PR #31109 完整报告

sgl-project/sglang

Remove QServe and FBGEMM FP8 quantization

合并时间：2026-07-18 08:10

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/31109

执行摘要

- 一句话：移除废弃的 QServe 和 FBGEMM FP8 量化路径
- 推荐动作：建议阅读此 PR 以了解如何系统性地移除废弃功能，包括清理注册、内核、测试和文档的完整步骤。对于量化路径的设计决策——聚焦主流方案、弃用低利用率路径——值得在类似场景中参考。

功能与动机

根据 Issue #28543 的讨论，QServe (QoQ) W4A8 量化在过去一年没有特性开发，FBGEMM FP8 仅在官方 Llama 405B FP8 检查点中使用且利用率低。团队决定移除这些实验性量化路径以简化代码库。PR 作者在评论中引用 issue 并确认原作者同意。

实现拆解

具体的实现步骤包括：

1. 删除 QServe (QoQ) W4A8 量化：移除 python/sglang/srt/layers/quantization/qoq.py 文件，其中包含 QoQConfig 配置类和 QoQLinearMethod 线性方法；从 quantization/__init__.py 中移除 "qoq" 的注册条目；从 model_config.py 中移除 "qoq" 的配置键映射；删除 sgl-kernel/python/sgl_kernel/gemm.py 中的 qserve_w4a8_per_chn_gemm 和 qserve_w4a8_per_group_gemm 包装函数；删除 sgl-kernel/csrc/gemm/ 下对应的 CUDA 算子实现文件。
2. 删除 FBGEMM FP8 量化：移除 python/sglang/srt/layers/quantization/fpgemm_fp8.py 文件（包含 FBGEMMFp8Config 和 FBGEMMFp8LinearMethod）；从 quantization/__init__.py 和 model_config.py 中移除相关注册和配置映射。
3. 移除配套测试和基准：删除 sgl-kernel/tests/test_qserve_w4a8_per_group_gemm.py、sgl-kernel/tests/test_qserve_w4a8_per_chn_gemm.py 和 sgl-kernel/benchmark/bench_qserve_w4a8_gemm.py。
4. 清理头文件声明：在 sgl-kernel/include/sgl_kernel_ops.h 中移除对应的算子声明。
5. 合并冲突处理：包含两次从 main 分支的合并提交以保持同步。整体清理共删除 2566 行代码，新增 1 行。

关键文件：

- python/sglang/srt/layers/quantization/qoq.py（模块 量化层；类别 source；类型 deletion；符号 QoQConfig, init, repr, get_supported_act_dtypes）：QServe W4A8 量化的核心实现，包含配置类 QoQConfig 和线性方法 QoQLinearMethod。删除该文件是本次 PR 的主要目标之一。
- python/sglang/srt/layers/quantization/fpgemm_fp8.py（模块 量化层；类别 source；类型 deletion；符号 FBGEMMFp8Config, init, get_name, get_supported_act_dtypes）：FBGEMM FP8 量化的核心实现，包含配置类 FBGEMMFp8Config 和线性方法 FBGEMMFp8LinearMethod。删除此文件是另一主要目标。
- sgl-kernel/python/sgl_kernel/gemm.py（模块 Kernel 接口；类别 source；类型 core-logic；符号 qserve_w4a8_per_chn_gemm, qserve_w4a8_per_group_gemm）：删除了 qserve_w4a8_per_chn_gemm 和 qserve_w4a8_per_group_gemm 两个 Python 包装函数，它们调用了底层 CUDA 算子。
- sgl-kernel/tests/test_qserve_w4a8_per_group_gemm.py（模块 GroupGEMM 测试；类别 test；类型 test-coverage；符号 convert_to_qserve_format, progressive_group_quantize_tensor, sym_quantize_tensor, torch_w4a8_per_group_gemm）：测试文件验证 per-group 量化的正确性。删除它反映了对应功能的移除。
- sgl-kernel/tests/test_qserve_w4a8_per_chn_gemm.py（模块 ChanGEMM 测试；类别 test；类型 test-coverage；符号 convert_to_qserve_format, asym_quantize_tensor, sym_quantize_tensor, torch_w4a8_per_chn_gemm）：测试文件验证 per-channel 量化的正确性。
- sgl-kernel/benchmark/bench_qserve_w4a8_gemm.py（模块 基准；类别 source；类型 deletion；符号 to_int8, benchmark, prepare_shapes）：基准测试文件，对比不同量化方案的性能。删除它属于清理范围。

关键符号：QoQConfig.init, QoQConfig.from_config, QoQLinearMethod.create_weights, FBGEMMFp8Config.init, FBGEMMFp8Config.from_config, FBGEMMFp8LinearMethod.create_weights, qserve_w4a8_per_chn_gemm, qserve_w4a8_per_group_gemm, convert_to_qserve_format, test_accuracy

