PR #34222 完整报告

sgl-project/sglang

[Feature] Support NVFP4 token embedding in ModelOpt mixed-precision checkpoints

合并时间：2026-08-10 14:37

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34222

执行摘要

- 一句话：新增 NVFP4 token embedding 支持，gather 时解量化
- 推荐动作：值得精读，尤其是 get_quant_method 分支顺序约束和 gather 时解量化的实现；对要支持 NVFP4 embedding checkpoint 的部署团队有直接价值。测试的独立 oracle 写法（逐元素循环避免镜像实现）也是可借鉴的验证模式。注意本 PR 没有 review 讨论，设计决策主要记录在 PR body 中，阅读时应一并参考。

功能与动机

PR body 明确指出：MIXED_PRECISION 格式用 per-prefix quantized_layers 描述量化，格式本身不限制 prefix 指向什么，但 ModelOptMixedPrecisionConfig.get_quant_method 只处理 LinearBase/ParallelLMHead、RadixAttention 和 FusedMoE，VocabParallelEmbedding 会落入最后的 return None——"a checkpoint whose map marks the token embedding cannot be served at all"。embedding 是行 gather 而非 GEMM，没有 NVFP4 kernel 可分发、也没有计算收益，"the gain is purely footprint"：200k token 表在数千通道上超过 10 亿参数，bf16 与 4-bit 的差值达 GiB 量级，单卡部署时省下的显存会直接变成 KV cache。

实现拆解

1. 新增 E2M1 查找表（modelopt_quant.py 中 _E2M1_LUT）与 ModelOptNvFp4EmbeddingMethod 类：create_weights 校验 input_size_per_partition 可被 group_size 整除，注册 packed uint8 权重（每行 hidden/2 字节）、e4m3 block scale（每 group 一个）、fp32 global scale weight_scale_2，以及非持久化 e2m1_lut buffer；process_weights_after_loading 为空实现。
2. 实现 gather 时解量化（embedding 方法）：token id 拍平后按行 gather packed 权重与 scale，拆偶 / 奇通道 nibble，低 3 位查 E2M1 表、最高位作符号，再乘 block_scale × global_scale，最后 reshape 回原输入形状并转 params_dtype。纯行 gather 无 GEMM，所以没有 NVFP4 kernel 可分发。
3. 扩展 get_quant_method 分发：在 ParallelLMHead 分支之后、RadixAttention/FusedMoE 分支之前插入 VocabParallelEmbedding 分支，排除或非 NVFP4 时返回 None，NVFP4 时返回 ModelOptNvFp4EmbeddingMethod；顺序约束因 ParallelLMHead 继承自 VocabParallelEmbedding，tie_word_embeddings 时 lm_head 就是 embedding 模块，代码注释记录了该顺序要求。

4. 测试配套：新增 test/registered/quant/test_nvfp4_embedding.py，包含独立逐元素循环的 reference_dequant oracle、build_layer 构造、rtol=0/atol=0 的 gather 对比测试，以及 hidden_size 必须整除 group_size 的守卫测试；注册到 CPU CI suite base-a-test-cpu。无其他配置或部署改动。

关键文件：

- python/sglang/srt/layers/quantization/modelopt_quant.py（模块 量化层；类别 source；类型 core-logic；符号 ModelOptNvFp4EmbeddingMethod, get_quant_method, create_weights, embedding）：核心实现文件：新增 ModelOptNvFp4EmbeddingMethod 类并在 get_quant_method 中扩展 VocabParallelEmbedding 分发分支，是让 NVFP4 embedding checkpoint 可被加载的关键。
- test/registered/quant/test_nvfp4_embedding.py（模块 量化测试；类别 test；类型 test-coverage；符号 reference_dequant, build_layer, TestNvFp4Embedding, test_matches_reference_dequant）：新增独立 oracle 测试，用逐元素循环验证 gather 解量化的数值正确性，并覆盖 group_size 整除守卫。

