PR #26402 完整报告

sgl-project/sglang

[5/N] Quantization Refactor: GPTQ schemes and kernel split

合并时间：2026-05-28 22:15

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/26402

执行摘要

- 一句话：GPTQ 量化重构：按 scheme/kernel 拆分为独立模块
- 推荐动作：值得精读，尤其是 scheme/kernel 分离的设计模式，以及如何通过工厂方法统一不同后端的量化逻辑。对于从事量化或硬件抽象层的工程师，这是一个很好的参考案例。PR 讨论中关于移除 is_xxx 检查的要点也体现了架构整洁性追求。

功能与动机

PR 描述指出遵循 AWQ 已使用的 scheme/kernel split 模式，使 GPTQ 代码结构更清晰，便于扩展新硬件后端和新量化方案。同时移除散布的平台判断逻辑，集中管理，降低耦合。

实现拆解

1. 创建 gptq 包与 schemes 子包：新建 python/sglang/srt/layers/quantization/gptq/ 目录（含 __init__.py 和 gptq.py）及 schemes/ 子目录，将原 gptq.py 中的配置、线性方案、Marlin 方案、MoE 方案拆解到独立文件。
2. 拆分线性层方案：gptq_linear.py 中定义 GPTQLinearScheme（GPU）和 GPTQAscendLinearScheme（NPU），通过 create_weights、process_weights_after_loading、apply_weights 统一接口，内部调用各自的 kernel 实现。
3. 拆分 Marlin 与 MoE 方案：gptq_marlin.py 定义 GPTQMarlinLinearScheme，gptq_moe.py 定义 GPTQMarlinMoEScheme 和 GPTQMoEAscendScheme，将 Marlin 相关的 repack 和 permute 逻辑独立。
4. 迁移 kernel 实现到硬件后端：GPU 的 GPTQLinearKernel、GPTQMarlinLinearKernel 等移入 hardware_backend/gpu/quantization/gptq_kernels.py；NPU 的 GPTQLinearAscendKernel、GPTQMoEAscendKernel 等移入 hardware_backend/npu/quantization/gptq_kernels.py。保留 gptq_marlin_moe_repack 工具函数。
5. 更新注册与删除旧文件：修改 layers/quantization/__init__.py 以指向新包，调整 auto_round.py 配置引用。删除原 gptq.py（1571 行），并通过 get_linear_quant_method 确保 embedding 层不被错误分配线性量化方法。

关键文件：

- python/sglang/srt/layers/quantization/gptq.py（模块 量化核心；类别 source；类型 deletion；符号 check_marlin_format, gptq_marlin_moe_repack,

