PR #47959 完整报告

vllm-project/vllm

[Rust Frontend] Integrate MM video support

合并时间：2026-07-10 16:15

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/47959

执行摘要

本 PR 在 Rust 前端中集成了视频多模态支持（video_url），并同时将原先的图像专用多模态路径重构为通用的按模态准备（per-modality preparation）架构。新增了 expand.rs（统一 placeholder 扩展）、video.rs（视频处理）和 image.rs（图像处理单独模块），核心结构 MultimodalModelInfo 现在支持图像和视频两个可选的模态。聊天模板渲染器也已适配，能够识别和分发视频内容块。这是一次影响范围较广的功能添加与内部重构。

功能与动机

PR #47530 升级了 llm-multimodal 版本，其中包含了基本的视频解码和预处理能力。此 PR 的目标是在 Rust 前端中为 OpenAI 兼容 API 添加 video_url 聊天内容支持，并同时将原有的（仅图像）多模态代码重构为模态无关的抽象，以降低未来扩展新模态（如音频）的成本。

实现拆解

1. 请求模型扩展：在 rust/src/chat/src/request.rs 的 ChatContentPart 枚举中新增 VideoUrl 变体，并添加了序列化测试。在 rust/src/server/src/routes/openai/chat_completions/convert.rs 中完成 OpenAI API 格式的转换。
2. 模态无关基础设施重构：rust/src/chat/src/multimodal.rs 是重构核心。原先只持有 image_processor 的 MultimodalModelInfo 现在持有 image: Option<ModalitySupport> 和 video: Option<ModalitySupport>。原 resolve_image_processor 被泛化为 resolve_vision_processor，同时为图像和视频提供处理器。新增了 expand、image、video 子模块。
3. 图像处理分离：创建 rust/src/chat/src/multimodal/image.rs，将原本混杂在主文件中的 prepare_images、preprocess_images、build_image_items 提取出来，专注于图像批预处理和每项特征构建。
4. 视频处理新建：创建 rust/src/chat/src/multimodal/video.rs，实现 prepare_videos、preprocess_video_clip、build_video_item。视频帧预处理使用 RGB 快速路径，回退到物化帧，通过 tokio::task::spawn_blocking 在独立阻塞线程中执行。
5. 占位符扩展提取：创建 rust/src/chat/src/multimodal/expand.rs，将原先内联的 placeholder 替换逻辑抽取为 expand_prompt_token_ids 函数，通过 ExpansionLane 管理多模态替换队列，支持交错标记一次遍历。返回的 PlaceholderRange 包含 is_embed 掩码。
6. 聊天模板渲染器适配：修改 rust/src/chat/src/renderer/hf/mod.rs，MultimodalRenderInfo 从单一 placeholder_token 改为 image_token: Option<String> 和 video_token: Option<String>，新增 TemplateContentPart::Video，在 string 和

OpenAI 两种 content format 下均能正确替换或拒绝视频内容。

7. 配置与错误处理：rust/src/chat/src/error.rs 新增 MediaConnectorError 转换；rust/src/text/src/backend/hf/model_files.rs 新增 video_preprocessor_config 加载；rust/src/chat/src/backend/hf.rs 传递视频相关文件；tensor.rs 支持视频主键 pixel_values_videos。

