PR #47024 完整报告

vllm-project/vllm

[Frontend] Support OpenAI Responses API namespace tools

合并时间：2026-07-06 14:21

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/47024

执行摘要

- 一句话：支持 Responses API namespace 工具名恢复
- 推荐动作：建议精读，特别是 vllm/tool_parsers/utils.py 中的映射函数设计。该 PR 解决了真实的客户端兼容性问题，设计上清晰地分离了内部展平名与外部原始名，是一个值得学习的前端协议适配案例。审核者 chaunceyjiang 的 review 提出了关键的代码组织建议，也值得关注。

功能与动机

关联 Issue #46737 报告：Codex Desktop 使用 /v1/responses API 发送 namespace 类型工具时，返回的 function_call.name 是展平后的 mcp__computer_use__get_app_state，Codex 无法执行该调用。需要在 Responses API 输出中保持外部原始工具名，同时内部仍使用展平名进行 prompt 和解析。

实现拆解

1. 在 vllm/tool_parsers/utils.py 中新增核心工具函数：ResponsesToolCallName dataclass 用于存储映射关系；flat_namespace_tool_name() 构建内部展平名；iter_response_function_tool_info() 和 iter_response_function_tool_dicts() 展开 NamespaceTool 为平面列表；build_responses_tool_call_name_map() 建立从展平名到原始名的映射字典；resolve_responses_tool_call_name() 反向解析工具名。同时修改 find_tool_properties 和 find_tool_name 以支持 namespace 工具。
2. 在 vllm/entrypoints/openai/responses/utils.py 中修改 build_response_output_items() 使其接受 tools 参数并利用映射恢复工具名；修改 _construct_message_from_response_item() 在构建历史 function_call 时使用展平名（内部表示）；修改 extract_function_tool_names() 和 construct_tool_dicts() 支持 namespace 类型。
3. 在 vllm/entrypoints/openai/responses/streaming_events.py 中修改 SimpleStreamingState 新增 tool_call_namespace 字段；emit_simple_tool_call_open() 和 emit_simple_tool_call_done() 传递 namespace；SimpleStreamingEventProcessor.init() 接受 tools 并构建映射，在 open handler 中通过 resolve_responses_tool_call_name() 还原工具名。
4. 在 vllm/entrypoints/openai/responses/protocol.py 中修改 check_tool_usage 类方法，使 named_tool_choice 验证支持 namespace 工具的内部展平名。

5. vllm/entrypoints/openai/responses/serving.py 和 context.py 做最小适配传递 tools 参数。
6. 新增回归测试 tests/entrypoints/openai/responses/test_namespace_tool_separator.py 覆盖非流式和流式场景；tests/tool_parsers/test_glm47_moe_tool_parser.py 增加 namespace 工具的 round-trip 测试验证从解析到输出的全流程。

关键文件：

- vllm/tool_parsers/utils.py（模块 工具解析器；类别 source；类型 core-logic；符号 ResponsesToolCallName, flat_namespace_tool_name, iter_response_function_tool_info, iter_response_function_tool_dicts）：核心映射逻辑所在，新增了 ResponsesToolCallName、flat_namespace_tool_name、iter_response_function_tool_info、iter_response_function_tool_dicts、build_responses_tool_call_name_map、resolve_responses_tool_call_name 等关键函数，是整个 PR 的枢纽。
- tests/entrypoints/openai/responses/test_namespace_tool_separator.py（模块 测试；类别 test；类型 test-coverage；符号 _assert_namespace_tool_call, test_namespace_tool_separator, test_namespace_tool_separator_streaming）：新增的回归测试文件，覆盖非流式和流式场景，验证工具名正确恢复以及 namespace 属性正确设置。
- vllm/entrypoints/openai/responses/streaming_events.py（模块 流式处理；类别 source；类型 core-logic；符号 init）：流式事件处理模块，修改了 SimpleStreamingState、emit_simple_tool_call_open/done 和 SimpleStreamingEventProcessor，确保 namespace 在流式事件中正确传递。
- vllm/entrypoints/openai/responses/utils.py（模块 工具函数；类别 source；类型 dependency-wiring）：Responses API 的工具函数集合，修改了 build_response_output_items、extract_function_tool_names、construct_tool_dicts 等函数以支持 namespace 映射。
- vllm/entrypoints/openai/responses/protocol.py（模块 协议层；类别 source；类型 core-logic）：修改了 check_tool_usage 验证逻辑，使 named_tool_choice 支持 namespace 工具的内部展平名。
- tests/tool_parsers/test_glm47_moe_tool_parser.py（模块 测试；类别 test；类型 test-coverage；符号 namespace_tool_request, test_namespace_tool_call_round_trip_to_responses_output）：增加了 GLM-5.2 模型的 namespace 工具 round-trip 测试，验证从解析到输出全流程。
- vllm/entrypoints/openai/responses/serving.py（模块 服务层；类别 source；类型 dependency-wiring）：适配改动，将 tools 参数传递到 build_response_output_items。