MarlinLinearLayerConfig, GPTQConfig）：原 GPTQ 量化核心文件，所有逻辑被拆分解除，是本次重构的主要改造目标

- python/sglang/srt/layers/quantization/gptq/gptq.py（模块 GPTQ 配置；类别 source；类型 dependency-wiring；符号 check_marlin_format, GPTQConfig, init, repr）：新的 GPTQ 配置与管理入口，整合 schemes
- python/sglang/srt/hardware_backend/gpu/quantization/gptq_kernels.py（模块 GPU 内核；类别 source；类型 dependency-wiring；符号 _unsupported_kernel, MarlinLinearLayerConfig, gptq_marlin_moe_repack, GPTQLinearKernel）：GPU 上 GPTQ kernel 的独立实现，包括 GPTQLinearKernel 和 GPTQMarlinLinearKernel
- python/sglang/srt/hardware_backend/npu/quantization/gptq_kernels.py（模块 NPU 内核；类别 source；类型 core-logic；符号 unpack_from_int32, GPTQLinearAscendKernel, init, process_weights_after_loading）：NPU 上 GPTQ kernel 的独立实现，包括 GPTQLinearAscendKernel 和 GPTQMoEAscendKernel
- python/sglang/srt/layers/quantization/gptq/schemes/gptq_moe.py（模块 MoE 方案；类别 source；类型 core-logic；符号 GPTQMoEAscendScheme, init, create_weights, create_moe_runner）：MoE 方案独立模块，包含 GPU Marlin MoE 和 NPU MoE 方案
- python/sglang/srt/layers/quantization/gptq/schemes/gptq_linear.py（模块 线性方案；类别 source；类型 dependency-wiring；符号 GPTQLinearScheme, init, _init_kernel, create_weights）：线性层方案独立模块，包含 GPU 和 NPU 线性方案
- python/sglang/srt/layers/quantization/gptq/schemes/gptq_marlin.py（模块 Marlin 方案；类别 source；类型 core-logic；符号 GPTQMarlinLinearScheme, init, create_weights, process_weights_after_loading）：Marlin 线性方案独立模块
- python/sglang/srt/layers/quantization/gptq/schemes/gptq_scheme.py（模块 方案基类；类别 source；类型 dependency-wiring；符号 GPTQLinearSchemeBase, create_weights, process_weights_after_loading, apply_weights）：定义 Scheme 基类 GPTQLinearSchemeBase 和 GPTQMoESchemeBase
- python/sglang/srt/layers/quantization/gptq/__init__.py（模块 包初始化；类别 source；类型 dependency-wiring）：gptq 包的初始化，导出核心类
- python/sglang/srt/layers/quantization/gptq/schemes/__init__.py（模块 方案初始化；类别 source；类型 dependency-wiring）：scheme 子包的初始化，导出所有方案类
- python/sglang/srt/layers/quantization/__init__.py（模块 量化入口；类别 source；类型 dependency-wiring）：上层导入更新，指向新的 gptq 包
- python/sglang/srt/layers/quantization/auto_round.py（模块 AutoRound；类别 source；类型 core-logic）：AutoRound 配置引用调整，适配新 GPTQ 包结构

关键符号：check_marlin_format, gptq_marlin_moe_repack, MarlinLinearLayerConfig, GPTQConfig, GPTQLinearKernel, GPTQMarlinLinearKernel, GPTQLinearAscendKernel, GPTQMoEAscendKernel, unpack_from_int32, GPTQLinearScheme, GPTQAscendLinearScheme, GPTQMarlinLinearScheme, GPTQMarlinMoEScheme, GPTQMoEAscendScheme, GPTQLinearSchemeBase, GPTQMoESchemeBase

