PR #43175 完整报告

vllm-project/vllm

[Frontend] Add MiniCPM5 XML tool call parser

合并时间：2026-05-27 15:39

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/43175

执行摘要

- 一句话：新增 MiniCPM5 XML 工具调用解析器
- 推荐动作：该 PR 设计清晰、测试全面，值得其他工具解析器实现参考。特别注意其流式参数增量构建方式（_streaming_args_diff）和 tokenizer 特殊字符归一化处理。建议阅读 minicpm5xml_tool_parser.py 中的注释理解关键决策。

功能与动机

MiniCPM5 模型以 XML 格式发出工具调用（如 <function name='...'><param name='...'>...</param></function>），现有解析器无法处理此格式。该 PR 填补这一空白，使 vLLM 能够为 MiniCPM5 模型启用工具调用功能。

实现拆解

1. 创建解析器模块：在 vllm/tool_parsers/minicpm5xml_tool_parser.py 中定义 MiniCPM5XMLToolParser 类，继承 ToolParser 基类。实现 adjust_request 方法（工具启用时设置 skip_special_tokens=False，因为 MiniCPM5 的 XML 标签是特殊 token）和核心 extract_tool_calls 非流式解析方法。
2. XML 解析逻辑：优先使用 lxml 的 etree 进行 XML 解析，若不可用则回退到 Python 内置 xml.etree.ElementTree。利用预编译正则（_FUNC_NAME_V1_REGEX、_PARAM_WITH_NAME_REGEX 等）进行标签提取和属性匹配，并实现 _normalize_model_output 对 tokenizer 产生的特殊字符（U+0120、U+010A）进行归一化。
3. 参数类型推断与强制转换：定义辅助函数 _get_argument_type 从 tool schema 中获取参数类型，_coerce_argument_value 根据类型对值进行转换（如数组 / 布尔值用 JSON/ast.literal_eval 解析，字符串强制序列化）。支持 CDATA 块内包含 JSON 内容。
4. 流式支持：新增 extract_tool_calls_streaming 方法，利用 _streaming_args_snapshot 和 _streaming_args_diff 在增量输出中逐步构建参数 JSON，并符合 OpenAI streaming 协议。
5. 注册与测试：在 vllm/tool_parsers/__init__.py 的 _tool_parsers 字典中新增 'minicpm5' 映射，实现 lazy loading。测试文件 tests/tool_parsers/test_minicpm5xml_tool_parser.py 包含 20+ 个测试用例，覆盖注册验证、请求调整、单次 / 多次调用、流式、边缘 case（空 / 缺失参数、别名归一化）以及额外文本过滤。

关键文件：

- vllm/tool_parsers/minicpm5xml_tool_parser.py（模块 工具解析器；类别 source；类型 core-logic；符号 _normalize_model_output, _strip_thinking_content, _streaming_args_snapshot, _streaming_args_diff）：核心解析器实现，包含 XML 标签解析、参数类型推断、流式处理和 schema 验证。是 PR 的主要变更。
- tests/tool_parsers/test_minicpm5xml_tool_parser.py（模块 测试覆盖；类别 test；类型 test-coverage；符号 _tool, make_tools_weather, make_tools_sum, make_tools_no_required）：全面的单元测试，覆盖注册、请求调整、非流式 / 流式调用、错误处理和边缘场景。是验证解析器正确性的关键。
- vllm/tool_parsers/__init__.py（模块 注册中心；类别 source；类型 configuration）：注册 minicpm5 解析器入口，使 --tool-call-parser minicpm5 生效。

关键符号：_normalize_model_output, _strip_thinking_content, _streaming_args_snapshot, _streaming_args_diff, _parse_arguments, _get_argument_type, _coerce_argument_value, _add_argument, MiniCPM5XMLToolParser.adjust_request, MiniCPM5XMLToolParser.extract_tool_calls, MiniCPM5XMLToolParser.extract_tool_calls_streaming

