PR #6779 完整报告

verl-project/verl

[rollout] feat: add Continuous Token for Agentic Rollout

合并时间：2026-06-22 18:57

原文链接：http://prhub.com.cn/verl-project/verl/pull/6779

执行摘要

- 一句话：Continuous Token 多轮令牌持续化机制
- 推荐动作：值得精读。该 PR 是精心设计的架构演进，展示了如何优雅地处理多模型家族下的 tokenization 通用性。关键设计亮点包括：Builder 钩子模式、模型家族自动推断、两层验证检查器。建议在后续多模态支持和默认启用后再深入评估与旧路径一致性。

功能与动机

Agentic multi-turn rollout 需要跨轮次拼接 token，而许多模型的聊天模板并非简单追加；若边界处理不当会导致多轮提示错误。Continuous Token 提供了一个通用设计来解决四个常见陷阱：重 token 化、基于合成上下文的增量提取、边界合并、结构验证。详情见关联 Issue #6719。

实现拆解

1. 核心构建器：在 verl/utils/tokenizer/continuous_token.py 中定义了 ContinuousTokenBuilder 基类及 MergeResult 数据结构，提供 build_initial_tokens、merge_non_assistant_tokens、merge_assistant_tokens 三个公开方法与 _merge_non_assistant_token_ids 等可重写钩子。子类实现模型特定边界处理（Qwen、MiniMax、GLM、Gemma、GPT-OSS）。
2. 工厂与模型家族解析：continuous_token_wiring.py 注册了各构建器类，并通过 ContinuousTokenModelFamily 枚举和 infer_continuous_token_model_family 函数实现自动检测或显式指定模型家族。
3. AgentLoop 集成：修改了 agent_loop.py、single_turn_agent_loop.py、tool_agent_loop.py，新增 ct_build_initial_tokens、ct_merge_non_assistant_msg、ct_merge_assistant_token 异步方法，通过 enable_continuous_token 标志切换。
4. 工具解析器增强：tool_parser.py 增加了 GLM 和 Seed 模型家族的专用解析器，保留 tool_call_id 并支持数据验证。
5. 验证工具：新增 scripts/chat_template_checker.py 用于模型兼容性检查，可运行两层诊断（原始模板前缀检查和 Continuous Token 构建器检查）。
6. 测试与配置：新增 test_continuous_token_on_cpu.py、test_tool_call_id_on_cpu.py CPU 单元测试，以及 vLLM/SGLang 端到端烟雾测试。配置项 multi_turn.continuous_token 默认关闭。注意不兼容多模态（processor 非 None 时

fallback 到旧路径）。

关键文件：

- verl/utils/tokenizer/continuous_token.py（模块 令牌构建器；类别 source；类型 core-logic；符号 ContinuousTokenBuilder, MergeResult, build_initial_tokens, merge_non_assistant_tokens）：Continuous Token 核心构建器，定义合并逻辑和模型特定钩子
- verl/utils/tokenizer/continuous_token_wiring.py（模块 令牌工厂；类别 source；类型 dependency-wiring；符号 ContinuousTokenModelFamily, create_continuous_token_builder, resolve_continuous_token_model_family, infer_continuous_token_model_family）：构建器工厂与模型家族解析
- verl/experimental/agent_loop/agent_loop.py（模块 代理循环引擎；类别 source；类型 core-logic；符号 ct_build_initial_tokens, ct_merge_non_assistant_msg, ct_merge_assistant_token, _cap_text_prompt_length）：AgentLoop 集成 Continuous Token 的入口
- verl/experimental/agent_loop/tool_parser.py（模块 工具解析器；类别 source；类型 core-logic；符号 GLMToolParser, SeedToolParser, extract_tool_calls, FunctionCall.tool_call_id）：新增 GLM 和 Seed 工具解析器，保留 tool_call_id
- verl/experimental/agent_loop/tool_agent_loop.py（模块 工具代理循环；类别 source；类型 core-logic；符号 _build_assistant_message）：工具代理循环中调整助手消息构建逻辑以支持 Continuous Token
- scripts/chat_template_checker.py（模块 模板检测；类别 source；类型 entrypoint；符号 CheckResult, _load_tokenizer, _initial_messages, _render_tokens）：模型兼容性验证工具，提供两层诊断
- scripts/chat_template_mock_trajectories.py（模块 模拟轨迹；类别 source；类型 data-contract；符号 TrajectoryStep, SingleTurnTrajectory, ToolAgentTrajectory, full_messages）：提供结构化模拟轨迹数据供验证和测试使用
- tests/utils/test_continuous_token_on_cpu.py（模块 CPU 单元测试；类别 test；类型 test-coverage；符号 _DummyTokenizer, _TemplateTokenizer, test_build_initial_tokens, test_merge_non_assistant_tokens）：Continuous Token 核心逻辑的 CPU 单元测试

