥校验（DarkLight1337）：要求阻止 content_parts 与 features 同时传入，落地为 _check_mm_fields_exclusive。
4. 多模态缓存（Isotr0py）：疑问「multi-epoch rollout 是否受益于 mm cache」；初版 skip_mm_cache=True 会跳过缓存，作者承认失误后移除，RL 重复 rollout 可复用媒体特征。
5. 为何用裸 connector 而非 item tracker（DarkLight1337）：追问后作者切回 AsyncMultiModalItemTracker，与 chat 端点保持一致的连接器与 SSRF 防护。
6. RL 是否要传预计算像素（Isotr0py）：作者回应走 HTTP 端点应传 URL 以减少传输成本、并与 vLLM 预处理逻辑保持一致；预计算诉求已有 features 字段覆盖。

- MediaConnector 缺少 SSRF 防护参数（HIGH）(security): 作者切换为 AsyncMultiModalItemTracker，复用 chat 端点的连接器配置与 SSRF 防护。
- ChatContentPart 与 MediaContentPart 结构重复 (design): 采用 flat MediaContentPart schema，删除 ChatContentPart 中间类型与 content_parts_to_media_parts 转换函数。
- content_parts 与 features 互斥校验 (correctness): 新增 model_validator _check_mm_fields_exclusive，同时传入时抛 ValueError。

- 多轮 rollout 是否受益于多模态缓存 (performance): 作者承认失误，移除 skip_mm_cache=True，RL 多轮 rollout 可复用媒体缓存。
- 直接操作 connector 还是使用 item tracker (design): 改回 AsyncMultiModalItemTracker，与 chat 端点共用连接器与防护配置。
- RL 框架是否要传预计算 pixel_values (question): 作者回应：走 HTTP 端点应传 URL 以减少传输成本并与 vLLM 预处理逻辑保持一致；预计算诉求已有 features 字段覆盖。
- content_parts 扩大 SSRF 攻击面（LOW） (security): 已由 AsyncMultiModalItemTracker 内部 _assert_url_in_allowed_media_domains 覆盖；建议未配置时增加告警日志。

风险与影响

- 风险：
  1. SSRF 风险面扩大：content_parts 让用户可控 URL 触发服务端抓取，攻击面从「仅接受预计算 token」扩展到「媒体抓取」；防护完全依赖 AsyncMultiModalItemTracker 透传的 --allowed-media-domains 配置，未配置时默认全放行（与 chat 端点行为一致），运维需显式配置，或考虑对未配置场景增加告警日志。
  2. 媒体解析错误处理：Python serving.py 分支未在片段中显示 try/except 映射，解析失败时可能依赖内部异常传播，建议补充显式错误响应；Rust 端已映射为 invalid_request，双端语义待对齐。
  3. 性能权衡：每条 generate 请求都走完整 renderer 管线，比直传 features（预计算张量）更重；依赖服务端多模态缓存消化 RL 多轮 rollouts 的重复预处理成本。
  4. 兼容性：新增可选字段向后兼容；features 与 content_parts 互斥校验防止新旧调用方混用；Rust GenerateRequest 的 serde(flatten) 保证未知字段仍被容忍，不会破坏既有调用。 - 影响：对 RL 生态（prime-rl 等）是直接利好：token 级推理与多模态输入从「绕道 chat/completions 或 render 往返」变为单请求直连，省去 MB 级像素数据在线传输。对系统而言，Python 与 Rust 双前端以各自的既有抽象（renderer 管线 / TextRequest.mm_features）接入，未引入新依赖，且为 /generate 建立了「features 预计算 + content_parts 原始媒体」的双通道格局。对团队的影响在于新增了一条需要与 chat 端点同步维护 SSRF 与媒体解析配置的入口路径，测试与文档成本需持续跟进。 - 风险标记：SSRF 风险面扩大，新增 API 字段，媒体解析错误处理，RL 多轮缓存依赖

关联脉络

- PR #51178 [Rust Frontend][gRPC] Add explicit data-parallel rank routing: 同属 Rust 前端 scale-out 推理链路（rust/src/server 路由层与 vllm/entrypoints），为同一 generate/inference 端点族扩展入口。
- PR #51144 [Rust Frontend] Support dynamic tools from developer messages: 同属 Rust chat 前端请求 / 转换层改造，涉及 renderer 与请求类型演进，与本 PR 的 Rust 双端改动模式一致。
- PR #51654 Fix chat completion 500 on non-object JSON bodies: 同为前端协议校验修复，位于 vllm/entrypoints 协议层，与 content_parts 互斥校验同一领域。