PR #41154 完整报告

vllm-project/vllm

[Model] Add Apertus Tool Parser

合并时间：2026-05-18 23:20

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/41154

执行摘要

- 一句话：为 Apertus 模型添加工具调用解析器
- 推荐动作：建议认可该 PR 的设计和测试覆盖，作为未来新增工具解析器的模板。建议后续改进异常处理，将通用捕获改为具体异常。

功能与动机

重新开启旧 PR #26307，基于新版 vLLM 的工具解析器接口，使 Apertus 模型（如 swiss-ai/Apertus-70B-Instruct-2509）能够支持函数调用。PR body 明确要求兼容两种 API 格式和多轮工具调用，并提供了测试验证。

实现拆解

1. 创建解析器类：在 vllm/tool_parsers/apertus_tool_parser.py 中定义 ApertusToolParser，继承 ToolParser。核心使用正则表达式 (<|tools_prefix|>)(.*?)(<|tools_suffix|>|$) 提取工具调用 JSON 数组。非流式方法 extract_tool_calls 将完整输出中的工具块解析为 ToolCall 列表；流式方法 extract_tool_calls_streaming 通过 _extract_streaming 和 _buffer_delta_text 处理增量 token，应对特殊标记被分块的情况。
2. 注册解析器：在 vllm/tool_parsers/__init__.py 的 _TOOL_PARSERS_TO_REGISTER 中添加 'apertus' 键，指向新模块。
3. 提供聊天模板：examples/tool_chat_template_apertus.jinja 针对 Apertus 的对话格式设计，兼容两种 API 的工具调用格式，支持多轮历史和并行调用。
4. 编写测试：tests/tool_parsers/test_apertus_tool_parser.py 覆盖非流式（无工具、单工具、多参数、嵌套参数、多个工具调用、不完整调用）和流式场景（工具块不同步、前缀缓冲区、后缀缓冲区、特殊标记分块等），共 30 余个测试用例。
5. 更新文档：在 docs/features/tool_calling.md 中添加 Apertus 章节，列出支持模型、启动参数和模板路径。

关键文件：

- vllm/tool_parsers/apertus_tool_parser.py（模块 工具解析器；类别 source；类型 core-logic；符号 ApertusToolParser, init, _reset_streaming_state, adjust_request）：核心实现，定义了 ApertusToolParser 类及其非流式 / 流式工具调用提取方法
- tests/tool_parsers/test_apertus_tool_parser.py（模块 测试套件；类别 test；类型 test-coverage；符号 mock_tokenizer, parser, mock_request, TestExtractToolCalls）：

全面覆盖非流式 / 流式工具调用提取场景，包含 30 余个测试用例

- examples/tool_chat_template_apertus.jinja（模块 聊天模板；类别 other；类型 core-logic）：提供了兼容两种 API 格式的聊天模板，支持多轮工具调用
- vllm/tool_parsers/__init__.py（模块 注册中心；类别 source；类型 core-logic）：注册 apertus 解析器到 ToolParserManager，使用户可通过 --tool-call-parser apertus 启用
- docs/features/tool_calling.md（模块 用户文档；类别 docs；类型 documentation）：添加 Apertus 模型工具调用的配置说明，包括支持模型和启动参数

关键符号：ApertusToolParser.init, ApertusToolParser._reset_streaming_state, ApertusToolParser.adjust_request, ApertusToolParser._buffer_delta_text, ApertusToolParser.extract_tool_calls, ApertusToolParser.extract_tool_calls_streaming, ApertusToolParser._extract_streaming