关键符号：ContinuousTokenBuilder.init, ContinuousTokenBuilder.build_initial_tokens, ContinuousTokenBuilder.merge_non_assistant_tokens, ContinuousTokenBuilder.merge_assistant_tokens, ContinuousTokenBuilder._merge_non_assistant_token_ids, ContinuousTokenBuilder._tokenize_tool_group, ContinuousTokenBuilder._tokenize_single_non_tool, create_continuous_token_builder, resolve_continuous_token_model_family, infer_continuous_token_model_family, BaseAgentLoop.ct_build_initial_tokens, BaseAgentLoop.ct_merge_non_assistant_msg, BaseAgentLoop.ct_merge_assistant_token, GLMToolParser.extract_tool_calls, SeedToolParser.extract_tool_calls, ToolAgentLoop._build_assistant_message

关键源码片段

verl/utils/tokenizer/continuous_token.py

Continuous Token 核心构建器，定义合并逻辑和模型特定钩子

```
# ------------------------------------------------------------------
# Continuous Token 核心构建器 —— 多轮 token 合并与边界处理
# 基类 ContinuousTokenBuilder 暴露三个公开 API：
# build_initial_tokens, merge_non_assistant_tokens, merge_assistant_tokens
# 子类通过重写 _merge_non_assistant_token_ids 等钩子处理模型特定边界
# ------------------------------------------------------------------

from dataclasses import dataclass, field

# MergeResult 记录了合并后 token 序列及元数据对齐所需的信息
@dataclass(frozen=True)
class MergeResult:
    token_ids: list[int]
    appended_token_count: int
    kind: MergeKind = "non_assistant"
    inserted_token_ids: list[int] = field(default_factory=list)
    removed_prefix_token_count: int = 0

class ContinuousTokenBuilder:
    def tokenize_non_assistant_incremental_messages(
        self,
        previous_messages: list[dict[str, Any]],
        updated_messages: list[dict[str, Any]],
        *,
        tools: list[dict[str, Any]] | None = None,
    ) -> list[int]:
        # 验证前后消息是追加关系
        self._assert_append_only(previous_messages, updated_messages)
        appended_messages = updated_messages[len(previous_messages) :]
        if not appended_messages:
            return []
        incremental_ids: list[int] = []
        # 按角色分组处理增量消息
        for group in self._iter_append_groups(appended_messages):
            role = group[0].get("role")
            if role == "tool":
                # 工具响应需要模型特定渲染（例如 GLM 用 <observation> 包裹）
                incremental_ids.extend(
                    self._tokenize_tool_group(group, previous_messages, tools=tools)
                )
            else: # user / system
                for message in group:
                    incremental_ids.extend(
                        self._tokenize_single_non_tool(
```

```
                    message, previous_messages, tools=tools
                )
            )
        return incremental_ids

    def _merge_non_assistant_token_ids(
        self,
        token_ids: list[int],
        incremental_ids: list[int],
    ) -> MergeResult:
        """默认非助手轮次合并：在 token 流末尾追加增量 token。
        子类可重写以实现边界 token 插入（如 MiniMax 需在增量前插入 assistant turn）。
        """
        return MergeResult(
            token_ids=token_ids + incremental_ids,
            appended_token_count=len(incremental_ids),
        )
```

