PR #6804 完整报告

verl-project/verl

[BREAKING][rollout] feat: Add Multimodal Continuous Token

合并时间：2026-08-14 12:42

原文链接：http://prhub.com.cn/verl-project/verl/pull/6804

执行摘要

- 一句话：AgentLoop 全面启用多模态 Continuous Token 分词
- 推荐动作：值得精读：create_continuous_token_builder 的 processor 门控逻辑、_MODEL_TYPE_TO_FAMILY 精确匹配设计、VL builder 基于 MRO 组合文本边界处理的方式都很典型。计划升级到该版本的团队应先核对自定义 AgentLoop 是否使用了被移除的 apply_chat_template / continuous_token.enable；使用 Qwen3.5 VL 的团队需等待 QWEN35 → QWEN3_VL 映射的后续修复。

功能与动机

PR body 指出，agent loop 的多模态 legacy 路径逐轮独立渲染，会在轮次边界丢失或重复特殊 token：Qwen/MiMo 漏掉 <|im_end|>\n 换行、GLM 重复 <|observation|> 边界 token、gemma4 工具路径丢掉工具响应中的图片 pad，而 CT 路径与完整编码结果逐 token 一致。因此需要把 CT 的 append-only token 流扩展到多模态：CT 只负责把图片占位符展开为正确的 pad token 数，重量级 pixel 张量则在末尾统一重建。wuxibin89 在 PR 评论中明确警告“This PR will break all user defined AgentLoop”，即该设计是主动接受破坏性影响的演进。

实现拆解

实现分五步完成：

1. 扩展 ContinuousTokenBuilder 多模态钩子（verl/utils/tokenizer/continuous_token.py）：基类新增 supports_multimodal()、render_tokens_with_mm()，build_initial_tokens 增加 images/videos/audios 参数（文本 builder 忽略）；新增 VLContinuousTokenMixin 与各 VL builder（QwenVLContinuousTokenBuilder、MiMoVLContinuousTokenBuilder、Gemma4VLContinuousTokenBuilder、GLM46VContinuousTokenBuilder、KimiVLContinuousTokenBuilder、MiniMaxVLContinuousTokenBuilder、DeepSeekVL2ContinuousTokenBuilder）。VL builder 通过 Python MRO 组合文本 family builder，复用 Qwen 的 ChatML 换行重插、GLM 的 <|observation|>/<|user|> 边界处理。
2. 家族路由改造（verl/utils/tokenizer/continuous_token_wiring.py）：ContinuousTokenModelFamily 增加 9 个 VL 枚举；新增 _MODEL_TYPE_TO_FAMILY 基于 config.json 的 model_type 精确匹配（替代此前从 model/tokenizer 路径猜测的脆弱逻辑）；新增 _TEXT_TO_VL_FAMILY 升级路径（default → vldefault、gemma4 → gemma4vl）；create_continuous_token_builder 接收 processor 与 hf_model_type，对

VL family 强制要求 processor，对“文本 family + processor”的配置直接抛错。

3. Agent loop 三件套改造（agent_loop.py / single_turn_agent_loop.py / tool_agent_loop.py）：删除 legacy apply_chat_template 实例方法、system prompt 处理与 gpt-oss/gemma4/text 工具响应 fallback；新增 _assert_mm_supported 守卫，任何多模态输入在构建 token 前先校验 builder 是否支持 VL 且 processor 存在；ct_build_initial_tokens 转发多模态输入，多模态 prompt 超长时报错而非左截断（截断会破坏 placeholder 与 feature 的 1:1 对齐）；tool agent loop 把工具响应图片以真实对象 {"type": "image", "image": img} 传入消息。
4. 多模态 tensor 重建：AgentLoopWorker._compute_multi_modal_inputs 在 postprocess 阶段用最终文本 + 累积的完整图片列表重新跑 processor，产出训练所需的 pixel_values / image_grid_thw；增量合并过程中产生的 pixel tensor 一律丢弃，保证 token 流与多模态 tensor 的 TITO 一致性。
5. 测试与配置配套：scripts/chat_template_mock_trajectories.py 新增 VL mock 轨迹（vl_singleturnchat / vl_multiturnsingletool / vl_multiturnmultitool）；scripts/chat_template_checker.py 扩展 processor 渲染路径与 GLM-4V/4.5V 单轮限制警告；tests/utils/test_continuous_token_on_cpu.py 大幅补充家族推断、VL gating、placeholder 展开测试；同时删除 data.continuous_token.enable 配置键及 _generated_* trainer 配置、run_deepseek_v4_*.sh 示例中的对应开关。