关键符号：ModelOptNvFp4EmbeddingMethod.embedding, ModelOptNvFp4EmbeddingMethod.create_weights, ModelOptNvFp4EmbeddingMethod.apply, ModelOptMixedPrecisionConfig.get_quant_method, reference_dequant, build_layer

关键源码片段

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

核心实现文件：新增 ModelOptNvFp4EmbeddingMethod 类并在 get_quant_method 中扩展 VocabParallelEmbedding 分发分支，是让 NVFP4 embedding checkpoint 可被加载的关键。

```
# python/sglang/srt/layers/quantization/modelopt_quant.py（整理后节选）

# E2M1 格式 4-bit code -> value，索引即 3-bit 尾数 code（低 3 位 magnitude，最高位 sign）
_E2M1_LUT = (0.0, 0.5, 1.0, 1.5, 2.0, 3.0, 4.0, 6.0)

class ModelOptNvFp4EmbeddingMethod(QuantizeMethodBase):
    """NVFP4 token embedding：权重保持 packed 布局，gather 时按行解量化。"""

    def __init__(self, quant_config: ModelOptFp4Config):
        self.quant_config = quant_config
        self.params_dtype = torch.bfloat16

    def embedding(self, layer: torch.nn.Module, input_: torch.Tensor) -> torch.Tensor:
        # 输入是任意形状的 token id，先拍平为 [T] 再做行 gather
        index_shape = input_.shape
        flat = input_.reshape(-1)
        packed = layer.weight[flat] # [T, H/2] uint8，每个字节装两个 4-bit code
        scale = layer.weight_scale[flat] # [T, H/16] e4m3，每个分组一个 block scale
        rows, half = packed.shape
        hidden = half * 2
```

```
        # 拆 nibble：偶通道取低 4 位，奇通道取高 4 位
        codes = packed.new_empty((rows, hidden))
        codes[:, 0::2] = packed & 0x0F
        codes[:, 1::2] = packed >> 4

        # 3-bit 尾数查 E2M1 表，最高位为符号位
        mag = layer.e2m1_lut[(codes & 0x7).long()]
        vals = torch.where(codes & 0x8 != 0, -mag, mag)

        # NVFP4 缩放 = block scale（e4m3）× global scale（fp32）的乘积
        group_size = self.quant_config.group_size
        eff = scale.float() * layer.weight_scale_2.float()
        out = vals.view(rows, hidden // group_size, group_size) * eff.unsqueeze(-1)
        return out.view(*index_shape, hidden).to(self.params_dtype)

    # get_quant_method 新分支（节选）：必须放在 ParallelLMHead 分支之后。
    # ParallelLMHead 继承自 VocabParallelEmbedding；tie_word_embeddings 时
    # lm_head 就是 embedding 模块，先命中前者才能拿到线性量化方法。
    if isinstance(layer, VocabParallelEmbedding):
        if is_layer_skipped(prefix, self.exclude_modules, self.packed_modules_mapping) \
                or self.is_layer_excluded(prefix):
            return None
        if quant_algo == "NVFP4":
            return ModelOptNvFp4EmbeddingMethod(self.nvfp4_config)
        return None
```

test/registered/quant/test_nvfp4_embedding.py

新增独立 oracle 测试，用逐元素循环验证 gather 解量化的数值正确性，并覆盖 group_size 整除守卫。

```
# test/registered/quant/test_nvfp4_embedding.py（新增，整理后节选）

GROUP_SIZE = 16
# 独立于实现的 oracle：E2M1 code 按 magnitude 升序，索引即 3-bit 尾数 code
_REFERENCE_E2M1 = [0.0, 0.5, 1.0, 1.5, 2.0, 3.0, 4.0, 6.0]

def reference_dequant(packed, block_scale, global_scale):
    """逐元素循环对照实现，刻意避免向量化，防止与待测代码互相镜像。"""
    rows, half = packed.shape
    hidden = half * 2
    out = torch.zeros(rows, hidden, dtype=torch.float32)
    for r in range(rows):
        for c in range(hidden):
            byte = int(packed[r, c // 2])
            code = (byte & 0x0F) if c % 2 == 0 else (byte >> 4) # 偶低 4 位，奇高 4 位
            magnitude = _REFERENCE_E2M1[code & 0x7]
            value = -magnitude if code & 0x8 else magnitude # 最高位为符号位
            scale = float(block_scale[r, c // GROUP_SIZE]) * global_scale
            out[r, c] = value * scale
```

```
    return out

# 测试断言：gather 结果与 oracle 在 rtol=0、atol=0 下完全一致
# def test_matches_reference_dequant(...): ...
```