关键符号：flat_namespace_tool_name, iter_response_function_tool_info, iter_response_function_tool_dicts, build_responses_tool_call_name_map, resolve_responses_tool_call_name, build_response_output_items, extract_function_tool_names, construct_tool_dicts, emit_simple_tool_call_open, emit_simple_tool_call_done, check_tool_usage

关键源码片段

vllm/tool_parsers/utils.py

核心映射逻辑所在，新增了 ResponsesToolCallName、flat_namespace_tool_name、iter_response_function_tool_info、iter_response_function_tool_dicts、build_responses_tool_call_name_map、resolve_responses_tool_call_name 等关键函数，是整个 PR 的枢纽。

```
# vllm/tool_parsers/utils.py 新增核心映射

from dataclasses import dataclass
from openai.types.responses import FunctionTool, NamespaceTool

# 常量：内部展平分隔符
_NAMESPACE_TOOL_SEPARATOR = "__"


@dataclass(frozen=True)
class ResponsesToolCallName:
    """存储一个工具调用的外部名称和可选的 namespace。"""
    name: str
    namespace: str | None = None


def flat_namespace_tool_name(namespace: str, name: str) -> str:
    """将 namespace 和内部工具名拼接为内部展平名，例如 `mcp__computer_use__get_app_
    state`。"""
    return f"{namespace}{_NAMESPACE_TOOL_SEPARATOR}{name}"


def iter_response_function_tool_info(
    tool: ResponsesTool,
) -> list[tuple[str, dict[str, Any] | None]]:
    """从单个 tool 中提取所有函数工具的信息（name, parameters）。

    遇到 FunctionTool 直接返回其自身；遇到 NamespaceTool 则遍历其内部的 tools 列表，
    将每个嵌套函数的 name 展平后返回。非函数工具返回空列表。
    """
    if isinstance(tool, FunctionTool):
        return [(tool.name, tool.parameters)]
    if not isinstance(tool, NamespaceTool):
        return []
    namespace = tool.name
    return [
        (flat_namespace_tool_name(namespace, nested.name), nested.parameters)
        for nested in tool.tools
        if nested.type == "function"
    ]

def build_responses_tool_call_name_map(
    tools: list[ResponsesTool] | None,
```

```
) -> dict[str, ResponsesToolCallName]:
    """构建从展平名到外部原始名的映射字典。

    仅处理 NamespaceTool，非 namespace 工具不会出现在映射中（保持原样）。
    """
    if not tools:
        return {}
    name_map: dict[str, ResponsesToolCallName] = {}
    for tool in tools:
        if not isinstance(tool, NamespaceTool):
            continue
        namespace = tool.name
        for nested in tool.tools:
            if nested.type != "function":
                continue
            flat_name = flat_namespace_tool_name(namespace, nested.name)
            name_map[flat_name] = ResponsesToolCallName(
                name=nested.name, namespace=namespace
            )
    return name_map


def resolve_responses_tool_call_name(
    name: str,
    tools: list[ResponsesTool] | None = None,
    tool_call_name_map: dict[str, ResponsesToolCallName] | None = None,
) -> ResponsesToolCallName:
    """反向解析：给定一个展平的工具名（或原始名），返回外部 ResponsesToolCallName。

    如果在映射中找到，则拆分为 name 和 namespace；否则视为普通工具，namespace 为 None。
    允许传入预制映射或自动从 tools 构建。
    """
    name_map = tool_call_name_map
    if name_map is None:
        name_map = build_responses_tool_call_name_map(tools)
    return name_map.get(name, ResponsesToolCallName(name=name))
```

