PR #21126 完整报告

sgl-project/sglang

[4/N] Quantization Refactor: AWQ schemes and Kernel call and weight init split

合并时间：2026-04-30 19:51

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/21126

执行摘要

- 一句话：重构 AWQ 量化模块，拆解为 scheme 结构并分离后端内核
- 推荐动作：值得精读。该 PR 展示了如何将庞大历史遗留模块拆解为 scheme + kernel 的干净架构，其设计思路可以借鉴到其他量化方法（如 GPTQ、FP8）甚至非量化的模型层。重点关注 get_quant_method 的分派逻辑、_init_kernel 钩子模式以及 hardware_backend 的隔离策略。

功能与动机

根据 Roadmap Issue #15194，目标是将量化方法统一为 scheme 结构，分离权重加载与推理逻辑，降低单个文件的复杂度。PR body 明确指出："Refactored AWQ to align with the scheme-based quantization structure used by modelslim and compressed_tensors"，并期望通过 scheme 结构使得不同框架可以共用同一套内核，避免 get_quant_method 无限膨胀。

实现拆解

1. 创建 scheme 新包：在 quantization/awq/ 下新建 __init__.py、awq.py 和 schemes/ 子包。核心的 AWQConfig 从旧 awq.py 迁移至新 awq/awq.py，并新增 get_linear_scheme 和 get_moe_scheme 方法用于分派具体的 scheme。
2. 定义 Scheme 基类与具体实现：在 schemes/awq_scheme.py 中定义 AWQLinearSchemeBase 和 AWQMoESchemeBase；awq_linear.py 和 awq_moe.py 分别实现 Linear 和 MoE 的 scheme，包含 create_weights、process_weights_after_loading、apply_weights 等标准方法，同时提供 Ascend 平台的派生类避免初始化 GPU 内核。
3. 分离后端内核逻辑：将 GPU（CUDA/HIP）的 dequantize、Marlin repack 等内核调用移至 hardware_backend/gpu/quantization/awq_kernels.py，将 NPU Ascend 的布局转换和 batch matmul 封装移至 hardware_backend/npu/quantization/awq_kernels.py，二者统一通过 process_weights_after_loading 和 apply 接口暴露。
4. 调整入口与清理旧文件：修改 quantization/__init__.py 的导入指向新包；删除旧 awq.py（-966 行）、将 awq_cpu.py 重命名为 schemes/awq_cpu.py 并精简为 scheme 形式；清理 WEIGHT_LOADER_V2_SUPPORTED 中已移除的 AWQLinearAscendMethod。

关键文件：

- python/sglang/srt/layers/quantization/awq.py（模块 量化模块；类别 source；类型 deletion；符号 is_layer_skipped_awq, AWQConfig, init, repr）：原 AWQ 量化主干文件

，被完全删除（-966 行）。其全部逻辑迁移至新包，是重构的收缩起点。

- python/sglang/srt/layers/quantization/awq/awq.py（模块 量化模块；类别 source；类型 dependency-wiring；符号 is_layer_skipped_awq, AWQConfig, init, repr）：新 AWQ 方案的核心模块，包含 AWQConfig、get_quant_method、get_linear_scheme、get_moe_scheme 等入口，负责将线性层和 MoE 层分派到具体 scheme。
- python/sglang/srt/hardware_backend/gpu/quantization/awq_kernels.py（模块 GPU 内核；类别 source；类型 dependency-wiring；符号 _unsupported_awq_dequantize, AWQLinearKernel, init, process_weights_after_loading）：GPU 后端内核封装，包含 AWQLinearKernel、AWQMarlinLinearKernel、AWQMoEKernel，统一通过 process_weights_after_loading 和 apply 接口提供服务，原 awq.py 中的 CUDA/HIP 内核代码迁移至此。
- python/sglang/srt/hardware_backend/npu/quantization/awq_kernels.py（模块 NPU 内核；类别 source；类型 core-logic；符号 AWQAscendLinearKernel, init, process_weights_after_loading, apply）：NPU Ascend 后后端内核封装，包含 AWQAscendLinearKernel 和 AWQAscendMoEKernel，处理 AWQ 权重到 NPU 布局的转换并调用 torch_npu 原生算子执行推理。
- python/sglang/srt/layers/quantization/awq/schemes/awq_linear.py（模块 量化方案；类别 source；类型 dependency-wiring；符号 AWQLinearScheme, init, _init_kernel, create_weights）：AWQ 线性层 scheme 实现，包括权重创建、加载后处理、前向调用，统称为 scheme 架构的核心单元。提供了 AWQLinearScheme 和 AWQAscendLinearScheme 两个变体。
- python/sglang/srt/layers/quantization/awq/schemes/awq_moe.py（模块 量化方案；类别 source；类型 dependency-wiring；符号 AWQMoEScheme, init, _init_kernel, create_weights）：AWQ MoE 层 scheme 实现，类似线性层 scheme，处理专家权重的创建、后处理和前向，包括 Ascend 变体。
- python/sglang/srt/layers/quantization/awq/schemes/awq_scheme.py（模块 量化方案；类别 source；类型 dependency-wiring；符号 AWQLinearSchemeBase, create_weights, process_weights_after_loading, apply_weights）：定义 AWQLinearSchemeBase 和 AWQMoESchemeBase 抽象基类，约定 create_weights、process_weights_after_loading、apply_weights 等接口，是 scheme 结构的契约。
- python/sglang/srt/layers/quantization/__init__.py（模块 量化模块；类别 source；类型 dependency-wiring）：量化包入口，更新导入路径以指向新 awq 子包，影响外部引用。

