PR #35077 完整报告

sgl-project/sglang

[Fix] Support Kimi-K3 ModelOpt mixed NVFP4/FP8 checkpoint

合并时间：2026-08-19 23:13

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35077

执行摘要

- 一句话：支持 Kimi-K3 ModelOpt NVFP4/FP8 混合检查点加载
- 推荐动作：建议精读。该 PR 展示了如何处理工业级混合精度检查点的典型套路：量化配置按层分发（_resolve_quant_algo）、scale 契约的语义对齐（SiTU 内部消费 GEMM1 scale）、以及小投影反量化的务实取舍。对从事量化模型推理、MoE 后端适配的工程师有较高参考价值。

功能与动机

PR body 明确指出：官方 nvidia/Kimi-K3-NVFP4 checkpoint 是 ModelOpt 混合精度检查点，routed MoE 专家使用 NVFP4 + SiTU（beta=4, linear_beta=25），attention 投影使用 weight-only FP8_PB_WO（128x128 block scale）。当前 main 分支无法承载该检查点：FlashInfer TRT-LLM MoE 后端在启动阶段拒绝 gated Situ 激活，也不会实例化 / 加载 block-FP8 attention scales；同时 K3 fused-front 需要将行布局的 FP32 router 输出传递给支持 stride 的 precomputed-routing 接口。

实现拆解

该 PR 的核心变更可分为以下几步：

1. FlashInfer TRT-LLM MoE runner 增加 Situ 激活支持（python/sglang/srt/layers/moe/moe_runner/flashinfer_trtllm.py）：
   - get_activation_type 的 gated 激活映射表中新增 "situ": ActivationType.Situ；
   - _compute_g1_scale_c 新增 activation 参数，当激活为 situ 时跳过 g1_alphas_up 乘法，只保留 GEMM2 输入 requant 因子，因为 SiTU 在 tanh 前会自行消费两个 GEMM1 scale；
   - _SUPPORTED_FP4_ACTIVATIONS 加入 situ。
2. ModelOpt 混合精度配置支持 FP8_PB_WO（python/sglang/srt/layers/quantization/modelopt_quant.py）：
   - ModelOptMixedPrecisionConfig 新增 fp8_pb_wo_config 字段，在 from_config 中构造 Fp8Config(is_checkpoint_fp8_serialized=True, activation_scheme="dynamic", weight_block_size=[128, 128])；
   - get_quant_method 新增 FP8_PB_WO 分支，分发到原生 Fp8LinearMethod；

- 修正 SiTU 激活下的 gemm1_clamp_limit 与 gemm1_beta 处理：clamp 置为 None，beta 取 gemm1_clamp_limit 而非 1/g1_alphas。

3. Kimi-K3 模型层加载与运行时适配（python/sglang/srt/models/kimi_k3.py）：
   - 新增 _uses_modelopt_fp8_pb_wo 辅助函数，通过 quant_config._resolve_quant_algo(prefix) 判断某层是否为 FP8_PB_WO；
   - 新增 _maybe_map_fp8_pb_scale_name 将模型文件里的 .weight_scale 映射到 SGLang 的 weight_scale_inv 参数；
   - 新增 _get_k3_dense_weight，对存在 weight_scale_inv 的模块应用 block_quant_dequant 得到 BF16 稠密权重，供 B/F_a 融合小 GEMM 使用；
   - MoE 的 output_format 判断条件在 hidden_act == "situ" 时把 FlashInfer TRT-LLM 也纳入 precomputed top-k 路径（原来是仅 flashinfer_mxfp4）；
   - KimiK3DeltaAttention 增加 _bfa_uses_block_fp8 标志，用于决定是否解量化并缓存 B/F_a 权重，以及用实际运行 dtype 编译 KDA recompute kernel。
4. 测试配套（3 个测试文件）：
   - test/registered/unit/models/test_kimi_k3_bfa_overlap.py：新增 block-FP8 权重反量化测试与 per-tensor FP8 不被错误反量化的防护测试；
   - test/registered/unit/model_loader/test_modelopt_loader.py：验证 FP8_PB_WO 分发到 Fp8LinearMethod 且 block size 为 [128, 128]；
   - test/registered/unit/layers/quantization/test_modelopt_nvfp4_moe_scales.py：验证 situ 激活下 g1_scale_c 只包含 GEMM2 输入 requant 因子。

