PR #6406 完整报告

verl-project/verl

[tool] feat: add Gemma4 tool parser with stop token and response formatting

合并时间：2026-05-21 09:56

原文链接：http://prhub.com.cn/verl-project/verl/pull/6406

执行摘要

- 一句话：为 Gemma4 添加专用工具解析器，支持多轮 agent 循环中的工具调用
- 推荐动作：该 PR 设计清晰，遵循了现有扩展模式，适合精读以了解工具解析器的抽象方式和不同模型的适配策略。建议关注 stop token 注入机制和响应格式化逻辑，这两个机制未来可能扩展至更多模型。另外，考虑为 tool parser 添加单元测试，减少对端到端测试的依赖。

功能与动机

Gemma4 在工具调用生成后不会自动输出 EOS token，而是继续生成，导致模型在一个回合内产生所有工具调用并虚构响应。此外，Gemma4 的 chat template 需要包含函数名称才能正确处理工具响应。因此需要为 Gemma4 提供专门的工具解析器，包括自定义 stop token 和响应格式化。

实现拆解

1. 基类扩展：在 ToolParser 基类（verl/experimental/agent_loop/tool_parser.py）新增 stop_token_ids 属性（默认为空列表），为需要提前停止生成的模型提供扩展点。
2. 新增 Gemma4ToolParser：注册为 "gemma4" 解析器，关键方法包括：
   - __init__：设置识别 tool_call 开始和结束的特殊 token，编译正则表达式用于匹配调用格式 <|tool_call>call:func_name{params}<tool_call|>。
   - stop_token_ids：返回 <tool_call|> 的 token id，确保生成停止在工具调用结束之后。
   - _parse_arguments：解析参数表达式，将字符串和数值（包括布尔、整数、浮点数）转换为 Python 字典。
   - extract_tool_calls：异步解析响应 token，解码时保留特殊 token，使用正则提取函数名称和参数，返回内容文本和 FunctionCall 列表。
3. Agent 循环适配：在 tool_agent_loop.py 的 _handle_generating_state 方法中注入 stop token：检查当前 tool_parser 的 stop_token_ids，若不为空则与已有的 stop_token_ids 合并去重后放入 sampling_params。在 _handle_processing_tools_state 方法中为 "gemma4" 添加分支，手动构造 <|tool_response>response:func_name{value:"content"}<tool_response|> 格式的响应文本并编码为 token ids，因为 Gemma4 的 chat template 无法直接处理标准 tool role 消息。

关键文件：

- verl/experimental/agent_loop/tool_parser.py（模块 工具解析器；类别 source；类型 core-logic；符号 stop_token_ids, Gemma4ToolParser, init, _parse_arguments）：核心实现，新增基类 stop_token_ids 属性和 Gemma4ToolParser 类，定义了工具调用的解析和参数提取逻辑
- verl/experimental/agent_loop/tool_agent_loop.py（模块 Agent 循环；类别 source；类型 core-logic；符号 _handle_generating_state, _handle_processing_tools_state）：修改 agent 循环，在生成时注入 stop token，并为 Gemma4 手动格式化工具响应消息

关键符号：stop_token_ids, Gemma4ToolParser.init, Gemma4ToolParser._parse_arguments, Gemma4ToolParser.extract_tool_calls, _handle_generating_state, _handle_processing_tools_state