tests/entrypoints/openai/responses/test_namespace_tool_separator.py

新增的回归测试文件，覆盖非流式和流式场景，验证工具名正确恢复以及 namespace 属性正确设置。

```
# tests/entrypoints/openai/responses/test_namespace_tool_separator.py

import json
import openai
import pytest

MODEL_NAME = "Qwen/Qwen3-1.7B"
NAMESPACE = "mcp__computer_use"
```

```
TOOL_NAME = "get_app_state"
FLAT_TOOL_NAME = f"{NAMESPACE}__{TOOL_NAME}"

# 模拟 Codex 发送的 namespace tools 结构
tools = [
    {
        "type": "namespace",
        "name": NAMESPACE,
        "description": "Computer control tools.",
        "tools": [
            {
                "type": "function",
                "name": TOOL_NAME,
                "description": "Get the current state of a desktop application.",
                "parameters": {
                    "type": "object",
                    "properties": {"app": {"type": "string"}},
                    "required": ["app"],
                    "additionalProperties": False,
                },
            }
        ],
    }
]

prompt = [{"role": "user", "content": "Use the computer app state tool to inspect Google Chrome.
"}]


def _assert_namespace_tool_call(tool_call) -> None:
    """统一断言：工具名应为原始内部名（如 'get_app_state'），而非展平名；namespace
    属性正确。"""
    assert tool_call.type == "function_call"
    assert tool_call.name == TOOL_NAME
    assert tool_call.namespace == NAMESPACE
    assert tool_call.name != FLAT_TOOL_NAME
    args = json.loads(tool_call.arguments)
    assert args["app"]


@pytest.mark.asyncio
async def test_namespace_tool_separator(client, model_name):
    """非流式场景：返回的 tool_call 应具有原始 name 和 namespace。"""
    response = await client.responses.create(
        model=model_name,
        input=prompt,
        tools=tools,
        temperature=0.0,
    )
```

```
    tool_call = next(out for out in response.output if out.type == "function_call")
    _assert_namespace_tool_call(tool_call)


@pytest.mark.asyncio
async def test_namespace_tool_separator_streaming(client, model_name):
    """流式场景：verify both output_item.added and done events have correct names."""
    stream = await client.responses.create(
        model=model_name, input=prompt, tools=tools, temperature=0.0, stream=True
    )
    events = [event async for event in stream]
    added_call = next(
        event.item for event in events
        if event.type == "response.output_item.added"
        and getattr(event.item, "type", None) == "function_call"
    )
    done_call = next(
        event.item for event in events
        if event.type == "response.output_item.done"
        and getattr(event.item, "type", None) == "function_call"
    )
    assert added_call.name == TOOL_NAME
    assert added_call.namespace == NAMESPACE
    _assert_namespace_tool_call(done_call)
```

vllm/entrypoints/openai/responses/streaming_events.py

流式事件处理模块，修改了 SimpleStreamingState、emit_simple_tool_call_open/done 和 SimpleStreamingEventProcessor，确保 namespace 在流式事件中正确传递。

```
# vllm/entrypoints/openai/responses/streaming_events.py 部分关键改动

from vllm.entrypoints.openai.responses.utils import (
    build_responses_tool_call_name_map,
    resolve_responses_tool_call_name,
)

@dataclass
class SimpleStreamingState:
    # ... 原有字段 ...
    tool_call_namespace: str | None = None # 新增：用于存储当前流式工具调用的 namespace


def emit_simple_tool_call_open(
    state: SimpleStreamingState,
    name: str,
    index: int | None,
    namespace: str | None = None, # 新增参数
) -> list[StreamingResponsesResponse]:
    # ... 设置 state ...
```

```
    state.tool_call_name = name
    state.tool_call_namespace = namespace # 保存 namespace
    # ... 构造 added 事件时携带 namespace
    return [ResponseOutputItemAddedEvent(
        ...,
        item=ResponseFunctionToolCall(
            id=..., call_id=..., name=name, namespace=namespace,
            arguments="", status="in_progress",
        ),
    )]


def emit_simple_tool_call_done(state, ...) -> list[StreamingResponsesResponse]:
    # ... 构造 done 事件时使用 state.tool_call_namespace
    item=ResponseFunctionToolCall(
        type="function_call",
        name=state.tool_call_name,
        namespace=state.tool_call_namespace,
        arguments=state.accumulated_text,
        status="completed",
        id=state.current_item_id,
        call_id=...,
    )
    state.tool_call_namespace = None # 重置

class SimpleStreamingEventProcessor:
    def __init__(self, state=None, tools=None):
        self.state = state or SimpleStreamingState()
        self.tool_call_name_map = build_responses_tool_call_name_map(tools)

    def open(self, delta_message):
        if target_state == _StateType.TOOL_CALL:
            call_name = resolve_responses_tool_call_name(
                tool_call.function.name,
                tool_call_name_map=self.tool_call_name_map,
            )
            return handlers.open_fn(
                self.state,
                call_name.name,
                tool_call.index,
                call_name.namespace,
            )
```