关键文件：

- verl/utils/tokenizer/continuous_token.py（模块 分词构建；类别 source；类型 core-logic；符号 supports_multimodal, render_tokens_with_mm, VLContinuousTokenMixin, QwenVLContinuousTokenBuilder）：CT builder 核心实现：新增多模态钩子、VLContinuousTokenMixin 与 7 个 VL builder，是整条多模态分词链路的根基。
- verl/utils/tokenizer/continuous_token_wiring.py（模块 家族路由；类别 source；类型 core-logic；符号 create_continuous_token_builder, infer_continuous_token_model_family, resolve_continuous_token_model_family, _is_multimodal_processor）：家族路由与工厂：从脆弱的名字猜测改为 config.json model_type 精确匹配，新增 VL family 注册与文本→VL 升级表，是本次 BREAKING 的决策中枢。
- verl/experimental/agent_loop/agent_loop.py（模块 代理循环；类别 source；类型 core-logic；符号 _assert_mm_supported, ct_build_initial_tokens, apply_chat_template）：AgentLoopBase 集成点：CT 成为唯一路径，新增多模态守卫与超长 prompt 报错逻辑，删除 legacy apply_chat_template。
- verl/experimental/agent_loop/tool_agent_loop.py（模块 工具循环；类别 source；类型 core-logic；符号 _handle_processing_tools_state, _handle_pending_state）：工具响应图片以真实对象进入消息并累积，新增图片时先走守卫再变更状态，删除全部 legacy 工具响应 fallback。
- scripts/chat_template_checker.py（模块 模板校验；类别 source；类型 test-coverage；符号 _render_ids, _token_repr, _diff_context, run_continuous_token_checks）：chat-template 检查器扩展 VL 覆盖：processor 渲染 ground truth、GLM 家族警告、per-boundary merge 校验，是防回归的关键工具。

- tests/utils/test_continuous_token_on_cpu.py（模块 单元测试；类别 test；类型 test-coverage；符号 test_auto_family_inference_uses_exact_root_model_type, test_auto_family_inference_normalizes_hf_model_type, test_auto_family_inference_does_not_guess_unregistered_model_type）：CPU 单测最大增量：覆盖新家族注册表、model_type 精确推断、VL processor gating、placeholder 展开与 merge 对齐。
- scripts/chat_template_mock_trajectories.py（模块 轨迹脚本；类别 source；类型 test-coverage；符号 VLToolResponseSpec, VLToolStep, build_vl_trajectories, _format_tool_call_text）：新增 VL mock 轨迹（含用户 prompt 与工具响应内嵌图片），支撑 checker 与 CT-vs-legacy 对比。

关键符号：create_continuous_token_builder, infer_continuous_token_model_family, resolve_continuous_token_model_family, supports_multimodal, render_tokens_with_mm, VLContinuousTokenMixin, _assert_mm_supported, ct_build_initial_tokens, _compute_multi_modal_inputs, build_vl_trajectories, run_continuous_token_checks