rust/src/chat/src/multimodal/expand.rs

核心新模块，实现多模态占位符统一扩展逻辑，是架构重构的关键抽象。

```
//! Prompt placeholder expansion shared across modalities.

use std::collections::{HashMap, VecDeque};
use llm_multimodal::{Modality, PromptReplacement};
use vllm_engine_core_client::protocol::multimodal::PlaceholderRange;
use vllm_engine_core_client::protocol::tensor::WireTensor;
use super::PreparedMedia;
use crate::error::{Error, Result, bail_multimodal};

/// One modality's queue of pending placeholder replacements.
struct ExpansionLane<'a> {
    modality: Modality,
    marker_token_id: u32,
    embed_token_id: u32,
    placeholder_token: String,
    replacements: VecDeque<&'a PromptReplacement>,
}

impl<'a> ExpansionLane<'a> {
    fn from_prepared(media: &'a PreparedMedia) -> Option<Self> {
        if media.replacements.is_empty() { return None; }
        Some(Self {
            modality: media.modality,
            marker_token_id: media.placeholder.marker_token_id,
            embed_token_id: media.placeholder.embed_token_id,
            placeholder_token: media.placeholder.token.clone(),
            replacements: media.replacements.iter().collect(),
        })
    }
}

/// Replace rendered placeholder markers with model-specific replacement
/// tokens across all modalities in one left-to-right pass.
pub(super) fn expand_prompt_token_ids(
    prompt_token_ids: &mut Vec<u32>,
    prepared: &[PreparedMedia],
) -> Result<HashMap<Modality, Vec<PlaceholderRange>>> {
    let mut lanes = prepared.iter().filter_map(ExpansionLane::from_prepared).collect::<Vec<_>>();
    if lanes.is_empty() { return Ok(HashMap::new()); }
```

```
    let replacement_growth = lanes.iter()
        .flat_map(|lane| lane.replacements.iter())
        .fold(0usize, |total, replacement| total.saturating_add(replacement.tokens.len().saturating_
        sub(1)));
    let expanded_len = prompt_token_ids.len().saturating_add(replacement_growth);
    let mut expanded = Vec::with_capacity(expanded_len);
    let mut ranges = HashMap::<Modality, Vec<PlaceholderRange>>::new();

    for &token in prompt_token_ids.iter() {
        let lane = lanes.iter_mut()
            .find(|lane| lane.marker_token_id == token && !lane.replacements.is_empty());
        let Some(lane) = lane else { expanded.push(token); continue; };
        let replacement = lane.replacements.pop_front().expect("lane queue is non-empty");
        debug_assert_eq!(replacement.modality, lane.modality);
        if replacement.tokens.is_empty() {
            bail_multimodal!("placeholder token `{}` expanded to no tokens", lane.placeholder_
            token);
        }
        let replacement_len = replacement.tokens.len();
        let is_embed = {
            let mask = replacement.tokens.iter()
                .map(|&token| token as u32 == lane.embed_token_id).collect::<Vec<_>>();
            WireTensor::from_bool(vec![replacement_len], mask).map_err(Error::Multimodal)?
        };
        let expanded_offset = expanded.len();
        expanded.extend(replacement.tokens.iter().map(|&token| token as u32));
        ranges.entry(lane.modality).or_default().push(PlaceholderRange {
            offset: expanded_offset,
            length: replacement_len,
            is_embed: Some(is_embed),
        });
    }

    for lane in &lanes {
        if !lane.replacements.is_empty() {
            bail_multimodal!(
                "placeholder token `{}` was not found in tokenized prompt for {} remaining `{}`
                item(s)",
                lane.placeholder_token, lane.replacements.len(), lane.modality
            );
        }
    }
    *prompt_token_ids = expanded;
    Ok(ranges)
}
```

rust/src/chat/src/multimodal/video.rs

视频模态处理核心，实现预处理、特征构建和快速路径回退逻辑。

```
//! Video-modality preparation.

use std::sync::Arc;
use itertools::izip;
use llm_multimodal::{FieldLayout, Modality, PreprocessedEncoderInputs, VideoClip};
use vllm_engine_core_client::protocol::dtype::ModelDtype;
use vllm_engine_core_client::protocol::multimodal::{
    MmBatchedField, MmField, MmFieldElem, MmFlatField, MmKwargsItem, MmSharedField,
    MmSlice,
    SliceSpec,
};
use super::{ModalitySupport, MultimodalModelInfo, PreparedItem, PreparedMedia, tensor};
use crate::error::{Error, Result, bail_multimodal, multimodal};

impl MultimodalModelInfo {
    /// Preprocess video clips one at a time and build per-item features.
    pub(super) async fn prepare_videos(
        &self,
        clips: Vec<Arc<VideoClip>>,
        uuids: Vec<Option<String>>,
        model_dtype: ModelDtype,
    ) -> Result<PreparedMedia> {
        let support = self.video.as_ref().ok_or_else(|| Error::UnsupportedModality {
            modality: Modality::Video.to_string(),
        })?;
        let mut replacements = Vec::with_capacity(clips.len());
        let mut items = Vec::with_capacity(clips.len());
        for (clip, uuid) in izip!(&clips, uuids) {
            let preprocessed = self.preprocess_video_clip(support, Arc::clone(clip)).await?;
            let mut clip_replacements = self.spec
                .prompt_replacements_for(&self.context, &preprocessed, Modality::Video)?;
            if clip_replacements.len() != 1 {
                bail_multimodal!(
                    "expected exactly one prompt replacement per video clip, got {}",
                    clip_replacements.len()
                );
            }
            replacements.push(clip_replacements.pop().unwrap());
            items.push(self.build_video_item(preprocessed, clip.hash.clone(), uuid, model_dtype)?);
        }
        Ok(PreparedMedia {
            modality: Modality::Video,
            placeholder: support.placeholder.clone(),
            replacements,
            items,
        })
    }

    /// Preprocess one decoded video clip, preferring the borrowed-RGB fast path.
```

```
    async fn preprocess_video_clip(
        &self,
        support: &ModalitySupport,
        clip: Arc<VideoClip>,
    ) -> Result<PreprocessedEncoderInputs> {
        let config = support.config.clone();
        let processor = support.processor;
        tokio::task::spawn_blocking(move || {
            if let Some(rgb_video) = clip.rgb_video() {
                match rgb_video.frame_refs() {
                    Ok(frame_refs) => match processor.preprocess_video_rgb(&frame_refs, &config)
                    {
                        Ok(preprocessed) => return Ok(preprocessed),
                        Err(error) => warn!(
                            error = %error.as_report(),
                            "RGB fast path failed; falling back to materialized frames"
                        ),
                    },
                    Err(error) => warn!(error, "invalid RGB frame refs; falling back"),
                }
            }
            let frames = clip.materialized_frames().map_err(|error| multimodal!("{error}"))?;
            Ok(processor.preprocess_video(&frames, &config)?)
        }).await.map_err(|error| multimodal!("video preprocessing task failed: {error}"))?
    }
}
```