关键文件：

- python/sglang/srt/models/kimi_k3.py（模块 模型层；类别 source；类型 data-contract；符号 _uses_modelopt_fp8_pb_wo, _maybe_map_fp8_pb_scale_name, _get_k3_dense_weight）：模型主文件：新增 ModelOpt FP8_PB_WO 检测、block-FP8 scale 名映射、B/F_a 小投影反量化，以及 SiTU 路由输出格式调整，是本次兼容性改造的核心。
- python/sglang/srt/layers/quantization/modelopt_quant.py（模块 量化层；类别 source；类型 data-contract）：ModelOpt 混合精度配置新增 FP8_PB_WO 的 Fp8Config（128x128 block scale）与分发逻辑，让检查点中的 block-FP8 attention 层能走原生 Fp8LinearMethod。
- python/sglang/srt/layers/moe/moe_runner/flashinfer_trtllm.py（模块 MoE 后端；类别 source；类型 core-logic）：FlashInfer TRT-LLM MoE runner 新增 SiTU 激活支持，并修正 g1_scale_c 计算，是 MoE 后端正向推进的关键改动。
- test/registered/unit/models/test_kimi_k3_bfa_overlap.py（模块 模型测试；类别 test；类型 test-coverage；符号 test_block_fp8_weight_is_dequantized_for_tiny_gemm, test_per_tensor_fp8_weight_is_not_block_dequantized）：新增 block-FP8 权重反量化的单元测试与 per-tensor FP8 不被误反量化的防护测试，直接守护 _get_k3_dense_weight 的行为。
- test/registered/unit/model_loader/test_modelopt_loader.py（模块 加载器测试；类别 test；类型 test-coverage；符号 test_fp8_pb_wo_dispatches_to_native_block_fp8）：验

证 ModelOpt FP8_PB_WO 配置分发到原生 block-FP8 线性方法，属于量化配置契约的回归测试。

- test/registered/unit/layers/quantization/test_modelopt_nvfp4_moe_scales.py（模块 量化测试；类别 test；类型 test-coverage；符号 test_situ_keeps_both_dequant_scales_inside_activation）：新增 SiTU scale 契约测试，验证 g1_scale_c 只包含 GEMM2 输入 requant 因子。

关键符号：_uses_modelopt_fp8_pb_wo, _maybe_map_fp8_pb_scale_name, _get_k3_dense_weight, _compute_g1_scale_c, get_activation_type, align_fp4_moe_weights_for_flashinfer_trtllm, ModelOptMixedPrecisionConfig.from_config, ModelOptMixedPrecisionConfig.get_quant_method

关键源码片段

python/sglang/srt/models/kimi_k3.py

模型主文件：新增 ModelOpt FP8_PB_WO 检测、block-FP8 scale 名映射、B/F_a 小投影反量化，以及 SiTU 路由输出格式调整，是本次兼容性改造的核心。

```
# python/sglang/srt/models/kimi_k3.py
# 以下三个辅助函数是本 PR 支持 ModelOpt FP8_PB_WO 检查点的核心。


def _uses_modelopt_fp8_pb_wo(
    quant_config: Optional[QuantizationConfig], prefix: str
) -> bool:
    # 通过 ModelOpt 量化配置的按前缀解析器，判断某层是否使用 FP8_PB_WO。
    # 返回 True 时，该层将走 SGLang 原生 block-FP8 线性路径。
    resolver = getattr(quant_config, "_resolve_quant_algo", None)
    return resolver is not None and resolver(prefix) == "FP8_PB_WO"


def _maybe_map_fp8_pb_scale_name(name: str, params_dict: dict) -> str:
    # ModelOpt 检查点中 block-FP8 层的 scale 参数名为 weight_scale，
    # SGLang block-FP8 线性层期望的是 weight_scale_inv，这里做逻辑名映射。
    if name.endswith(".weight_scale"):
        candidate = name.removesuffix(".weight_scale") + ".weight_scale_inv"
        if candidate in params_dict:
            return candidate
    return name


def _get_k3_dense_weight(module: nn.Module) -> torch.Tensor:
    """Return a dense weight with serialized block-FP8 scales applied.

    K3 的 B/F_a 融合小 GEMM 走 BF16 路径，因此需要把序列化的
    block-FP8 权重先反量化为 BF16；主 KDA 投影仍走原生 block-FP8 GEMM。
    """
```

```
    weight = module.weight.data
    if not hasattr(module, "weight_scale_inv"):
        # per-tensor FP8 或未量化层直接返回原始权重，避免误反量化。
        return weight
    return block_quant_dequant(
        weight,
        module.weight_scale_inv,
        module.quant_method.weight_block_size,
        module.params_dtype,
    )
```

