PR #40422 完整报告

vllm-project/vllm

[Feature] add cohere reasoning and tool parsers

合并时间：2026-04-29 12:07

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/40422

执行摘要

- 一句话：新增 Cohere Command A 推理与工具解析器
- 推荐动作：本 PR 值得关注设计思路：如何将外部复杂的推理 / 工具解析器以可选方式集成到 vLLM 插件体系中。尤其是 try/except 动态导入 + 错误提示的实践，以及 PyFilter 有状态对象的管理策略。建议阅读代码中的 collect_tool_schema 使用 xgrammar 从 JSON Schema 生成 EBNF 语法的手段。但注意缺少测试覆盖，使用解析器时需谨慎。

功能与动机

根据 PR 描述，目的是为 Cohere 模型（如 command-a-reasoning-08-2025）添加推理和工具解析器。依赖 Cohere 的 melody 库完成大部分处理。

实现拆解

1. 新增推理解析器(vllm/reasoning/cohere_command_reasoning_parser.py): 定义 CohereTagRegistry、CohereTagStyle 数据结构表示结构标签对，实现 collect_tool_schema 函数使用 xgrammar 从工具 Schema 生成 EBNF 语法。提供 CohereCommand3ReasoningParser 和 CohereCommand4ReasoningParser 两个解析器类，继承 ReasoningParser，分别对应不同的模型架构（Cohere2ForCausalLM 等）。
2. 新增工具解析器(vllm/tool_parsers/cohere_command_tool_parser.py): 定义 BaseCohereCommandToolParser，内部使用 cohere_melody 的 PyFilter 对象处理流式和非流式工具调用提取。CohereCommand3ToolParser 和 CohereCommand4ToolParser 继承基类，仅通过 PyFilterOptions().cmd3() 或 .cmd4() 区分配置。
3. 注册解析器(vllm/reasoning/__init__.py 和 vllm/tool_parsers/__init__.py): 在 _REASONING_PARSERS_TO_REGISTER 和 _TOOL_PARSERS_TO_REGISTER 字典中分别添加 cohere_command3 和 cohere_command4 条目，映射到实际模块和类名。
4. 更新文档(docs/features/tool_calling.md 和 docs/features/reasoning_outputs.md): 在工具调用和推理输出文档中添加 Cohere 模型的支持说明，包括启动参数和前置条件（需安装 cohere_melody）。
5. 依赖处理: cohere_melody 不作为强制依赖，在导入位置使用 try/except ImportError 进行动态导入，并在错误提示中指导用户安装。

关键文件：

- vllm/reasoning/cohere_command_reasoning_parser.py（模块 推理解析器；类别 source；类型 core-logic；符号 CohereTagRegistry, CohereTagStyle, CohereNormalizedTool, collect_tool_schema）：核心推理解析器文件，定义数据结构、EBNF 语法生成和两个解析器类，是整个 PR 的核心。
- vllm/tool_parsers/cohere_command_tool_parser.py（模块 工具解析器；类别 source；类型 core-logic；符号 BaseCohereCommandToolParser, init, adjust_request, extract_tool_calls_streaming）：核心工具解析器文件，定义基类和两个具体实现，通过 PyFilter 组件处理流式 / 非流式工具调用。
- vllm/reasoning/__init__.py（模块 推理解析器；类别 source；类型 core-logic）：注册 cohere_command3 和 cohere_command4 推理解析器，实现懒加载。
- vllm/tool_parsers/__init__.py（模块 工具解析器；类别 source；类型 core-logic）：注册 cohere_command3 和 cohere_command4 工具解析器，实现懒加载。
- docs/features/tool_calling.md（模块 文档；类别 docs；类型 documentation）：更新工具调用文档，添加 Cohere 解析器的 CLI 使用说明和依赖注意事项。
- docs/features/reasoning_outputs.md（模块 文档；类别 docs；类型 documentation）：在推理模型列表中增加 Cohere Command A Reasoning 的表格行。

关键符号：collect_tool_schema, _tool_definitions_to_schema_list, convert_schema_to_structural_tags, extract_tool_calls_streaming, extract_tool_calls, adjust_request

关键源码片段

vllm/reasoning/cohere_command_reasoning_parser.py

核心推理解析器文件，定义数据结构、EBNF 语法生成和两个解析器类，是整个 PR 的核心。

```
def collect_tool_schema(tool_schema: list[CohereNormalizedTool]) -> str:
    """Build an xgrammar EBNF grammar that matches a JSON array of tool calls.

    The grammar shape is architecture-independent; callers are responsible for
    wrapping it in the correct structural tag (see ``CohereTagStyle.tools``).
    """
    tool_dictionary: dict[str, str] = {}
    for tool in tool_schema:
        tool_name = tool["name"]
        # 构建每个工具的 JSON Schema 字符串
        json_schema = f"""{{
            "type": "object",
            "properties": {{
                "tool_call_id": {{"type": "string", "pattern": "^[0-9]+$"}},
                "tool_name": {{"type": "string", "const": "{tool_name}"}},
                "parameters": {json.dumps(tool["parameters"])}
            }}
        }}"""
        # 使用 xgrammar 将 JSON Schema 转为 EBNF 规则
        tool_grammar = str(xgr.Grammar.from_json_schema(json_schema))
```

```
        # 通过正则替换，对规则名称添加 tool_name 前缀以避免冲突
        for match in re.findall(r"\b(\w+)\s*::=", tool_grammar):
            tool_grammar = re.sub(
                rf"\b{re.escape(match)}\b", tool_name + match, tool_grammar
            )
        tool_dictionary[tool_name] = f"{tool_name} ::= {tool_name}root\n{tool_grammar}"
    # 组合最终文法：根规则匹配 JSON 数组，每个数组元素为工具选项
    tool_alternatives = "tool ::= " + " | ".join(tool_dictionary.keys())
    tool_rules = "\n    ".join(tool_dictionary.values())
    grammar = f"""root ::= tools
tools ::= ws "[" ws tool ws ("," ws tool)* ws "]" ws
ws    ::= (" " | "\\t" | "\\n")*
{tool_alternatives}
{tool_rules}
"""
    return grammar
```