关键源码片段

vllm/tool_parsers/minicpm5xml_tool_parser.py

核心解析器实现，包含 XML 标签解析、参数类型推断、流式处理和 schema 验证。是 PR 的主要变更。

```
# SPDX-License-Identifier: Apache-2.0
# SPDX-FileCopyrightText: Copyright contributors to the vLLM project

import ast
import json
from collections.abc import Sequence
from typing import Any

import regex as re

from vllm.entrypoints.chat_utils import make_tool_call_id
from vllm.entrypoints.openai.chat_completion.protocol import (
    ChatCompletionRequest,
    ChatCompletionToolsParam,
)
from vllm.entrypoints.openai.engine.protocol import (
    DeltaFunctionCall,
    DeltaMessage,
    DeltaToolCall,
    ExtractedToolCallInformation,
    FunctionCall,
    ToolCall,
)
```

```
from vllm.entrypoints.openai.responses.protocol import ResponsesRequest
from vllm.logger import init_logger
from vllm.tokenizers import TokenizerLike
from vllm.tool_parsers.abstract_tool_parser import (
    Tool,
    ToolParser,
)
from vllm.tool_parsers.utils import partial_tag_overlap
from vllm.utils import random_uuid

logger = init_logger(__name__)

try:
    from lxml import etree as ET # type: ignore

    _HAS_LXML = True
except Exception: # pragma: no cover
    import xml.etree.ElementTree as ET # type: ignore

    _HAS_LXML = False

# 预编译正则，用于快速定位 tool call 标签
_FUNC_NAME_V1_REGEX = re.compile(r"<function\s+name=['\"]([^'\"]+)['\"][^>]*>")
_PARAM_WITH_NAME_REGEX = re.compile(
    r"<param\s+name=['\"]([^'\"]+)['\"]>([\s\S]*?)</param>", re.DOTALL
)
_PARAM_MISSING_NAME_REGEX = re.compile(r"<param(?![^>]*\bname=)[^>]*>", re.DOTALL)
_FUNC_BLOCK_REGEX = re.compile(r"<function.*?</function>", re.DOTALL)

# SentencePiece/GPT-style 解码器可能输出 U+0120 (Ġ) / U+010A (Ċ) 表示空格和换行
_TOKENIZER_SPACE = "\u0120"
_TOKENIZER_NEWLINE = "\u010a"



def _normalize_model_output(text: str) -> str:
    """归一化模型输出：替换 tokenizer 特殊字符，修复缩并的标签名称。"""
    if (
        _TOKENIZER_SPACE not in text
        and _TOKENIZER_NEWLINE not in text
        and "<functionname=" not in text
        and "<paramname=" not in text
    ):
        return text

    normalized = text.replace(_TOKENIZER_SPACE, " ")
    normalized = normalized.replace(_TOKENIZER_NEWLINE, "\n")
    # 一些模型输出 <functionname="foo"> 或 <paramname="bar"> 缺少空格
    normalized = normalized.replace("<functionname=", "<function name=")
    normalized = normalized.replace("<paramname=", "<param name=")
```

```
    return normalized
```