关键源码片段

python/sglang/srt/layers/quantization/fpgemm_fp8.py

FBGEMM FP8 量化的核心实现，包含配置类 FBGEMMFp8Config 和线性方法 FBGEMMFp8LinearMethod。删除此文件是另一主要目标。

```
# 文件：fpgemm_fp8.py（已在本 PR 中删除）
# FBGEMM FP8 量化配置类，继承自 QuantizationConfig
class FBGEMMFp8Config(QuantizationConfig):
    """Config class for FBGEMM Fp8."""

    def __init__(self, ignore_list: list[str], input_scale_ub: float):
        super().__init__()
        self.ignore_list = ignore_list if ignore_list else []
        self.input_scale_ub = input_scale_ub
```

```
        # 对于缺乏 FP8 硬件支持的 GPU，可以使用 Marlin 进行 weight-only 量化
        self.use_marlin = False
        if _is_cuda:
            force_marlin = get_bool_env_var('SGLANG_FORCE_FP8_MARLIN')
            auto_enable = can_auto_enable_marlin_fp8()
            self.use_marlin = force_marlin or auto_enable

    @classmethod
    def get_name(cls) -> str:
        return 'fbgemm_fp8'

    @classmethod
    def get_supported_act_dtypes(cls) -> list[torch.dtype]:
        return [torch.bfloat16, torch.float16]

    @classmethod
    def get_min_capability(cls) -> int:
        return 80

    @classmethod
    def get_config_filenames(cls) -> list[str]:
        return []

    @classmethod
    def from_config(cls, config: dict[str, Any]) -> 'FBGEMMFp8Config':
        ignore_list = cls.get_from_keys(config, ['modules_to_not_convert'])
        input_scale_ub = cls.get_from_keys(config, ['activation_scale_ub'])
        return cls(ignore_list=ignore_list, input_scale_ub=input_scale_ub)

    def get_quant_method(self, layer: torch.nn.Module, prefix: str):
        if isinstance(layer, LinearBase):
            if is_layer_skipped(prefix=prefix, ignored_layers=self.ignore_list, fused_mapping=self.
            packed_modules_mapping):
                return UnquantizedLinearMethod()
            return FBGEMMFp8LinearMethod(self)
        return None
```

sgl-kernel/python/sgl_kernel/gemm.py

删除了 qserve_w4a8_per_chn_gemm 和 qserve_w4a8_per_group_gemm 两个 Python 包装函数，它们调用了底层 CUDA 算子。

```
# sgl-kernel/python/sgl_kernel/gemm.py ( 被删除的部分 )
# 这两个函数是对 CUDA 算子的纯 Python 包装，已被本 PR 删除

def qserve_w4a8_per_chn_gemm(
    in_feats: torch.Tensor,
    kernel: torch.Tensor,
    wscales: torch.Tensor,
    ascales: torch.Tensor,
```

```
    w_szs: torch.Tensor,
    a_ssums: torch.Tensor,
    out_feats: Optional[torch.Tensor] = None,
) -> torch.Tensor:
    if out_feats is None:
        # 只支持 float16 输出
        out_feats = torch.empty(
            (in_feats.shape[0], kernel.shape[0]),
            device=in_feats.device,
            dtype=torch.float16,
        )
    torch.ops.sgl_kernel.qserve_w4a8_per_chn_gemm.default(
        in_feats, kernel, wscales, ascales, w_szs, a_ssums, out_feats
    )
    return out_feats


def qserve_w4a8_per_group_gemm(
    in_feats: torch.Tensor,
    kernel: torch.Tensor,
    zeros: torch.Tensor,
    scales_i8: torch.Tensor,
    wscales: torch.Tensor,
    ascales: torch.Tensor,
    out_feats: Optional[torch.Tensor] = None,
) -> torch.Tensor:
    if out_feats is None:
        out_feats = torch.empty(
            (in_feats.shape[0], kernel.shape[0]),
            device=in_feats.device,
            dtype=torch.float16,
        )
    torch.ops.sgl_kernel.qserve_w4a8_per_group_gemm.default(
        in_feats, kernel, zeros, scales_i8, wscales, ascales, out_feats
    )
    return out_feats
```

评论区精华

在关联 Issue #28543 中，用户 yujiongzhang 询问移除原因，作者 b8zhong 回复：“There hasn't been any development in a year. And the author is OK with it.” 该 PR 本身没有 review 评论，但获得了 Fridge003 的核准。

- 移除 QServe 和 FBGEMM FP8 的原因 (question): 作者 b8zhong 回复：“Hello, there hasn't been any development in a year. And the author is OK with it.”

风险与影响

- 风险：主要风险包括：

- 用户迁移风险：如果现有用户使用了 QServe 或 FBGEMM FP8 量化权重，他们将无法加载模型，需要转换为其他支持的量化格式（如 ModelOpt FP8 或 compressed-tensors FP8）。
- 代码恢复难度：被删除的 CUDA 内核 (qserve_w4a8_per_chn_gemm 和 qserve_w4a8_per_group_gemm) 从仓库中完全移除，未来如需恢复需要从 git 历史中还原。
- 遗漏引用：可能还有文档或配置中的引用没有被清理，但 PR 通过 Mintlify 预览同步了文档更新，覆盖较为全面。
- 影响较小：由于这些量化路径是实验性的且极少被使用，总体风险可控。
- 影响：影响范围限制在少数使用了这两个实验性量化的用户。对于绝大多数用户没有影响。移除后代码库更简洁，减少了约 2566 行代码，降低了后续维护成本和 CI 时间。团队需要确保替代量化方案的文档指引清晰。
- 风险标记：用户依赖的量化路径被移除，被删除 CUDA 内核不可逆，可能需要更新文档说明替代方案，清理可能遗留少量引用

关联脉络

- PR #31094 Remove deprecated Mamba flags from doc, wrong FP8 GEMM docstrings and change Nemotron image to 0.5.15: 同一时期执行的废弃清理工作，旨在移除已弃用的特性，展现了仓库持续降低技术债务的方向。