vllm/tool_parsers/cohere_command_tool_parser.py

核心工具解析器文件，定义基类和两个具体实现，通过 PyFilter 组件处理流式 / 非流式工具调用。

```
class BaseCohereCommandToolParser(ToolParser):
    def __init__(
        self,
        tokenizer: TokenizerLike,
        streaming_opts: PyFilterOptions,
        unary_opts: PyFilterOptions,
    ):
        super().__init__(tokenizer)
        # 保存初始选项，以便后续为新请求创建新的 PyFilter 实例
        self.unary_opts = unary_opts
        self.melody_streaming = PyFilter(streaming_opts)
        self.melody_unary = PyFilter(unary_opts)

    def adjust_request(self, request: ChatCompletionRequest | ResponsesRequest):
        request = super().adjust_request(request)
        # Cohere 模型需要保留特殊标记以识别推理 / 工具边界
        request.skip_special_tokens = False
        return request

    def extract_tool_calls_streaming(
        self, previous_text: str, current_text: str, delta_text: str,
        previous_token_ids, current_token_ids, delta_token_ids,
        request: ChatCompletionRequest,
    ) -> DeltaMessage | None:
        # 使用 stream 模式 PyFilter 处理增量文本
        r = self.melody_streaming.write_decoded(delta_text)
        if r.content is not None:
            return DeltaMessage(content=r.content)
```

```
        if r.reasoning is not None:
            return DeltaMessage(reasoning=r.reasoning)
        if r.tool_calls:
            return DeltaMessage(
                tool_calls=[DeltaToolCall(
                    id=tc.id, index=tc.index, type="function",
                    function=DeltaFunctionCall(name=tc.name, arguments=tc.arguments),
                ) for tc in r.tool_calls]
            )
        return None

    def extract_tool_calls(
        self, model_output: str, request: ChatCompletionRequest,
    ) -> ExtractedToolCallInformation:
        # 使用新的 unary PyFilter 实例处理完整文本，避免状态污染
        result = PyFilter(self.unary_opts).process_full_text(model_output)
        tool_calls = [ToolCall(
            id=tc.id, type="function",
            function=FunctionCall(name=tc.name, arguments=tc.arguments),
        ) for tc in result.tool_calls]
        return ExtractedToolCallInformation(
            tools_called=len(tool_calls) > 0,
            tool_calls=tool_calls,
            content=result.content,
        )
```

评论区精华

主要争议点：

- 外部库依赖必要性（sfeng33, chaunceyjiang）：评审者质疑将解析逻辑外包给 cohere_melody 是否合适，建议使用 out-of-tree 插件。作者回应 out-of-tree 不满足要求，但将依赖改为可选动态导入。
- 代码质量问题（gemini-code-assist）：指出 EBNF 生成中使用 str.replace 可能导致部分匹配替换，建议改用 re.sub 并已修复；发现调试 print 语句，已删除；指出 PyFilter 状态管理问题，建议存储 unary_opts 并在每次调用时创建新实例，已按建议修改。
- 推理起始 / 结束符号（chaunceyjiang）：要求添加 reasoning_start_str 和 reasoning_end_str 属性，已作为属性添加。
- cohere-melody 依赖必要性及动态导入 (design): 作者改为 try/except 动态导入，保留依赖但设为可选。
- EBNF 生成中 str.replace 安全问题 (correctness): 作者采纳建议，已修复为 re.sub。
- PyFilter 状态管理问题 (correctness): 作者在 reasoning parser 中添加 self.unary_opts 存储，并在 extract_content_ids 中使用新实例。
- 调试 print 语句残留 (style): 作者已删除。
- 添加 reasoning_start/end_str 属性 (feature): 作者已添加为属性。

风险与影响

- 风险：
  1. 外部库依赖风险：cohere_melody 是 Cohere 提供的库，但不属于 vLLM 核心依赖。虽然采用动态导入，但该库的稳定性和兼容性不在 vLLM 控制范围内，未来可能出现 API 变更或安全问题，需关注外部库升级。
  2. 缺少测试覆盖：PR 最初包含测试但因模型权限问题被移除。当前变更没有任何自动测试，解析器的正确性无法通过 CI 验证，增加了回归风险，特别是 PyFilter 状态管理和 EBNF 生成逻辑的变更容易引入隐蔽错误。
  3. 状态管理复杂度：PyFilter 是有状态对象，在多请求并发场景或重复使用时可能出现数据污染。虽然已通过存储 unary_opts 并创建新实例进行缓解，但未经过充分测试，仍需警惕。 - 影响：用户影响：Cohere 模型用户可以指定 --tool-call-parser cohere_command3 和 --reasoning-parser cohere_command3 来启用推理和工具调用功能。如果未安装 cohere_melody，vLLM 会给出明确安装提示。对其他模型用户无影响。系统影响：无数据面影响，仅在导入和使用第三方库时增加内存占用。控制面新增两个解析器的注册条目。团队影响：需要维护与 cohere_melody 库的接口兼容性，但大部分逻辑封装在外部队列中，维护负担较低。
- 风险标记：缺少测试覆盖，外部依赖不可控，有状态对象需注意

关联脉络

- PR #41129 [New Model] Laguna XS.2 implementation：同样新增了推理解析器和工具解析器支持，与当前 PR 属于同一类功能扩展模式。