

PR #55063 完整报告

vllm-project/vllm

[Model] Add K2-Horizon model support

合并时间: 2026-09-03 16:17

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/55063>

执行摘要

- 一句话: 为 vLLM 添加 K2-Horizon 模型完整支持, 包含推理与工具调用解析。
- 推荐动作: 这是一个标准的新模型添加 PR, 变更量较大但结构清晰。对于需要支持或使用 K2-Horizon 模型的团队, 此 PR 值得精读, 以了解其实现细节, 特别是 MoE 路由和 IFM 工具调用格式的解析逻辑。对于关注模型扩展方法的开发者, 此 PR 提供了完整的参考案例。

功能与动机

根据 PR body, 核心动机是为 IFM 的 K2-Horizon 模型 (特别是 36B 规格) 添加 vLLM 原生推理支持, 使其能够通过 OpenAI 兼容的 API 提供服务, 并支持其特有的推理 (reasoning) 和工具调用 (tool calling) 功能。

实现拆解

1. 模型架构实现: 在 `vllm/model_executor/models/k2_horizon.py` 中新增 `K2HorizonForCausalLM` 模型类。实现了模型的层结构 (Attention, MLP, MoE), 并定义了多个关键辅助函数: `split_to_interleaved/interleaved_to_split` 用于处理交错排列的张量, `calc_router_weights` 实现了支持 sigmoid/softmax 的路由器, `fused_mova_impl` 封装了 MoE 的融合前向传播逻辑, `K2HorizonRMSNorm` 实现了分组的 RMSNorm。
2. 工具调用解析器: 在 `vllm/tool_parsers/k2_horizon_tool_parser.py` 中新增 `K2HorizonToolParser` 类。该解析器支持三种工具调用格式 (json, xml, xml_typed), 通过 `_request_tool_format` 从请求参数中动态选择格式, 并使用复杂的正则表达式来解析模型输出中的 `<ifmltool_calls>` 等标记。
3. 推理解析器: 在 `vllm/reasoning/k2_horizon_reasoning_parser.py` 中新增 `K2HorizonReasoningParser` 类。该解析器继承自 `DeepSeekR1ReasoningParser`, 支持三种思考力度 (high, medium, low), 对应不同的起始 / 结束标记 (如 `<ifmlthink>`), 并实现了流式和非流式的推理内容提取。
4. 注册与文档: 更新 `vllm/model_executor/models/registry.py` 和 `tests/models/registry.py` 以注册新模型。在 `vllm/reasoning/__init__.py` 和 `vllm/tool_parsers/__init__.py` 中注册新的解析器。更新 `docs/models/supported_models.md` 文档。
5. 测试覆盖: 新增 `tests/tool_parsers/test_k2_horizon_tool_parser.py` 和 `tests/reasoning/test_k2_horizon_reasoning_parser.py`, 全面测试解析器在不同格式、参数和边界情况下的行为。

关键文件：

- `vllm/model_executor/models/k2_horizon.py`（模块 模型架构；类别 source；类型 core-logic；符号 `K2HorizonForCausalLM`, `calc_router_weights`, `fused_mova_impl`, `K2HorizonRMSNORM`）：核心模型实现文件，包含完整的 K2-Horizon 模型架构定义，包括自定义的 MoE 路由和归一化层。
- `vllm/tool_parsers/k2_horizon_tool_parser.py`（模块 工具解析；类别 source；类型 dependency-wiring；符号 `K2HorizonToolParser`, `_request_tool_format`, `_parse_xml_call`, `extract_tool_calls`）：实现 K2-Horizon 特有的 IFM 工具调用格式解析器，支持 JSON 和 XML 两种格式，并处理流式与非流式场景。
- `vllm/reasoning/k2_horizon_reasoning_parser.py`（模块 推理解析；类别 source；类型 dependency-wiring；符号 `K2HorizonReasoningParser`, `_EFFORT_TOKENS`, `_split_model_output`, `extract_reasoning_streaming`）：实现 K2-Horizon 的推理解析器，支持多级思考力度并处理与工具调用的衔接。
- `tests/tool_parsers/test_k2_horizon_tool_parser.py`（模块 工具解析；类别 test；类型 test-coverage；符号 `test_xml_default_pares_multiple_calls_and_coerces_schema`, `test_request_level_format_overrides`, `test_character_split_streaming`）：为工具解析器提供全面的单元测试，覆盖多种格式、参数组合和边界情况。
- `tests/reasoning/test_k2_horizon_reasoning_parser.py`（模块 推理解析；类别 test；类型 test-coverage；符号 `test_reasoning_effort_selects_tokens`, `test_composed_streaming_reasoning_to_tool_handoff`）：为推理解析器提供单元测试，验证不同推理力度和流式处理逻辑。