关键源码片段

verl/utils/tokenizer/continuous_token.py

CT builder 核心实现：新增多模态钩子、VLContinuousTokenMixin 与 7 个 VL builder，是整条多模态分词链路的根基。

```
# verl/utils/tokenizer/continuous_token.py
# 多模态钩子全部集中在 VL 层：文本 builder 不携带 processor 与 mm_processor_kwargs，
# 避免纯文本路径出现无法使用的多模态状态。
class ContinuousTokenBuilder:
    """连续 token 运行时基类，VL 子类通过覆写以下钩子接入 processor 渲染。"""

    def build_initial_tokens(
        self,
        messages: list[dict[str, Any]],
        *,
        tools: list[dict[str, Any]] | None = None,
        images: list[Any] | None = None,
        videos: list[Any] | None = None,
        audios: list[Any] | None = None,
    ) -> list[int]:
        # 文本 builder 直接忽略多模态入参；VL builder 会覆写此方法走 processor。
        return self._render_tokens(messages, add_generation_prompt=True, tools=tools)

    # === Multimodal hooks（VL 子类覆写） ===

    @classmethod
    def supports_multimodal(cls) -> bool:
        """是否支持视觉输入；wiring 层用它决定是否把图片传入 CT 管线。"""
        return False

    def render_tokens_with_mm(
```

```
        self,
        messages: list[dict[str, Any]],
        images: list[Any],
        *,
        videos: list[Any] | None = None,
        audios: list[Any] | None = None,
        add_generation_prompt: bool = True,
    ) -> list[int]:
        """通过完整 processor 渲染消息，把图片占位符展开为 rollout 后端
        实际消费的 token ID。与 `_render_tokens`（仅 tokenizer）不同，
        这里会应用构造时捕获的 mm_processor_kwargs（min/max pixels 等）。

        返回值刻意不包含 pixel 张量：最终多模态 tensor 由 agent loop 的
        postprocess 阶段基于完整图片列表统一重建。
        """
        raise NotImplementedError(f"{type(self).__name__} does not implement render_tokens_with_
        mm.")
```

verl/utils/tokenizer/continuous_token_wiring.py

家族路由与工厂：从脆弱的名字猜测改为 config.json model_type 精确匹配，新增 VL family 注册与文本→VL 升级表，是本次 BREAKING 的决策中枢。

```
# verl/utils/tokenizer/continuous_token_wiring.py
def create_continuous_token_builder(
    tokenizer: Any,
    *,
    model_family: str | ContinuousTokenModelFamily = "auto",
    hf_model_type: str | None = None,
    chat_template_kwargs: dict[str, Any] | None = None,
    mm_processor_kwargs: dict[str, Any] | None = None,
    processor: Any | None = None,
    **builder_kwargs: Any,
) -> Any:
    """实例化 CT builder：区分文本与 VL 两条分支。"""
    has_mm_processor = _is_multimodal_processor(processor)
    resolved_family = resolve_continuous_token_model_family(
        model_family,
        hf_model_type=hf_model_type,
        has_multimodal_processor=has_mm_processor,
    )
    builder_cls = get_continuous_token_builder_class(resolved_family)

    if has_mm_processor:
        # --- 多模态运行：VL family 直接使用，统一文本 family 升级到 VL 版 ---
        if builder_cls.supports_multimodal():
            # mm_processor_kwargs 只在构建时注入一次，之后由 builder 内部持有
            return builder_cls(
                tokenizer,
                processor,
```

```
                chat_template_kwargs=chat_template_kwargs,
                mm_processor_kwargs=mm_processor_kwargs,
                **builder_kwargs,
            )
        # 例如 gemma4 → gemma4vl、default → vldefault
        if resolved_family in _TEXT_TO_VL_FAMILY:
            upgraded_family = _TEXT_TO_VL_FAMILY[resolved_family]
            return get_continuous_token_builder_class(upgraded_family)(
                tokenizer,
                processor,
                chat_template_kwargs=chat_template_kwargs,
                mm_processor_kwargs=mm_processor_kwargs,
                **builder_kwargs,
            )
        # 明确是文本专用 family 却配了 processor：视为配置错误，直接报错
        raise ValueError(
            f"Model resolved to the text Continuous Token family {resolved_family!r}, "
            "but a multimodal processor was provided. Register config.json model_type "
            f"{_normalize_hf_model_type(hf_model_type)!r} as a VL or unified family, "
            "or do not load a multimodal processor."
        )

    # --- 纯文本运行：VL family 缺 processor 同样报错 ---
    if builder_cls.supports_multimodal():
        raise ValueError(
            f"Model resolved to the VL Continuous Token family {resolved_family!r} "
            f"({builder_cls.__name__}), which requires a processor, but none was provided."
        )
    return builder_cls(tokenizer, chat_template_kwargs=chat_template_kwargs, **builder_kwargs)


def _is_multimodal_processor(processor: Any | None) -> bool:
    # 判定标准是 processor 是否携带标准 image_processor
    return processor is not None and getattr(processor, "image_processor", None) is not None
```

