PR #18172 完整报告

sgl-project/sglang

[NPU]Support model Trinity-mini for Npu, accuracy 90%

合并时间：2026-05-08 01:58

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/18172

执行摘要

- 一句话：NPU 支持 Trinity-mini 模型，准确率 90%
- 推荐动作：该 PR 展示了如何针对特定硬件后端适配 MoE 模型，值得 NPU 和 MoE 相关开发者阅读。关键设计决策包括设备感知的导入策略、TopK 参数的动态组合、以及 NPU 算子的统一调用方式。建议在后续 PR 中补充 GPU 回归测试并抽象设备判断逻辑。

功能与动机

根据 PR 描述，此前 NPU 不支持 trinity mini 模型。该模型具有独特的路由函数（custom_routing_function），需要在 NPU 后端进行适配。

实现拆解

1. 设备感知模块导入：在 afmoe.py 中通过 is_npu() 判断设备，根据设备选择 fused_moe 实现：GPU 使用 Triton 版本，NPU 使用自定义 NPU 版本。
2. MoE 路由参数适配：在 AfmoeMoE.__init__ 中，为 NPU 设置 correction_bias = self.expert_bias，并根据 score_func 传递 scoring_func 参数到 TopK；针对 NPU 禁用 renormalize。
3. NPU TopK 统一重构：在 fused_topk_npu 中，将之前分离的 grouped topk、correction_bias+sigmoid、num_token_non_padded 等多个条件合并为一个 elif 分支，使用 npu_moe_gating_top_k 统一处理，同时处理 group_select_mode 和 norm_type 等参数。
4. 测试与配置：新增 test_ascend_trinity_mini.py 端到端测试，继承 GSM8K mixin 验证准确率不低于0.85；在test_ascend_utils.py中添加TRINITY_MINI_WEIGHTS_PATH常量。

关键文件：

- python/sglang/srt/models/afmoe.py（模块 模型层；类别 source；类型 core-logic；符号 AfmoeMLP, AfmoeMoE, get_attention_sliding_window_size）：核心模型文件，新增设备感知的 fused_moe 导入、调整 MoE 路由参数以适配 NPU。
- test/registered/ascend/llm_models/test_ascend_trinity_mini.py（模块 测试；类别 test；类型 test-coverage；符号 TestTrinityMini）：新增的端到端测试，验证 Trinity-mini 在 NPU 上 GSM8K 准确率不低于 0.85。
- python/sglang/srt/hardware_backend/npu/moe/topk.py（模块 TopK 算子；类别 source；类型 core-logic；符号 fused_topk_npu）：NPU TopK 核心实现，重构分支逻辑使得支

持 grouped topk、correction_bias、sigmoid 等多种场景。

- python/sglang/test/ascend/test_ascend_utils.py（模块 测试工具；类别 test；类型 test-coverage）：添加 Trinity-mini 模型权重路径常量，供测试使用。

关键符号：fused_topk_npu, AfmoeMoE.init, AfmoeMoE.forward, AfmoeMoE.pack_params

关键源码片段

python/sglang/srt/models/afmoe.py

核心模型文件，新增设备感知的 fused_moe 导入、调整 MoE 路由参数以适配 NPU。

```
# python/sglang/srt/models/afmoe.py ( 关键变更片段 )

from sglang.srt.utils import add_prefix, is_npu

# 缓存设备判断结果
_is_npu = is_npu()

# 根据设备选择 fused_moe 后端
if not _is_npu:
    from sglang.srt.layers.moe.fused_moe_triton import fused_moe
else:
    from sglang.srt.hardware_backend.npu.quantization.fused_moe_method_npu import (
        fused_moe_npu as fused_moe,
    )

class AfmoeMoE(nn.Module):
    def __init__(self, config, prefix=""):
        # ... ( 省略相同部分 ) ...

        # 为 NPU 将 correction_bias 默认设置为 expert_bias
        correction_bias = None if not _is_npu else self.expert_bias
        if self.use_grouped_topk:
            correction_bias = self.expert_bias
        elif self.score_func == "sigmoid":
            custom_routing_fn = functools.partial(...)

        # NPU 上禁用 renormalize
        renormalize = (
            self.route_norm if self.score_func == "sigmoid" and not _is_npu else False
        )

        # 构建 TopK 对象，NPU 额外传递 scoring_func
        self.topk = TopK(
            top_k=self.top_k,
            renormalize=renormalize,
            use_grouped_topk=self.use_grouped_topk,
            # ... 其他原样参数 ...
```

```
            correction_bias=correction_bias,
            routed_scaling_factor=self.route_scale,
            **({"scoring_func": self.score_func} if _is_npu else {}),
        )

    def forward(self, hidden_states):
        router_logits, _ = self.gate(hidden_states)
        topk_output = self.topk(hidden_states, router_logits)
        # 直接调用 fused_moe（已按设备绑定对应实现）
        final_hidden_states = fused_moe(
            hidden_states,
            w1=self.w1,
            w2=self.w2,
            topk_output=topk_output,
            moe_runner_config=MoeRunnerConfig(inplace=True, routed_scaling_factor=self.route_
            scale),
        )
        return final_hidden_states
```