关键源码片段

verl/experimental/agent_loop/tool_agent_loop.py

修改 agent 循环，在生成时注入 stop token，并为 Gemma4 手动格式化工具响应消息

```
# ============================================================================
==
# _handle_generating_state：注入 stop token
# ============================================================================
==
async def _handle_generating_state(
    self, agent_data: AgentData, sampling_params: dict[str, Any], ignore_termination: bool =
    False
) -> AgentState:
    # 在生成之前，如果 tool parser 定义了 stop_token_ids，则合并到 sampling_params 中
    if self.tool_parser.stop_token_ids:
        # 合并已有的 stop_token_ids（如果有）和 parser 的 stop_token_ids，去重
        stop_token_ids = list(
            set(
                (sampling_params.get("stop_token_ids") or [])
                + self.tool_parser.stop_token_ids
            )
        )
        sampling_params = {**sampling_params, "stop_token_ids": stop_token_ids}
    # 后续生成逻辑 ...
    with simple_timer("generate_sequences", agent_data.metrics):
        output = await self.server_manager.generate(
            request_id=agent_data.request_id,
            prompt_ids=agent_data.prompt_ids,
            sampling_params=sampling_params,
            image_data=agent_data.image_data,
            video_data=agent_data.video_data,
            audio_data=agent_data.audio_data,
            mm_processor_kwargs=agent_data.mm_processor_kwargs,
        )
    # ... 后续处理
```

```
# ================================================================================
==
# _handle_processing_tools_state：Gemma4 工具响应格式化
# ================================================================================
==
async def _handle_processing_tools_state(self, agent_data: AgentData) -> AgentState:
    # ... 先构建 add_messages 和 tool_call_names ...
    if self.tool_parser_name == "gpt-oss":
        # gpt-oss 特殊处理（已有逻辑）
        ...

    elif self.tool_parser_name == "gemma4":
        # Gemma4 的 chat template 无法处理标准 tool role 消息，
        # 需要手动构造为 Gemma4 期待的格式：
        # <|tool_response>response:func_name{value:<|"|>content<|"|>}<tool_response|>
        parts = []
        for msg, name in zip(add_messages, tool_call_names, strict=True):
            content = msg.get("content", "")
            # 如果 content 是列表（多模态工具场景），提取文本部分
            if isinstance(content, list):
                content = "".join(
                    [item.get("text", "") for item in content if item.get("type") == "text"]
                )
            parts.append(
                f'<|tool_response>response:{name}{{value:<|"|>{content}<|"|>}}<tool_response|>'
            )
        tool_response_text = "".join(parts)
        response_ids = await self.loop.run_in_executor(
            None, lambda: self.tokenizer.encode(tool_response_text, add_special_tokens=False)
        )
    else:
        # 其他模型走默认 chat template
        ...
    # ... 后续处理
```

评论区精华

在 code review 中，gemini-code-assist[bot] 指出了两个高优先级问题：第一，stop token 注入直接覆盖了 sampling_params 中已有的 stop_token_ids，应改为合并；第二，工具响应格式化假设msg['content']为字符串，但在多模态工具场景下content是列表，需提取文本部分。作者 nanastassacos 采纳了建议，在最终提交中修复了这两个问题。

- stop token 注入应合并而非覆盖 (correctness): 作者采纳建议，修改为合并去重。
- 多模态工具响应 content 为列表时的错误 (correctness): 作者添加列表处理逻辑。

风险与影响

- 风险：风险较低。主要风险在于：新添加的 Gemma4ToolParser 仅在 Gemma4 模型上测试（8xH200 FSDP2 + vLLM），未在其他环境验证；stop token 合并逻辑依赖 sampling_params 的 API 稳定性（vLLM 的 stop_token_ids 参数）；手动构造的工具响应文本格式需与 Gemma4 的 chat template 保持同步，后者可能随版本变化；当前未包含单元测试，仅端到端测试。此外，多模态工具路径的 content 处理在 Gemma4 分支中已修复，但通用工具响应格式化仍沿用原始假设，可能存在隐患。
- 影响：直接用户为使用 Gemma4 模型进行多轮工具调用的训练场景。对现有支持的模型（Qwen、Hermes、GPT-OSS）无影响，因默认 stop_token_ids 为空，且仅当 tool_parser_name 为 'gemma4' 时触发特殊格式化。对系统架构引入一个轻量扩展点（基类 stop_token_ids 属性），为未来模型预留了类似定制能力。
- 风险标记：多模态路径测试不足，stop token 合并依赖上游 API，手动格式化响应与 chat template 耦合

关联脉络

- 暂无明显关联 PR