关键符号：AWQConfig.init, AWQConfig.get_quant_method, AWQConfig.get_linear_scheme, AWQConfig.get_moe_scheme, AWQLinearScheme._init_kernel, AWQLinearScheme.create_weights, AWQLinearScheme.process_weights_after_loading, AWQLinearScheme.apply_weights, AWQMarlinLinearScheme.process_weights_after_loading, AWQMoEScheme.create_weights, AWQMoEScheme.apply_weights, AWQLinearKernel.apply, AWQMarlinLinearKernel.process_weights_after_loading, AWQAscendLinearKernel.process_weights_after_loading, AWQAscendLinearKernel.apply, AWQAscendMoEKernel._convert_awq_weight_to_npu_lay

out, AWQAscendMoEKernel._convert_awq_qzeros_to_npu_offset, is_layer_skipped_awq

关键源码片段

python/sglang/srt/hardware_backend/gpu/quantization/awq_kernels.py

GPU 后端内核封装，包含 AWQLinearKernel、AWQMarlinLinearKernel、AWQMoEKernel，统一通过 process_weights_after_loading 和 apply 接口提供服务，原 awq.py 中的 CUDA/HIP 内核代码迁移至此。

```
class AWQLinearKernel:
    """GPU AWQ 线性层内核：封装反量化 + 矩阵乘"""
    def __init__(self, quant_config: Optional["QuantizationConfig"] = None):
        self.quant_config = quant_config

    def process_weights_after_loading(self, layer: torch.nn.Module) -> None:
        # 将权重参数标记为不需要梯度
        layer.qweight = torch.nn.Parameter(layer.qweight.data, requires_grad=False)
        layer.qzeros = torch.nn.Parameter(layer.qzeros.data, requires_grad=False)
        layer.scales = torch.nn.Parameter(layer.scales.data, requires_grad=False)

    def apply(
        self,
        layer: torch.nn.Module,
        x: torch.Tensor,
        bias: Optional[torch.Tensor] = None,
    ) -> torch.Tensor:
        qweight = layer.qweight
        scales = layer.scales
        qzeros = layer.qzeros
        pack_factor = self.quant_config.pack_factor
        out_shape = x.shape[:-1] + (qweight.shape[-1] * pack_factor,)
        reshaped_x = x.reshape(-1, x.shape[-1])
        # 先反量化 (dequantize)，再执行矩阵乘
        out = awq_dequantize(qweight, scales, qzeros)
        out = torch.matmul(reshaped_x, out)
        if bias is not None:
            out.add_(bias)
        return out.reshape(out_shape)
```

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

NPU Ascend 后后端内核封装，包含 AWQAscendLinearKernel 和 AWQAscendMoEKernel，处理 AWQ 权重到 NPU 布局的转换并调用 torch_npu 原生算子执行推理。

```
class AWQAscendLinearKernel:
    """NPU Ascend AWQ 线性层内核：将 AWQ 权重转换为 NPU 布局并使用 batch matmul"""
    def __init__(self, quant_config: Optional["QuantizationConfig"] = None):
        self.quant_config = quant_config

    def process_weights_after_loading(self, layer: torch.nn.Module) -> None:
```

```
        layer.scales = torch.nn.Parameter(layer.scales.data, requires_grad=False)
        qweight_tmp = torch.zeros_like(layer.qweight.data)
        qzeros_tmp = layer.qzeros.data
        qzeros_list = []
        shifts = [0, 4, 1, 5, 2, 6, 3, 7] # NPU 特殊 nibble 顺序
        for i in range(0, self.quant_config.pack_factor):
            shift_num = shifts[i] * 4
            qzeros_list.append((qzeros_tmp.reshape(-1, 1) >> shift_num) & 0xF)
            qweight_tmp.bitwise_or_(
                ((layer.qweight.data >> shift_num) & 0xF) << (4 * i)
            )
        qweight_tmp.bitwise_xor_(0x88888888) # AWQ NPU 布局要求异或
        qzeros_tmp = torch.cat(qzeros_list, dim=-1).reshape(qzeros_tmp.shape[0], -1)
        qzeros_tmp = -(qzeros_tmp - 8) # 将 zero-point 转换为 offset
        qzeros_tmp = qzeros_tmp.to(layer.scales.data.dtype)
        layer.zeros = torch.nn.Parameter(qzeros_tmp, requires_grad=False)
        layer.weight = torch.nn.Parameter(qweight_tmp, requires_grad=False)

    def apply(self, layer, x, bias=None):
        # 使用 NPU 原生的批量矩阵乘算子
        out = torch_npu.npu_weight_quant_batchmatmul(
            x.reshape(-1, x.shape[-1]),
            layer.weight,
            antiquant_scale=layer.scales,
            antiquant_offset=layer.zeros,
            antiquant_group_size=self.quant_config.group_size,
            bias=bias,
        )
        return out.reshape(x.shape[:-1] + (layer.weight.shape[-1] * self.quant_config.pack_factor,)
        )
```