python/sglang/srt/hardware_backend/npu/moe/topk.py

NPU TopK 核心实现，重构分支逻辑使得支持 grouped topk、correction_bias、sigmoid 等多种场景。

```
# python/sglang/srt/hardware_backend/npu/moe/topk.py ( 关键变更片段 )

def fused_topk_npu(
    hidden_states: torch.Tensor,
    router_logits: torch.Tensor,
    topk_config: "TopKConfig",
    num_token_non_padded: Optional[torch.Tensor] = None,
    expert_location_dispatch_info: Optional["ExpertLocationDispatchInfo"] = None,
    layer_id: Optional[int] = None,
) -> "TopKOutput":
    use_grouped_topk = topk_config.use_grouped_topk
    renormalize = topk_config.renormalize
    correction_bias = topk_config.correction_bias

    # 快速路径：简单 top-k，无 grouped、无 bias
    if not use_grouped_topk and correction_bias is None:
        topk_weights, topk_ids, _ = torch.ops.npu.npu_moe_gating_top_k_softmax(
            router_logits, k=topk_config.top_k,
        )
        if renormalize:
            topk_weights = l1_norm(topk_weights)
        topk_weights = topk_weights.to(torch.float32)

    # 统一处理：correction_bias / sigmoid / num_token_non_padded / grouped topk
    elif (
        correction_bias is not None
```

```
        or topk_config.scoring_func == "sigmoid"
        or num_token_non_padded is not None
    ):
        # 调用 npu_moe_gating_top_k 算子，根据是否分组设置参数
        topk_weights, topk_ids, _ = torch.ops.npu.npu_moe_gating_top_k(
            router_logits.to(torch.float32),
            k=topk_config.top_k,
            bias=correction_bias.to(torch.float32) if correction_bias is not None else None,
            k_group=topk_config.topk_group if use_grouped_topk else 1,
            group_count=topk_config.num_expert_group if use_grouped_topk else 1,
            group_select_mode=(1 if use_grouped_topk else 0),
            renorm=0,
            norm_type=1, # 1 = sigmoid, 0 = softmax
            routed_scaling_factor=(1 if renormalize else topk_config.routed_scaling_factor),
            eps=1e-20,
        )
    else:
        # 兜底：退回 torch 原生
        topk_config.torch_native = True
        return select_experts(
            hidden_states=hidden_states,
            layer_id=layer_id,
            router_logits=router_logits,
            topk_config=topk_config,
            num_token_non_padded=num_token_non_padded,
            expert_location_dispatch_info=expert_location_dispatch_info,
        )

    # 物理映射与记录
    if expert_location_dispatch_info is not None:
        topk_ids = topk_ids_logical_to_physical(topk_ids, expert_location_dispatch_info)
    get_global_expert_distribution_recorder().on_select_experts(topk_ids=topk_ids)
    return StandardTopKOutput(topk_weights, topk_ids, router_logits)
```

评论区精华

Review 中主要讨论了以下要点：

- Gemini 检测到 envs."FORWARD_NATIVE_TOPK" 语法错误（critical），作者修复为 envs.FORWARD_NATIVE_TOPK。
- moe_forward_native.py 中 not is_npu 条件因 is_npu 是函数对象导致永远为 False，作者后改为使用 _is_npu。
- 评论者 ping1jing2 对多个实现细节提出疑问，包括 custom_routing_function 变量的必要性、magic number 1e-2 的含义、import 方式简化建议；作者均回复采纳或解释。
- iforgetmyname 建议回退到默认 topk 设置，该模型无性能需求；作者采纳。
- 针对 rope_theta 和 rope_scaling 的变更，作者说明是 bugfix（适配 transformers 5.3.0）。

- envs.FORWARD_NATIVE_TOPK 语法错误 (correctness): 作者后续修复，改为 envs.FORWARD_NATIVE_TOPK.get() 并在 Envs 中添加该变量。
- not is_npu 条件永远为 False (correctness): 作者修正，使用模块级变量 _is_npu。
- 回退默认 topk 设置 (design): 作者采纳，将 topk 设置为默认值。
- rope 配置变更是否为 bugfix (correctness): 作者说明是 bugfix，适配 transformers 5.3.0 的接口变化。
- import 方式简化建议 (design): 作者未明确回复，但最终代码采用了 if not _is_npu: ... else: ... 的结构。

风险与影响

- 风险:
  1. 回归风险（高）：afmoe.py 中修改了 AfmoeMoE.__init__ 的 TopK 参数和 forward 中的 fused_moe 调用，可能影响 GPU 上原有 afmoe 模型的正确性。当前 PR 未添加针对 GPU 的回归测试。
  2. NPU 专属路径维护成本：新增的 _is_npu 条件分支使得代码中出现多处设备判断，后续维护需同时考虑 GPU 和 NPU 行为差异。
  3. 数值精度风险：topk.py 中移除了标准化（renormalize）路径，改为在 NPU kernel 中处理，若 NPU kernel 的 softmax/sigmoid 实现存在精度差异，可能导致准确率下降。
  4. 配置兼容性：TRINITY_MINI_WEIGHTS_PATH 常量强依赖特定目录结构，若权重路径变更，测试会失败。 - 影响：用户影响：NPU 用户现在可以运行 arcee-ai/Trinity-Mini 模型，并达到 90% GSM8K 准确率；GPU 用户无影响。系统影响：增加了约 8KB 代码，主要条件分支在初始化时确定，运行时无额外开销。测试影响：新增 nightly 级别的端到端测试，依赖 2 卡 NPU A3 实例，会占用 CI 资源。
- 风险标记：NPU 条件分支，GPU 回归测试缺失，TopK 重构覆盖面

关联脉络

- 暂无明显关联 PR