关键符号：`calc_router_weights`, `fused_mova_impl`, `K2HorizonRMSNORM.forward`, `K2HorizonToolParser.extract_tool_calls`, `K2HorizonToolParser._parse_xml_call`, `K2HorizonReasoningParser._split_model_output`, `K2HorizonReasoningParser.extract_reasoning_streaming`

关键源码片段

`vllm/model_executor/models/k2_horizon.py`

核心模型实现文件，包含完整的 K2-Horizon 模型架构定义，包括自定义的 MoE 路由和归一化层。

```
# 计算路由器权重，支持 softmax 和 sigmoid 两种评分函数，并支持 e_score_correction_bias。
def calc_router_weights(
    router_logits: torch.Tensor,
    e_score_correction_bias: torch.Tensor | None,
    score_func: str,
    top_k: int,
    scaling_factor: float | None,
    renormalize: bool | None = None,
) -> tuple[torch.Tensor, torch.Tensor]:
    if score_func == "softmax":
        routing_scores = F.softmax(router_logits, dim=-1, dtype=torch.float32)
    else:
```

```

# 仅支持 sigmoid 作为备选评分函数。
assert score_func == "sigmoid", (
    f"Unsupported router score function: {score_func}"
)
routing_scores = torch.sigmoid(router_logits.to(torch.float32))

# 如果提供，将偏置加到用于选择的分数上。
selection_scores = routing_scores
if e_score_correction_bias is not None:
    selection_scores = selection_scores + e_score_correction_bias.to(
        selection_scores
    )

# 选择 top_k 个专家。
selected_indices = torch.topk(selection_scores, top_k, dim=-1).indices
# 从原始的 routing_scores（而非 selection_scores）中收集权重。
routing_weights = torch.gather(routing_scores, dim=-1, index=selected_indices)

# 默认在 top_k > 1 时重新归一化权重。
if renormalize is None:
    renormalize = top_k > 1
if renormalize:
    routing_weights = routing_weights / routing_weights.sum(dim=-1, keepdim=True)

# 应用可选的缩放因子。
if scaling_factor is not None:
    routing_weights = routing_weights * scaling_factor
return routing_weights.to(router_logits.dtype), selected_indices

class K2HorizonRMSNorm(RMSNorm):
    def __init__(self, hidden_size: int, n_groups: int, eps: float = 1e-6):
        assert hidden_size % n_groups == 0
        super().__init__(hidden_size, eps=eps)
        self.hidden_size = hidden_size
        self.n_groups = n_groups
        self.group_size = hidden_size // n_groups

    def forward(self, x, residual=None):
        if self.n_groups == 1:
            return super().forward(x, residual)

        if residual is not None:
            x = x + residual
            residual = x

        orig_shape = x.shape
        # 将输入重塑为 (... , n_groups, group_size)，然后合并前两个维度以高效批处理。
        x_grouped = x.reshape(-1, self.n_groups, self.group_size).reshape(

```

```

        -1, self.group_size
    )
    y_grouped = torch.empty_like(x_grouped)

    # 对每个组独立复用 vLLM 的 RMSNorm CUDA 核心。
    ops.rms_norm(
        y_grouped,
        x_grouped,
        torch.ones(self.group_size, dtype=x.dtype, device=x.device),
        self.variance_epsilon,
    )

    # 恢复原始形状并应用可学习权重。
    y = y_grouped.reshape(orig_shape) * self.weight
    return (y, residual) if residual is not None else y

```

vllm/tool_parsers/k2_horizon_tool_parser.py

实现 K2-Horizon 特有的 IFM 工具调用格式解析器，支持 JSON 和 XML 两种格式，并处理流式与非流式场景。

```

class K2HorizonToolParser(ToolParser):
    """Parser for K2 Horizon's IFM tool-call formats."""

    supports_required_and_named = False
    engine_based_streaming = True

    # 定义支持的工具调用格式。
    _SUPPORTED_TOOL_FORMATS = frozenset({"json", "xml", "xml_typed"})

    # 定义 IFM XML 格式的标记。
    _TOOL_CALLS_START = "<ifmltool_calls>"
    _TOOL_CALLS_END = "</ifmltool_calls>"
    _TOOL_CALL_START = "<ifmltool_call>"
    _TOOL_CALL_END = "</ifmltool_call>"
    _ARG_KEY_START = "<ifmlarg_key>"
    _ARG_KEY_END = "</ifmlarg_key>"
    _ARG_TYPE_START = "<ifmlarg_type>"
    _ARG_TYPE_END = "</ifmlarg_type>"
    _ARG_VALUE_START = "<ifmlarg_value>"
    _ARG_VALUE_END = "</ifmlarg_value>"

    # 用于匹配单个工具调用的正则表达式。
    _TOOL_CALL_RE = re.compile(
        re.escape(_TOOL_CALL_START) + r"(.*)" + re.escape(_TOOL_CALL_END),
        re.DOTALL,
    )

    # 从请求的 chat_template_kwargs 中获取工具调用格式，若未指定则默认为 "xml"。
    @classmethod

