PR #31220 完整报告

sgl-project/sglang

Qwen3.5-MoE: support modelopt_fp4 checkpoints that quantize attention (+ load baked FP8 KV scales)

合并时间：2026-07-31 05:30

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/31220

执行摘要

- 一句话：支持 modelopt_fp4 检查点量化 attention 并加载 FP8 KV scales
- 推荐动作：此 PR 值得精读，原因如下：
  1. 展示了如何从硬编码（基于 quant method 名称无条件覆盖）转向基于检查点配置的逐层决策（利用已有的 is_layer_excluded），这是一种更健壮的设计。
  2. WeightsMapper 的使用提供了一个优雅的参数名映射模式，可推广到其他模型。
  3. REPL 验证结果表明，即使改动看似简单，也需全面回归测试。该 PR 的测试覆盖了三个关键层面（排除逻辑、参数注册、名称映射），值得学习。

功能与动机

之前的 PR #18937 为支持 NVIDIA 的 Qwen3.5 NVFP4 检查点，硬编码了 attention 模块的 quant_config = None（忽略模型自身的 exclude_modules），导致任何量化 attention 的 modelopt_fp4 检查点无法正确加载 attention 层和 baked FP8 KV scales。此问题在 #18937 的 review 中已被指出（comment），但未被修复。本 PR 正是为纠正此问题，根据检查点的 exclude_modules 动态决定 attention 是否量化，并加载对应的 KV scales。

实现拆解

1. 移除 attention 模块的 quant_config 硬编码：在 Qwen3_5LinearDecoderLayer 和 Qwen3_5AttentionDecoderLayer 的 __init__中，删除将 quant_config 强制置 None 的三元表达式，直接传递 quant_config，使 attention 模块的量化与否完全由 ModelOptFp4Config.is_layer_excluded() 根据 checkpoint 的 exclude_modules 决定。
2. 将 quant_config 传入 RadixAttention：在 Qwen3_5AttentionDecoderLayer 中，将 quant_config 参数传递给 RadixAttention 构造器。RadixAttention 在 kv_cache_quant_algo 不为 None 时会注册 k_scale/v_scale 参数，从而为 baked FP8 KV scales 提供存储位置。
3. 定义 QWEN3_5_KV_SCALE_MAPPER：在 qwen3_5.py 中声明一个 WeightsMapper，将 ModelOpt 导出格式中的 .self_attn.k_proj.k_scale 映射为 .attn.k_scale（v 类似）。该映射是模块级常量，在 load_weights 开头调用 apply。
4. 在所有模型类的 load_weights 中应用映射：包括 Qwen3_5ForCausalLM、Qwen3_5MoeForCausalLM、Qwen3_5ForConditionalGeneration、

Qwen3_5MoeForConditionalGeneration。映射后 scale 名称不再包含 k_proj/v_proj，因此不会进入 qkv_proj 分片匹配逻辑，而是通过通用 weight_loader 分支直接加载到 RadixAttention 的 scale 参数上。

5. 补充 CPU 单元测试：新增 test/registered/unit/models/test_qwen3_5_modelopt_fp4.py，覆盖 is_layer_excluded 在两类 checkpoint 上的行为、RadixAttention 的 scale 参数注册条件（有无 quant_config 及 kv_cache_quant_algo）、以及 QWEN3_5_KV_SCALE_MAPPER 的映射正确性。

关键文件：

- python/sglang/srt/models/qwen3_5.py（模块 模型加载；类别 source；类型 core-logic；符号 QWEN3_5_KV_SCALE_MAPPER, Qwen3_5LinearDecoderLayer.init, Qwen3_5AttentionDecoderLayer.init, Qwen3_5ForCausalLM.load_weights）：核心实现文件，包含 attention 量化移除硬编码、传递 quant_config 给 RadixAttention、定义 KV scale 映射器以及应用映射到所有模型类的 load_weights。
- test/registered/unit/models/test_qwen3_5_modelopt_fp4.py（模块 单元测试；类别 test；类型 test-coverage；符号 TestModelOptFp4AttentionExclusion, test_moe_only_checkpoint_excludes_attention, test_uniform_w4a4_checkpoint_quantizes_attention, TestRadixAttentionKvScaleRegistration）：新增的 CPU 单元测试，覆盖 attention 排除逻辑、RadixAttention scale 参数注册和 WeightsMapper 映射，确保改动正确性且无回归。