verl/experimental/agent_loop/agent_loop.py

AgentLoop 集成 Continuous Token 的入口

```
# verl/experimental/agent_loop/agent_loop.py
# Continuous Token 集成部分
from verl.utils.tokenizer.continuous_token_wiring import create_continuous_token_builder

class BaseAgentLoop:
    def __init__(self, ...):
        # Continuous Token 初始化（仅纯文本模式）
        self.enable_continuous_token = False
        continuous_token_config = self.data_config.continuous_token
        if continuous_token_config.enable and self.processor is None:
            model_config = self.config.actor_rollout_ref.model
            self.continuous_token_builder = create_continuous_token_builder(
                self.tokenizer,
                model_family=continuous_token_config.model_family,
                model_path=model_config.path,
                chat_template_kwargs=self.apply_chat_template_kwargs,
            )
            self.enable_continuous_token = True
            self.system_prompt = None # CT 不使用旧式的 removable system prompt
        else:
            # 旧路径：使用 processor 或 tokenizer 初始化 system prompt
            processing_class = self.processor if self.processor is not None else self.tokenizer
            self.system_prompt = initialize_system_prompt(processing_class, **...)

    async def ct_build_initial_tokens(self, messages, tools=None) -> list[int]:
        """异步包装 CT 构建初始 token，通过 run_in_executor 避免阻塞事件循环。"""
        prompt_ids = await self.loop.run_in_executor(
            None,
```

```
            lambda: self.continuous_token_builder.build_initial_tokens(messages, tools=tools),
        )
        return self._cap_text_prompt_length(prompt_ids)

    async def ct_merge_non_assistant_msg(
        self, previous_messages, updated_messages, runtime_token_ids, ...
    ) -> MergeResult:
        """异步包装非助手 token 合并。"""
        merge_result = await self.loop.run_in_executor(
            None,
            lambda: self.continuous_token_builder.merge_non_assistant_tokens(
                previous_messages, updated_messages, runtime_token_ids,
                tools=tools
            ),
        )
        # 更新 assistant message 索引映射
        ...
        return merge_result

    async def ct_merge_assistant_token(
        self, messages, runtime_token_ids, response_ids, response_logprobs=None
    ) -> MergeResult:
        """异步包装助手 token 追加。"""
        merge_result = await self.loop.run_in_executor(
            None,
            lambda: self.continuous_token_builder.merge_assistant_tokens(
                messages, runtime_token_ids, response_ids,
                response_logprobs=response_logprobs
            ),
        )
        # 将记录的插入 token 信息应用到 loss mask / logprobs 对齐
        ...
        return merge_result
```

评论区精华

review 中主要讨论了三个要点：

- 默认启用还是手动开启：wuxibin89 建议单轮 / 工具代理循环默认启用，作者因多模态未支持暂不启用，约定在多模态支持 PR 中转为默认。
- 配置文件位置：wuxibin89 建议将 continuous_token 从 rollout 配置移至 data 配置（使其同时适用于 PPO 和 SFT），作者已采纳。
- JSON 解析异常处理：gemini-code-assist[bot] 指出 tool_agent_loop._build_assistant_message 中对非法 JSON 直接抛出 ValueError 会导致训练崩溃，建议降级为警告；该问题在 PR 中未看到明确修复，需后续关注。
    - Continuous Token 是否默认启用 (design): 暂时保持关闭，待多模态支持后默认启用。
    - 配置文件位置调整 (design): 作者已采纳移动。

- tool_agent_loop 中 JSON 解析异常处理 (correctness): 评论已提交，未在 PR 中看到明确修改。

风险与影响

- 风险：主要风险包括： 1) 多模态缺失：Continuous Token 当前不支持多模态输入，启用后若 processor 非 None 会静默 fallback，可能导致用户误以为启用。 2) 模型覆盖不全：仅内置 5 个模型系列，使用未知模型时使用默认 Builder，边界处理可能不正确。 3) 默认关闭导致覆盖不足：CI 仅在显式 enable 时才运行，长期可能退化。 4) 性能影响：非异步运行的 tokenize 可能阻塞事件循环（当前通过 run_in_executor 缓解，但需确认无阻塞）。 5) 与旧路径的兼容性：若用户自定义 AgentLoop 未迁移，可能出现拼接逻辑冲突。 6) 代码量巨大（+3685）引入较多新抽象，维护成本上升。
  - 影响：直接影响：引入多轮 rollout 的令牌持续化能力，未来训练 / 推理的 token 流可保持一致性。间接影响：重构了 AgentLoop 的 tokenization 路径，影响所有使用 agent_loop 的流水线（PPO、GRPO 等）。用户层面无破坏性变更，但需要了解 Continuous Token 用于新开发。团队需维护新增的构建器和解析器。测试框架新增 CPU 单元测试和 GPU 烟雾测试。
  - 风险标记：核心路径变更，多模态未支持，默认未启用，模型覆盖不全，性能阻塞风险，测试覆盖有限

关联脉络

- PR #6719 Continuous Token Support for Multi-Turn AgentLoop Rollout: 关联 Issue，本 PR 的目标和设计讨论发起处