rust/src/chat/src/multimodal.rs

架构重构核心，从单图像处理器改为图像 / 视频双重模态支持。

```
/// Resolved multimodal support for one loaded model.
#[derive(Clone)]
pub struct MultimodalModelInfo {
    context: MultimodalModelContext,
    spec: ResolvedMultimodalSpec,
    // 之前是单一的 image_processor: ResolvedImageProcessor,
    // 现在每个支持的模态对应一个 Optional ModalitySupport
    image: Option<ModalitySupport>,
    video: Option<ModalitySupport>,
    media_connector: Arc<MediaConnector>,
}

// ModalitySupport 同时包含处理器、占位符和配置
#[derive(Clone)]
pub(super) struct ModalitySupport {
    pub processor: &'static dyn VisionPreProcessor,
    pub placeholder: ResolvedPlaceholder,
    pub config: PreProcessorConfig,
}
```

```
impl MultimodalModelContext {
    /// 视觉处理器同时服务图像和视频模态
    fn resolve_vision_processor(&self) -> Option<&'static dyn VisionPreProcessor> {
        static REGISTRY: LazyLock<VisionProcessorRegistry> =
            LazyLock::new(VisionProcessorRegistry::with_defaults);
        REGISTRY.find(&self.model_id, self.model_type.as_deref())
    }
}

impl MultimodalModelInfo {
    // from_loaded 方法中根据模型能力加载图像和视频支持
    pub fn from_loaded(files: &ResolvedModelFiles) -> Result<Option<Self>> {
        // ...
        let image_support = if has_image {
            Some(ModalitySupport::new(placeholder_image, processor, preprocessor_config))
        } else { None };
        let video_support = if has_video {
            let video_config = video::load_video_preprocessor_config(
                files.video_preprocessor_config, files.processor_config
            )?;
            Some(ModalitySupport::new(placeholder_video, processor, video_config))
        } else { None };
        // ...
    }
}
```

评论区精华

- SSRF 安全风险（depthfirst-app[bot]）：“用户提供的 video_url 直接传入 MediaConnector 获取，没有 SSRF 保护”。结论：PR 未解决，标记为已知风险。
- Qwen3-VL 占位符偏移（chatgpt-codex-connector[bot]）：“左侧匹配可能使 offset 指向错误位置”。作者 BugenZhao 回应：“上游 llm-multimodal 的 fix 已经调整模板，offset 现在正确指向 vision_start 之前”。reviewer Isotr0py 验证通过。
- 预处理器配置 fallback（Isotr0py）：“是否应为图像也添加 processor_config.json fallback？”。作者已更新，并承认视频获取配置的 min_frames/max_frames 字段未在此 PR 处理。

风险与影响

- SSRF 漏洞：MediaConnector 直接获取用户 URL，无私有 IP 过滤。需后续添加 URL 验证或配置安全客户端。
- 视频预处理性能：preprocess_video_clip 使用 spawn_blocking，高并发下可能阻塞异步运行时。
- 模型兼容性：不同模型的视频处理配置字段各异，当前默认配置可能不适合所有模型。
- 缺少测试覆盖：无端到端集成测试，仅有序列化单元测试。回归风险较高。
- 影响范围：仅 Rust 前端部署受影响。用户可使用 video_url API；代码架构从图像专用变为模态通用，便于新模态扩展。

关联脉络

本 PR 基于上游 llm-multimodal 的视频支持（关联 PR #47530），是 vLLM 项目引入视频模态的第一步。此前已有多模态图像支持的 Rust 前端实现（见 #44301 等），本 PR 在此基础上重构为通用架构，与 Python 前端的视频支持同步。未来可在此基础上添加音频等模态。