tests/tool_parsers/test_minicpm5xml_tool_parser.py

全面的单元测试，覆盖注册、请求调整、非流式 / 流式调用、错误处理和边缘场景。是验证解析器正确性的关键。

```
# SPDX-License-Identifier: Apache-2.0
# SPDX-FileCopyrightText: Copyright contributors to the vLLM project
# ruff: noqa: E501

import json
import random
from unittest.mock import MagicMock

import pytest

from tests.tool_parsers.utils import run_tool_extraction_streaming
from vllm.entrypoints.openai.chat_completion.protocol import (
    ChatCompletionRequest,
    ChatCompletionToolsParam,
)
from vllm.entrypoints.openai.engine.protocol import FunctionCall, ToolCall
from vllm.tool_parsers import ToolParser, ToolParserManager
from vllm.tool_parsers.minicpm5xml_tool_parser import MiniCPM5XMLToolParser

# 辅助函数：构造工具参数对象
def _tool(name: str, parameters: dict) -> ChatCompletionToolsParam:
    return ChatCompletionToolsParam(
        type="function",
        function={
            "name": name,
            "parameters": parameters,
        },
    )

# 创建测试用 weather 工具
def make_tools_weather() -> list[ChatCompletionToolsParam]:
    return [
        _tool(
            "get_weather",
            {
                "type": "object",
                "properties": {
                    "city": {"type": "string"},
                    "date": {"type": "string"},
                },
                "required": ["city"],
            },
        )
```

```
    ]

# 测试解析器是否注册成功
def test_registered_in_tool_parser_manager() -> None:
    cls = ToolParserManager.get_tool_parser("minicpm5")
    assert cls is MiniCPM5XMLToolParser

@pytest.fixture
def parser() -> ToolParser:
    mock_tokenizer = MagicMock()
    return MiniCPM5XMLToolParser(mock_tokenizer)
```

评论区精华

1. parser registration 语法错误（reviewer: gemini-code-assist[bot]）：注册字典项 "minicpm5": (...) 末尾缺少逗号，导致 SyntaxError。已在后续提交修复。
2. 异常处理范围过宽（reviewer: gemini-code-assist[bot]）：except Exception 将 TypeError 等也吞掉，可能掩盖工具定义中的 required 字段错误。作者改为捕获 TypeError 并记录警告，提高了可见性。
3. 缺少流式方法（reviewer: chaunceyjiang）：最初版本只实现了 extract_tool_calls 非流式方法，缺少流式支持。作者在后续提交中添加了 extract_tool_calls_streaming 及对应测试。
4. lxml XXE 安全风险（reviewer: depthfirst-app[bot]）：使用默认 lxml.etree.XMLParser 未设置 resolve_entities=False，可能允许外部实体注入攻击（低严重性）。建议启用防御性配置。该问题在最终代码中未修改，需后续跟进。

- 注册语法错误 (correctness): 作者在后续提交中添加了缺失的逗号，修复了语法错误。
- 过度宽泛的异常处理 (design): 作者改为捕获 TypeError 并记录警告，提高错误可见性。
- 缺少流式提取方法 (feature): 作者添加了 extract_tool_calls_streaming 方法及相应测试，使解析器支持流式输出。
- lxml XXE 安全风险 (security): 未在最终代码中修复（尚未设置该选项），需后续跟进加固。

风险与影响

- 风险：
  1. 正则与 XML 解析可靠性：代码同时依赖正则和 XML 解析器，模型输出格式意外变化可能导致漏解析或误解析。已用多种测试覆盖常见 case，但生产环境中模型可能输出未预料的变体。
  2. lxml XXE 安全风险：未显式设置 resolve_entities=False，若环境使用 lxml <5.0，解析器默认解析外部实体，可能导致服务器本地文件泄露。虽影响较低（模型输出受用户提示影响），但建议修复。
  3. 参数类型转换错误：_coerce_argument_value 使用 ast.literal_eval 解析参数值，可输入安全风险（如资源耗尽），但仅作用于模型输出且受 schema 限制，风险可控。

4. skip_special_tokens 变更影响：工具启用时自动设置 skip_special_tokens=False，可能影响非工具场景下的文本生成输出（保留特殊 token）。该行为与其他解析器一致，但需用户知晓。 - 影响：用户：可使用 --tool-call-parser minicpm5 为 MiniCPM5 模型启用工具调用，功能齐备（非流式与流式均支持）。系统：新增约 1.5K 行代码，工具解析器注册表增加一项，无向后兼容性问题。团队：需维护 XML 解析逻辑，但模块独立、测试完备，风险可控。

- 风险标记：lxml XXE 风险 (unresolved)，正则与 XML 双解析可靠性，参数类型转换安全，skip_special_tokens 影响

关联脉络

- 暂无明显关联 PR