评论区精华

- b8zhong 质疑 GPU 内核移入 hardware_backend 的合理性，TamirBaydasov 解释这是与 @rainj-me 讨论后的决策，目的是清空 quantization 文件夹，使导航更清晰，后续可能将更多 GPU 内核统一迁入。最终保留当前结构，但 awq_kernels.py 中仍存在少量 is_xxx 判断， reviewer 建议今后进一步消除。
- gemini-code-assist[bot] 指出关键 bug：AWQMoEKernel 中 w13_scales 的 permute 参数 size_k 错误使用了 intermediate_size_per_partition 而非 hidden_size，可能导致 MoE 权重加载后结果错误。作者后续修复了该问题。
- alexnails 提出 XPU 回归：重构后 XPU 的 dequantize 路径因导入顺序变化可能退化到 Triton 分支而非使用 sgl_kernel。作者及时恢复 XPU 优先导入 sgl_kernel.awq_dequantize 修复。
- alexnails 指出 AWQAscendMoEScheme 不必要的 GPU 初始化开销：因该 scheme 继承自 AWQMoEScheme，父类 __init__ 会导入 AWQMoEKernel（含 marlin 依赖）。作者通过添加 _init_kernel 钩子使 Ascend 子类跳过父类初始化，直接注入 AMD Ascend 内核。

- OrangeRedeng 建议 awq_w4a16.py 重命名为 awq_linear.py，作者接受并更新，使文件职责更清晰。
- GPU 内核是否应移入 hardware_backend (design): 保留当前结构，但 awq_kernels.py 中仍存在 is_xxx 判断，reviewer 要求后续逐步消除。
- AWQMoEKernel w13_scales 的 size_k 参数错误 (correctness): 作者确认并修复为 layer.hidden_size。
- XPU AWQ dequantize 路径回归 (bugfix): 作者修复导入顺序，XPU 优先尝试 sgl_kernel.awq_dequantize，再 fallback 到 Triton。
- AWQAscendMoEScheme 不必要的 GPU 初始化 (performance): 作者添加 _init_kernel 钩子，使 Ascend 子类跳过父类初始化，直接创建 AWQAscendMoEKernel，避免不必要的 GPU 导入。
- awq_w4a16.py 重命名建议 (style): 作者接受并重命名为 awq_linear.py，同时更新 schemes/__init__.py 中的导入。

风险与影响

- 风险：
  1. GPU 回归风险：Marlin MoE 的 size_k 参数错误在 review 中被修复，但可能还有其他未发现的参数传递问题，尤其是 marlin_moe_permute_scales 和 marlin_permute_scales 的调用。
  2. XPU 兼容风险：dequantize 路径依赖 sgl_kernel，若导入顺序错误会降级到 Triton，虽然已修复但缺乏自动化测试覆盖。
  3. NPU 布局转换正确性：AWQAscendLinearKernel 中的位操作和 xor 0x88888888 等细节高度平台相关，若有误解可能导致精度下降。
  4. CPU AMX 路径简化不完整：awq_cpu.py 中 AWQIntelAMXLinearScheme 重写了 __init__ 但未调用 super().__init__，虽已改但脆弱。
  5. 导入循环依赖风险：新包结构引入了更多交叉引用，尤其是 scheme 和 backend kernel 之间的导入，审核需谨慎。 - 影响：对用户：功能完全透明，模型推理行为不变，用户无需升级配置。对系统：模块内聚性提升，新增量化方案（如其他 bit-width 或硬件后后端）时只需添加新的 scheme 类和 kernel 实现，无需修改 AWQConfig 主干。对团队：量化开发人员可以更快定位和扩展特定硬件路径，review 新量化方案时关注点从单一巨大文件缩小到 scheme 和 kernel 两个层次。影响范围：中等，波及 20 个文件，但核心逻辑保持等价。 - 风险标记：GPU 内核回归风险，XPU 兼容性风险，NPU 权重布局转换正确性，MoE 缩放参数 size_k 曾出错，CPU AMX 路径未完全清理

关联脉络

- PR #17503 Quantization Refactor: AWQ schemes and Kernel call and weight init split: 本 PR 的前一个版本或相关拆分，被 follow up 直接引用。
- PR #17361 MoE refactoring: 讨论中 TamirBaydasov 引用该 PR 作为 NPU MoE kernel 调用方式的参考。