关键符号：QWEN3_5_KV_SCALE_MAPPER (module constant), Qwen3_5LinearDecoderLayer.init, Qwen3_5AttentionDecoderLayer.init, Qwen3_5ForCausalLM.load_weights, Qwen3_5MoeForCausalLM.load_weights, Qwen3_5ForConditionalGeneration.load_weights, Qwen3_5MoeForConditionalGeneration.load_weights, ModelOptFp4Config.is_layer_excluded

关键源码片段

python/sglang/srt/models/qwen3_5.py

核心实现文件，包含 attention 量化移除硬编码、传递 quant_config 给 RadixAttention、定义 KV scale 映射器以及应用映射到所有模型类的 load_weights。

```
# 从 sglang.srt.models.utils 导入 WeightsMapper（新增）
from sglang.srt.models.utils import WeightsMapper

# 模块级常量：将 ModelOpt 检查点中的 self_attn KV scale 名称映射到 sglang RadixAttention 参数名
QWEN3_5_KV_SCALE_MAPPER = WeightsMapper(
    orig_to_new_substr={
        ".self_attn.k_proj.k_scale": ".attn.k_scale", # k_proj.k_scale -> attn.k_scale
        ".self_attn.v_proj.v_scale": ".attn.v_scale", # v_proj.v_scale -> attn.v_scale
    },
)
```

```
# 在 Qwen3_5ForCausalLM 的 load_weights 中应用（其他三个模型同理）
def load_weights(self, weights: Iterable[Tuple[str, torch.Tensor]]):
    # 在 stacked_params_mapping 处理之前应用映射，确保 scale 名称不会被 qkv_proj 匹配消费
    weights = QWEN3_5_KV_SCALE_MAPPER.apply(weights)
    stacked_params_mapping = [
        ("qkv_proj", "q_proj", "q"),
        # ...
    ]
    # 后续逻辑不变
```

test/registered/unit/models/test_qwen3_5_modelopt_fp4.py

新增的 CPU 单元测试，覆盖 attention 排除逻辑、RadixAttention scale 参数注册和 WeightsMapper 映射，确保改动正确性且无回归。

```
class TestModelOptFp4AttentionExclusion(CustomTestCase):
    """验证 ModelOptFp4Config.is_layer_excluded 根据 exclude_modules 正确决策"""

    def test_moe_only_checkpoint_excludes_attention(self):
        # NVIDIA 官方 MoE-only 检查点的 exclude_modules 包含 *self_attn* 和 lm_head
        cfg = ModelOptFp4Config(
            is_checkpoint_nvfp4_serialized=True,
            kv_cache_quant_algo="FP8",
            group_size=16,
            exclude_modules=["*self_attn*", "lm_head"],
        )
        # attention 层应被排除（返回 True）
        self.assertTrue(cfg.is_layer_excluded("model.layers.0.self_attn.qkv_proj"))
        self.assertTrue(cfg.is_layer_excluded("lm_head"))
        # MoE experts 不应被排除
        self.assertFalse(cfg.is_layer_excluded("model.layers.0.mlp.experts.3.gate_up_proj"))

    def test_uniform_w4a4_checkpoint_quantizes_attention(self):
        # 均匀量化 checkpoints 只排除 lm_head
        cfg = ModelOptFp4Config(
            is_checkpoint_nvfp4_serialized=True,
            kv_cache_quant_algo="FP8",
            group_size=16,
            exclude_modules=["lm_head"],
        )
        # attention 应被量化（返回 False）
        self.assertFalse(cfg.is_layer_excluded("model.layers.0.self_attn.qkv_proj"))
        self.assertFalse(cfg.is_layer_excluded("model.layers.0.linear_attn.in_proj_qkvz"))
        self.assertTrue(cfg.is_layer_excluded("lm_head"))

class TestRadixAttentionKvScaleRegistration(CustomTestCase):
    """验证 RadixAttention 在给定 quant_config 时是否注册 k_scale/v_scale"""

    def _make_attn(self, quant_config):
        return RadixAttention(
```

```
            num_heads=2, head_dim=8, scaling=1.0, num_kv_heads=2,
            layer_id=0, quant_config=quant_config, prefix="model.layers.0.attn",
        )

    def test_with_fp8_kv_quant_config_registers_scale_params(self):
        cfg = ModelOptFp4Config(
            is_checkpoint_nvfp4_serialized=True,
            kv_cache_quant_algo="FP8", # 启用 FP8 KV cache
            group_size=16, exclude_modules=[],
        )
        attn = self._make_attn(cfg)
        # 应生成 k_scale 和 v_scale 参数，初始值 -1.0（哨兵值）
        self.assertIsInstance(attn.k_scale, torch.nn.Parameter)
        self.assertIsInstance(attn.v_scale, torch.nn.Parameter)
        self.assertEqual(attn.k_scale.item(), -1.0)
        self.assertEqual(attn.v_scale.item(), -1.0)

    def test_without_quant_config_has_no_scale_params(self):
        attn = self._make_attn(None)
        self.assertIsNone(attn.k_scale)
        self.assertIsNone(attn.v_scale)
```