```

```

def _request_tool_format(cls, request: ToolParserRequest) -> str:
    chat_template_kwargs = request.chat_template_kwargs or {}
    tool_format = chat_template_kwargs.get("tool_call_format", "xml")
    if (
        not isinstance(tool_format, str)
        or tool_format not in cls._SUPPORTED_TOOL_FORMATS
    ):
        supported = ", ".join(sorted(cls._SUPPORTED_TOOL_FORMATS))
        raise ValueError(
            f"Unsupported tool_call_format {tool_format!r}. "
            f"Supported values: {supported}."
        )
    return tool_format

# 解析 XML 格式的工具调用体。
@classmethod
def _parse_xml_call(
    cls,
    body: str,
    request: ToolParserRequest,
    tool_format: str,
) -> tuple[str, dict[str, Any]]:
    first_arg = body.find(cls._ARG_KEY_START)
    if first_arg == -1:
        if any(marker in body for marker in cls._ARG_MARKERS):
            raise ValueError("Malformed IFM XML argument tags.")
        name = cls._validate_tool_name(body.strip(), request)
        return name, {}

    name = cls._validate_tool_name(body[:first_arg].strip(), request)
    properties = find_tool_properties(request.tools, name)
    arguments: dict[str, Any] = {}
    position = first_arg
    for match in cls._ARG_RE.finditer(body, first_arg):
        if body[position : match.start()].strip():
            raise ValueError("Malformed IFM XML argument tags.")

        arg_name = match.group(1).strip()
        if not arg_name:
            raise ValueError("IFM XML argument is missing a name.")
        if arg_name in arguments:
            raise ValueError(f"Duplicate IFM XML argument: {arg_name}.")

        explicit_type = match.group(2)
        if explicit_type is not None:
            explicit_type = explicit_type.strip()
        if tool_format == "xml" and explicit_type is not None:
            raise ValueError("Unexpected type tag in IFM XML tool call.")
        if tool_format == "xml_typed" and not explicit_type:

```

```
        raise ValueError("IFM typed XML argument is missing a type.")
arguments[arg_name] = cls._coerce_xml_value(
    match.group(3),
    properties.get(arg_name),
    explicit_type,
)
position = match.end()

if position == first_arg or body[position:].strip():
    raise ValueError("Malformed IFM XML argument tags.")
return name, arguments
```

评论区精华

PR 的讨论集中在 CI 和代码格式上。[mergify\[bot\]](#) 提供了文档预览链接并提示 pre-commit 检查失败。[jeejeelee](#) 两次通过 `/ci run` 触发了 CI。作者 [tanyuqian](#) 根据 CI 反馈修复了 pre-commit linting 问题。没有出现关于核心架构设计或实现的技术讨论，主要是一次性的代码质量修复。

- CI 运行与代码格式修复 (testing): 通过多次提交修复了 pre-commit linting 问题，最终 CI 通过。

风险与影响

- 风险：主要风险在于新模型实现的正确性和性能。由于 `k2_horizon.py` 是一个全新的、复杂的模型文件（1445 行），其 MoE 路由、融合内核调用（`fused_mova_impl`）和分组归一化（`K2HorizonRMSNorm`）逻辑需要与 vLLM 现有的算子库（如 `dispatch_fused_moe_kernel`, `ops.rms_norm`）正确集成，可能存在张量形状或类型不匹配的回归风险。工具解析器的复杂正则表达式可能在处理极端或恶意生成的输出时存在性能或安全隐患（例如回溯爆炸）。
- 影响：对用户的影响是积极的，使 vLLM 能够原生支持运行 K2-Horizon 模型，扩展了 vLLM 支持的模型生态。对系统的影响范围有限，主要是新增文件，对现有模块的修改仅限于注册表和解析器索引，属于低风险变更。对团队的影响是增加了模型维护和测试的负担，需要持续关注该模型相关的 issue。
- 风险标记：新模型实现正确性风险，复杂正则表达式性能风险

关联脉络

- 暂无明显关联 PR