关键源码片段

vllm/tool_parsers/apertus_tool_parser.py

核心实现，定义了 ApertusToolParser 类及其非流式 / 流式工具调用提取方法

```
# SPDX-License-Identifier: Apache-2.0
# SPDX-FileCopyrightText: Copyright contributors to the vLLM project
"""
Tool call parser for Apertus models.

Extracts tool calls from the format:
<|tools_prefix|>[{"function_name": {"arg1": "value1", ...}}, ...]<|tools_suffix|>

Used when --enable-auto-tool-choice --tool-call-parser apertus are set.
"""

import json
from collections.abc import Sequence

import regex as re
from partial_json_parser.core.options import Allow

from vllm.entrypoints.chat_utils import make_tool_call_id
from vllm.entrypoints.openai.chat_completion.protocol import ChatCompletionRequest
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 find_common_prefix, partial_json_loads

logger = init_logger(__name__)
```

```
# Apertus special tokens for tool calls
TOOL_CALLS_PREFIX = "<|tools_prefix|>"
TOOL_CALLS_SUFFIX = "<|tools_suffix|>"


class ApertusToolParser(ToolParser):
    """
    Tool call parser for Apertus models.
    Handles extraction from both non-streaming and streaming environments.
    Format: `<|tools_prefix|>[{"func": {...}}, ...]<|tools_suffix|>`
    """

    def __init__(self, tokenizer: TokenizerLike, tools: list[Tool] | None = None):
        super().__init__(tokenizer, tools)

        if not self.model_tokenizer:
            raise ValueError(
                "The model tokenizer must be passed to the ToolParser "
                "constructor during construction."
            )
        # Regex to extract tool calls block (suffix is optional for incomplete outputs)
        self.tool_call_regex = re.compile(
            rf"{re.escape(TOOL_CALLS_PREFIX)}"
            rf"(.*?)"
            rf"(?:{re.escape(TOOL_CALLS_SUFFIX)}|$)",
            re.DOTALL,
        )

        self._reset_streaming_state()

    def _reset_streaming_state(self) -> None:
        """Resets all streaming state variables for a new completion request."""
        self.buffered_delta_text = ""
        self.current_tool_id = -1
        self.current_tool_name_sent = False
        self.streamed_args_for_tool: list[str] = []

    def adjust_request(
        self, request: ChatCompletionRequest | ResponsesRequest
    ) -> ChatCompletionRequest | ResponsesRequest:
        """Forces `skip_special_tokens=False` so that tool tokens are surfaced to the engine."""
        request = super().adjust_request(request)
        if request.tools and request.tool_choice != "none":
            request.skip_special_tokens = False
        return request

    # ... additional methods like _buffer_delta_text, extract_tool_calls,
    # extract_tool_calls_streaming, _extract_streaming are defined below.
```

评论区精华

Review 中 gemini-code-assist 两次提出应将 except Exception 替换为具体的 JSONDecodeError 等异常，以避免掩盖错误（文件内评论）。该建议未获得作者回应或代码变更。另外，bbrowning 发现 docstring 中多余空格导致文档构建失败，提供了修复 diff，由作者应用到最终版本。

- 通用异常捕获应替换为具体异常 (correctness): 未在 PR 中看到修改；作者未回应，最终合并时未变更。
- 文档构建失败：空格问题 (documentation): 作者随后应用了修复，文档构建通过。

风险与影响

- 风险：主要风险来自流式解析器对特殊标记分块的处理：若 token 边界切在 <|tools_prefix|> 中间，缓冲逻辑可能出错。当前测试未覆盖超长文本或高并发场景。异常处理使用宽泛的 except Exception 可能隐藏 JSON 格式错误或正则匹配失败，增加调试难度。但这些风险对已有功能无影响，仅影响新解析器本身。
- 影响：对用户：Apertus 模型现已支持工具调用，用户可通过命令行启用，体验与其他模型一致。对系统：独立的解析器模块，无侵入性，不会影响其他工具解析器。对团队：增加了需维护的组件，但聊天模板和解析逻辑较通用，维护成本可控。
- 风险标记：异常处理不够具体，流式解析边界条件，新模块需社区验证

关联脉络

- PR #26307 [Model] Add Apertus Tool Parser: 该 PR 是 #26307 的重新开放和升级版本，基于新版 vLLM 接口重新实现。