verl/experimental/agent_loop/agent_loop.py

AgentLoopBase 集成点：CT 成为唯一路径，新增多模态守卫与超长 prompt 报错逻辑，删除 legacy apply_chat_template。

```
# verl/experimental/agent_loop/agent_loop.py
    def _assert_mm_supported(self, has_multi_modal: bool) -> None:
        """多模态输入存在但当前 builder/processor 不支持时，立刻抛错。

        禁止静默回退：调用方必须在任何状态变更之前调用本方法，
        避免失败时留下半构建的 prompt。
        """
        if not has_multi_modal:
            return
        if not (self.continuous_token_builder.supports_multimodal() and self.processor is not None)
```

```
        :
            raise ValueError(
                "Multimodal inputs require a Continuous Token builder that supports multimodal "
                "AND a non-None processor, but got "
                f"supports_multimodal={self.continuous_token_builder.supports_multimodal()}, "
                f"processor={'set' if self.processor is not None else 'None'}. "
                "Use a VL base model (with its processor) or remove multimodal inputs."
            )

    async def ct_build_initial_tokens(
        self,
        messages: list[dict],
        tools: list[dict] = None,
        images: list[Image.Image] = None,
        videos: list[tuple[torch.Tensor, dict]] = None,
        audios: list[Any] = None,
    ) -> list[int]:
        """构建初始 prompt：多模态输入转发给 VL builder 展开占位符。"""
        prompt_ids = await self.loop.run_in_executor(
            None,
            lambda: self.continuous_token_builder.build_initial_tokens(
                messages, tools=tools, images=images, videos=videos, audios=audios
            ),
        )
        # 多模态 prompt 不能左截断：placeholder 必须与 multi_modal_inputs
        # 的 feature 严格 1:1 对齐，超长直接作为配置错误抛出。
        prompt_length = self.rollout_config.prompt_length
        if (images or videos or audios) and len(prompt_ids) > prompt_length:
            raise ValueError(
                f"Multimodal prompt produced {len(prompt_ids)} tokens, exceeding "
                f"rollout.prompt_length={prompt_length}. Truncating multimodal token "
                f"sequences corrupts vision/audio feature alignment, so this is treated "
                f"as a configuration error. Reduce the multimodal input size "
                f"(e.g. total_pixels / max_pixels / fps / number of frames) or "
                f"increase rollout.prompt_length."
            )
        return self._cap_text_prompt_length(prompt_ids)
```

评论区精华

review 中最有价值的交锋集中在三处：

- model family 推断方式：wuxibin89 指出“It's too fragile to infer model family from model/tokenizer path, we should infer architecture from config.json”，并举例 DeepSeek-R1 会被误判为 DEEPSEEK。最终实现改为 _MODEL_TYPE_TO_FAMILY 对 config.json 的 model_type 做精确匹配，并移除路径 / 名称正则猜测。
- Qwen3.5 VL 遗漏：qy0720 报告 qwen3_5_moe 使用 Qwen3VLProcessor 却只映射到文本 family QWEN35，运行时报 ValueError: Model resolved to the text Continuous

Token family 'qwen35', but a multimodal processor was provided。作者认可该修复方向（_TEXT_TO_VL_FAMILY 增加 QWEN35 → QWEN3_VL），但合并版本中尚未落地。