评论区精华

本 PR 没有 review 评论（review_comments_count=0），核心设计决策记录在 PR body 与代码注释中：embedding 是行 gather 而非 GEMM，因此没有 NVFP4 kernel 可分发、也无计算收益，收益纯粹是 footprint；新分支必须放在 ParallelLMHead 分支之后，因为 ParallelLMHead 继承自 VocabParallelEmbedding，tie_word_embeddings 时 lm_head 就是 embedding 模块，若 checkpoint 把 lm_head 标成 NVFP4，apply() 会抛 NotImplementedError 提示修改 recipe。Issue 评论只有一次 /rerun-test registered/quant/test_nvfp4_embedding.py，CI 显示 1 个测试通过。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 分发顺序敏感：get_quant_method 新分支强依赖位于 ParallelLMHead 之后，后续重构若调整顺序，tie_word_embeddings 的 lm_head 会被当成纯 embedding 拿到 gather-only 方法，推理时 apply() 抛 NotImplementedError 导致服务不可启动。
  2. tie_word_embeddings 边界：若 checkpoint 的 quantized_layers 把 lm_head（= embedding）前缀标记为 NVFP4，本 PR 只保证加载不崩溃，推理走到 apply() 仍会报错；需要 checkpoint 作者在 recipe 中排除 embedding，或后续增加 dtype 解量化回退。
  3. 数值 / 格式差异：NVFP4 缩放是 e4m3 per-16 block scale × fp32 global scale，与 KV 路径的 E8M0 power-of-two 不同，后续若复用 kvfp4_tensor.batched_dequantize 会产生数值偏差。
  4. CUDA graph 与 float8 兼容：e2m1_lut 用非持久 buffer 规避 capture 期 host→device 拷贝，但 float8_e4m3fn 乘法在部分后端（AMD/CPU）的 kernel 支持需实测；当前测试仅跑在 CPU CI。
  5. 影响范围有限：仅影响 ModelOpt MIXED_PRECISION 加载路径，对现有非 NVFP4 embedding checkpoint 行为零变化。 - 影响：对用户 / 模型：ModelOpt 混合精度 checkpoint 若包含 NVFP4 token embedding，现在可以正常加载提供服务；对现有 checkpoint 行为完全不变（非 NVFP4 embedding 仍返回 None）。对系统资源：大词表模型（如 200k vocab × 数千 hidden）单卡部署时 embedding 显存可降低约 75%，省下的显存直接转化为 KV cache。对团队：sglang 量化链路新增一种量化方法类型，测试注册到 base-a-test-cpu 套件；改动集中在 modelopt_quant.py 的分发逻辑，不触及加载器和其他量化路径。 - 风险标记：核心分发分支顺序敏感，tie_word_embeddings 场景会抛错，CUDA graph / float8 兼容需验证，仅覆盖 NVFP4 embedding

关联脉络

- PR #34669 [CI] Split Kimi K2.5 performance batches by config: 同为 NVFP4 量化在 GB300/Blackwell 上的落地与验证，本 PR 补上 token embedding 的 NVFP4 支持，两者共同完善 NVFP4 大词表模型的部署链路。
- PR #32755 [Perf] Occupancy tuning for DSA indexer fp8-quant Q kernel: 同一量化 kernel 方向的性能调优，反映 sglang 在 fp8/nvfp4 量化链路上的持续投入。