关键源码片段

python/sglang/srt/hardware_backend/npu/quantization/gptq_kernels.py

NPU 上 GPTQ kernel 的独立实现，包括 GPTQLinearAscendKernel 和 GPTQMoEAscendKernel

```
from __future__ import annotations

from typing import TYPE_CHECKING, Optional

import torch
import torch_npu

# NPU fused experts 实现
from sglang.srt.hardware_backend.npu.quantization.fused_moe_method_npu import (
    npu_fused_experts,
)

def unpack_from_int32(weight: torch.Tensor, num_bits: int, packed_dim: int = 1) -> torch.
Tensor:
    """
    将int32格式的量化权重解包回原始位数。
    支持沿dim=0或dim=1解包，返回int8类型（以零点为中心）。
    """
    assert weight.dtype == torch.int32
    pack_factor = 32 // num_bits
    mask = (1 << num_bits) - 1

    if packed_dim == 1:
        unpacked_weight = torch.zeros(
            (weight.shape[0], weight.shape[1] * pack_factor),
            device=weight.device, dtype=torch.int32,
        )
        for i in range(pack_factor):
            unpacked_weight[:, i::pack_factor] = (weight >> (num_bits * i)) & mask
    else:
        unpacked_weight = torch.zeros(
            (weight.shape[0] * pack_factor, weight.shape[1]),
            device=weight.device, dtype=torch.int32,
        )
        for i in range(pack_factor):
            unpacked_weight[i::pack_factor, :] = (weight >> (num_bits * i)) & mask
    offset = pow(2, num_bits) // 2
    unpacked_weight = (unpacked_weight - offset).to(torch.int8)
    return unpacked_weight

class GPTQLinearAscendKernel:
    """NPU上的GPTQ线性层内核，使用torch_npu定制算子。"""
    def __init__(self, quant_config: Optional["QuantizationConfig"] = None):
        self.quant_config = quant_config
```

```
        self.use_v2_format = quant_config.checkpoint_format == "gptq_v2"

    def process_weights_after_loading(self, layer: torch.nn.Module) -> None:
        # 解包 qzeros（沿 packed_dim=1）并调整数据类型
        layer.qzeros = torch.nn.Parameter(
            unpack_from_int32(
                layer.qzeros.data.contiguous(),
                self.quant_config.weight_bits,
                packed_dim=1,
            ).to(layer.scales.dtype),
            requires_grad=False,
        )
        if not self.use_v2_format:
            layer.qzeros += 1 # GPTQ v1 需要偏移

        qweight_tmp = unpack_from_int32(
            layer.qweight.data.contiguous(), self.quant_config.weight_bits, packed_dim=0
        )
        if self.quant_config.weight_bits != 4:
            # 非 4bit 时直接存储 int8
            layer.qweight = torch.nn.Parameter(qweight_tmp, requires_grad=False)
            return

        # 4bit 时使用 NPU 特定 int4pack 格式以节省内存
        layer.qweight = torch.nn.Parameter(
            torch_npu.npu_convert_weight_to_int4pack(qweight_tmp.to(torch.int32)),
            requires_grad=False,
        )

    def apply(self, layer, x, bias=None):
        qweight = layer.qweight
        scales = layer.scales
        qzeros = layer.qzeros
        reshaped_x = x.reshape(-1, x.shape[-1])

        if bias is not None and bias.dtype == torch.bfloat16:
            bias = bias.float()

        out_shape = x.shape[:-1] + (qweight.shape[-1] * 8,)
        out = torch_npu.npu_weight_quant_batchmatmul(
            reshaped_x, qweight,
            antiquant_scale=scales,
            antiquant_offset=qzeros,
            antiquant_group_size=self.quant_config.group_size,
            bias=bias,
        )
        return out.reshape(out_shape)
```

评论区精华

review 中 ping1jing2 指出重构后不应保留 is_xxx 平台检查，Alisehen 同意并移除了散布的 is_cuda/is_npu 检查，将平台选择集中到量化注册表中。这一讨论体现了对架构清晰度的追求。

- 去除 is_xxx 平台检查 (design): 同意移除散布的平台检查，集中到注册表

风险与影响

- 风险：
  - 回归风险：重构涉及大量文件移动和拆分，可能引入新 bug，特别是 Marlin repack 和 MoE 权重加载路径。CI 已通过 GPU 和 AMD 测试，但 NPU 路径仅编译检查，缺少端到端推理验证。
  - 性能影响：重构未改变计算逻辑，但间接层可能引入微小开销（如额外的方法调用）。理论上不影响性能。
  - 可维护性：增加类层次结构和模块数量，新开发者需要时间熟悉。但采用与 AWQ 一致的设计模式，降低了长期认知负载。
  - 测试覆盖：未新增单元测试，依赖现有的集成测试和手动验证。对于这种大规模重构，风险较高。
- 影响：
  - 用户：无功能变化，GPTQ 量化模型加载和推理行为一致。--quantization gptq_marlin 等选项继续生效。
  - 系统：不影响其他量化方法（AWQ、FP8 等）。
  - 团队：新架构更易扩展新硬件后端（如 XPU、CPU），降低后续添加新量化方案的代码冲突。但程序员需要适应新的文件布局。
  - 风险标记：核心路径变更，缺少测试覆盖，多层继承复杂度

关联脉络

- 暂无明显关联 PR