评论区精华

审核者 chaunceyjiang 提出关键设计建议：

- 将部分工具映射逻辑从 vllm/entrypoints/openai/responses/utils.py 移至 vllm/tool_parsers/utils.py 以集中管理（已采纳）。

- 要求 find_tool_name() 和 find_tool_properties() 支持递归检查 namespace 工具的内部函数（已在最终实现中覆盖）。
- 对 iter_response_function_tool_dicts() 提出简化建议：去掉单独的 FunctionTool 分支（已在后续提交中调整）。
- 审核者本地测试了 Qwen3.5 和 GLM-5.2，初期 GLM-5.2 异常，最终两个模型均验证通过。
- 建议在 test_function_call.py 中增加 E2E 测试，作者同意在后续 PR 添加。
- 代码组织：将工具映射逻辑移至 vllm/tool_parsers/utils.py (design): 已采纳，新函数全部位于 vllm/tool_parsers/utils.py。
- find_tool_properties 和 find_tool_name 需要支持 namespace (correctness): 已通过 iter_response_function_tool_info 扩展支持，在 find_tool_properties 循环中增加对 NamespaceTool 的处理。
- iter_response_function_tool_dicts 处理 FunctionTool 分支合理性 (design): 已按建议简化，移除了 FunctionTool 分支。
- E2E 测试的进一步补充 (testing): 作者同意在后续 PR 中添加。

风险与影响

- 风险：
  1. 兼容性风险：修改了 Responses API 多个核心函数的行为（build_response_output_items、emit_simple_tool_call_*），若映射逻辑未正确覆盖所有调用路径，可能导致非 namespace 工具遭受回归（如 name 字段意外被修改）。但设计通过分离内部展平名与外部原始名、仅在有映射时修改名称，降低了影响。
  2. 流式事件一致性风险：output_item.added 和 output_item.done 事件中的 name 必须一致，测试验证了这点。
  3. 历史回放风险：当 replay 带有 namespace 的 function_call 历史时，需要反向展平以匹配内部格式，_construct_message_from_response_item 中进行了处理。
  4. GLM-5.2 工具解析器适配风险：早期测试显示 GLM-5.2 异常，但后续修复后通过。对非 Qwen 模型可能仍有边界情况。
  5. 配置键调整风险：check_tool_usage 中的工具名集合现在同时包含原始名和展平名，可能影响输入验证逻辑。 - 影响：用户影响：使用 Responses API namespace 工具（如 Codex 的 Computer Use）的客户端将正常工作，获取到可执行的原始工具名和 namespace 属性。对其他类型工具（function、web_search 等）无影响。系统影响：核心工具映射逻辑集中在 vllm/tool_parsers/utils.py，新增了约 130 行纯函数，依赖清晰；流式处理模块新增了 namespace 字段，状态机复杂度略有增加。团队影响：明确了 namespace 工具在 Responses 管道中的处理策略，后续添加新的工具类型可参考此模式。 - 风险标记：核心路径变更，多模型适配差异，流式 / 非流式一致性，GLM-5.2 初始异常

关联脉络

- PR #47379 [Bugfix][Frontend][gpt-oss] Recover raw tail when Harmony parser ends non-terminal: 同为 Responses API 相关 bugfix，涉及 Harmony 解析器和 responses 输

出正确性，属于同一功能领域。