评论区精华

Review 中核心讨论集中在 KV scale 名称映射的实现方式上。

- trevor-m要求使用 WeightsMapper 或类似机制（如 hf_to_sglang_mapper），而不是自定义的 remap-and-load 辅助函数。他指出 "This kind of mapping should be handled in the model file. Usually via hf_to_sglang_mapper, packed_modules_mapping, or similar. Also look at how other models use maybe_remap_kv_scale_name()."
- vroomfondel采纳建议，将 bespoke helper 替换为声明式的 WeightsMapper 模块常量，在 load_weights 入口调用 apply。
- trevor-m后续要求移除代码中冗余的 AI 注释（“verbose AI comments”），vroomfondel 在最终提交中清理了注释。 未解决的疑虑：暂无。
- KV scale 名称映射的实现方式 (design): 采用了声明式的 WeightsMapper，在 load_weights 入口应用，去除了自定义 helper。
- 移除冗余 AI 注释 (style): vroomfondel 在最后一个提交中清理了注释。

风险与影响

- 风险：
  - 兼容性风险：改动移除了 quant_config 的硬编码，理论上可能影响 NVIDIA 官方 MoE-only 检查点的行为，但 PR 作者和验证者（janbernloehr）已在 DGX Spark 上验证了 nvidia/Qwen3.6-35B-A3B-NVFP4 正常工作，未出现回归。
  - 性能风险：无直接性能影响，改动仅在模型初始化和加载时发生，推理路径不变。
  - 安全风险：无。

    - 未覆盖场景：MTP（多头 token 预测）模型的 load_weights 未应用映射，因为尚无此类检查点需要 baked KV scales，但若未来出现，需扩展。
- 影响：
    - 用户：部署了非 NVIDIA 官方 modelopt_fp4 检查点（如 uniform W4A4，attention 也被量化）的用户现在可以正常加载和运行，FP8 KV scales 不再回退到 1.0。
    - 系统：仅影响 Qwen3.5 系列模型加载流程，改动范围小（1 个模型文件 + 1 个测试文件）。
    - 团队：提高了 modelopt_fp4 量化方案的通用性，减少了硬编码，为未来支持更多 NVFP4 变体打下了基础。
    - 风险标记：核心路径变更，兼容性风险，量化配置变更

关联脉络

- PR #18937 Enable nvfp4 checkpoint: 本 PR 修复了 #18937 引入的硬编码问题，是直接的修复 PR。
- PR #29577 Qwen3.5 text-only ModelOpt FP4 reaches base AttentionBackend from RadixLinearAttention: 关联 issue 报告了 text-only 模型在 RadixLinearAttention 中崩溃，可能与 attention 量化相关，本 PR 可能间接解决或避免该问题。