- 前缀漂移风险：gxlvera 在自审时指出，VL merge 对 full_token_ids 按长度切片而不校验 runtime_token_ids 是否为真实前缀，会导致 prefix drift 被静默接受；后续提交通过“Keep merge result token-only for VL”将 MergeResult 收窄为纯 token，并配合完整渲染路径规避该风险。
- model family 推断方式：从路径改为 config.json model_type (design): 最终实现改为 _MODEL_TYPE_TO_FAMILY 对 config.json 的 model_type 精确查表，移除路径 / 名称正则推断，并保留 _TEXT_TO_VL_FAMILY 针对统一模型的升级路径。
- 该 PR 会破坏所有用户自定义 AgentLoop (other): 标题已标 BREAKING，CT 成为唯一 tokenization 路径，data.continuous_token.enable 开关被移除；团队接受该破坏性代价以换取统一路径。
- Qwen3.5 VL（qwen3_5_moe）未注册为 VL family (correctness): 作者认可修复方向，但合并版本中 _TEXT_TO_VL_FAMILY 仍未包含 QWEN35 映射，该缺口在合并后依然存在。
- VL merge 前缀切片未校验 runtime 前缀 (correctness): 后续通过“Keep merge result token-only for VL”等提交，将 VL merge 收窄为纯 token 并依赖完整渲染路径，降低前缀漂移风险。
- hasattr 冗余检查与过度防御式编程 (style): 最终 head 代码已直接调用 builder_cls.supports_multimodal()，冗余检查被移除。

风险与影响

- 风险：主要风险有：

1. BREAKING 破坏面：data.continuous_token.enable 开关与 legacy apply_chat_template 路径被彻底移除，任何用户自定义 AgentLoop 子类或依赖旧配置的训练脚本都会直接失败（示例脚本已同步删除相关键）。
2. Qwen3.5 VL 缺口：qwen3_5_moe 在多模态数据下仍会抛 ValueError，该问题在合并版本中未修复，属于已知缺口。
3. 多模态全量重处理开销：postprocess 阶段对最终文本 + 完整图片列表重跑 processor，PR body 自述“not the most efficient way”，图片多、分辨率高时 CPU 开销明显。
4. 超长多模态 prompt 直接报错：ct_build_initial_tokens 对超过 rollout.prompt_length 的多模态 prompt 抛错而非左截断，用户必须手动调大 prompt_length 或压缩视觉输入，否则任务中断。
5. 未知 model_type 回退风险：infer_continuous_token_model_family 对未注册的 model_type 会回退到 DEFAULT/VL_DEFAULT，非标准 processor（如 Nemotron 的 InternVL 风格管道）会被 _is_multimodal_processor 误判为文本路径。 - 影响：影响范围为 agent loop（SingleTurnAgentLoop + ToolAgentLoop）的整条 rollout tokenization 链路：多模态模型（Qwen2.5-VL、Qwen3-VL、GLM-4.6V、Gemma-4、MiMo-VL）在 agentic 多轮训练中首次获得与完整编码逐 token 一致的分词结果，CT vs Legacy 对比中所有 mismatch 均为 legacy 错误而 CT 正确。对用户而言这是破坏性升级，自定义 AgentLoop 需要重写；对团队而言，工具链（chat-template checker、mock 轨迹、CPU 单测）已同步扩展，后续维护成本集中在 builder 家族注册表和多模态 tensor 重建性能。

- 风险标记：BREAKING 变更，移除 legacy AgentLoop 路径，已知缺口：Qwen3.5 VL 未注册，多模态全量重处理开销，超长多模态 prompt 报错而非截断，未知 model_type 回退依赖 processor 判定

关联脉络

- PR #7413 [sglang] fix: lora sglang e2e: 同为 rollout/agent 链路的后续修复，围绕 agent loop 的 tokenization 与 rollout 后端服务展开，无直接文件重叠。
- PR #7357 [ci] test: migrate workflows from fully_async/one_step_off_policy to v1 separate_async: agent loop / rollout 链路的 CI 演进，v1 separate_async 的 AgentLoopWorker 是 CT 默认化后的承载路径。