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

ModelOpt 混合精度配置新增 FP8_PB_WO 的 Fp8Config（128x128 block scale）与分发逻辑，让检查点中的 block-FP8 attention 层能走原生 Fp8LinearMethod。

```
# python/sglang/srt/layers/quantization/modelopt_quant.py
# ModelOptMixedPrecisionConfig 中新增 FP8_PB_WO 子配置与分发。

# from_config 内部新增：
fp8_pb_wo_config = Fp8Config(
    is_checkpoint_fp8_serialized=True, # 检查点中权重为序列化 FP8
    activation_scheme="dynamic", # 激活按 token 动态量化
    weight_block_size=[128, 128], # ModelOpt FP8_PB_WO 的块大小
    packed_modules_mapping=packed_modules_mapping,
)

# get_quant_method 中新增分发分支：
if quant_algo == "FP8_PB_WO":
    return Fp8LinearMethod(self.fp8_pb_wo_config)

# TRT-LLM MoE weight 准备阶段，SiTU 激活的特殊处理：
if runner_config.gemm1_alpha is not None:
    copy_or_rebind_param(
        layer, "gemm1_alpha",
        torch.full_like(layer.g1_alphas, runner_config.gemm1_alpha,
                        dtype=torch.float32),
    )
    # SiTU 在激活内部完成两个 GEMM1 scale 的消耗，gemm1_beta 直接取
    # 模型提供的 clamp 系数；非 SiTU 路径保持原有的 1/g1_alphas。
    gemm1_beta = (
        torch.full_like(layer.g1_alphas, runner_config.gemm1_clamp_limit,
                        dtype=torch.float32)
        if is_situ
        else (1.0 / layer.g1_alphas).to(torch.float32)
    )
    copy_or_rebind_param(layer, "gemm1_beta", gemm1_beta)
```

python/sglang/srt/layers/moe/moe_runner/flashinfer_trtllm.py

FlashInfer TRT-LLM MoE runner 新增 SiTU 激活支持，并修正 g1_scale_c 计算，是 MoE 后端正向推进的关键改动。

```
# python/sglang/srt/layers/moe/moe_runner/flashinfer_trtllm.py
# SiTU 激活下 GEMM1 输出 scale 的契约：

def _compute_g1_scale_c(
    w2_input_scale_quant: torch.Tensor,
    g1_alphas: torch.Tensor,
    g1_alphas_up: torch.Tensor,
    is_gated: bool,
    activation: Optional[str] = None,
) -> torch.Tensor:
    """TRT-LLM GEMM1-output scale for the up (w3) half.

    普通 gated 激活（Swiglu 等）需要把 gate 半边的 dequant scale 与
    GEMM2 输入 requant 因子合并进 g1_scale_c；SiTU 在 tanh 之前会自行
    消费两个 GEMM1 scale，因此 g1_scale_c 只携带 GEMM2 输入 requant 因子，
    避免双重应用。
    """
    if activation == "situ":
        num_experts = g1_alphas.shape[0]
        return (
            w2_input_scale_quant.to(torch.float32)
            .expand(num_experts)
            .contiguous()
        )
    if is_gated:
        return (w2_input_scale_quant * g1_alphas_up).to(torch.float32)
    num_experts = g1_alphas.shape[0]
    return (
        w2_input_scale_quant.to(torch.float32).expand(num_experts).contiguous()
    )
```

评论区精华

该 PR 无实质性 review 评论，CV 审核人 yhyang201 直接 APPROVED。PR 内部的决策主要体现在提交历史与 PR body 中：

- SiTU scale 契约的处理：作者明确将 W13 up-dequant scale 保留在 SiTU 激活内部，避免通过 g1_scale_c 双重应用。
- 路由路径选择：由于 TRT-LLM 无法消费 fused-front 的行步长 router 输出，K3 选择保留 FP32 router 并改用 precomputed top-k 路径，而不是关闭 fused-front。
- B/F_a 小投影采用 BF16 反量化融合 GEMM，而非走原生 block-FP8 GEMM，权衡了 kernel 复杂度和精度。
- 暂无高价值评论线程

风险与影响

• 风险：主要风险集中在：

1. SiTU scale 契约的脆弱性（flashinfer_trtllm.py）：_compute_g1_scale_c 新增的 activation == "situ" 分支依赖 FlashInfer 对 SiTU 的语义实现。若 FlashInfer 内部改变 SiTU 的 scale 消费方式，可能出现双重应用或缺失 scale 的精度回归。
2. FP8_PB_WO 分发范围（modelopt_quant.py）：get_quant_method 中 FP8_PB_WO 分发到 Fp8LinearMethod，但依赖 _resolve_quant_algo 的 prefix 匹配。若 ModelOpt 检查点中某些层的前缀命名与 SGLang 的 prefix 不一致，可能漏分发或错误分发，导致启动崩溃或静默精度损失。
3. block-FP8 权重反量化的内存开销（kimi_k3.py）：_get_k3_dense_weight 对 B/F_a 投影做 BF16 反量化，属预期内的小开销；但如果未来 _bfa_uses_block_fp8 的判定条件在更多模型配置下被意外触发，可能引入额外显存占用。
4. KDA recompute kernel 的 dtype 编译：若实际运行 dtype 与序列化权重 dtype 不一致的处理不完整，可能导致 kernel 编译失败或数值错误。
5. 缺少性能测试：PR body 明确说明未测量速度，属于纯正确性变更，仍需在真实负载下观察 MoE scale 计算链路变更是否引入额外开销。 - 影响：影响范围：
   • 用户：解锁 NVIDIA 官方 Kimi-K3-NVFP4 检查点在 SGLang 上的部署，TP8 下 GSM8K-200 达 0.985；FlashInfer TRT-LLM MoE 后端新增对 SiTU 激活模型的支持。
   • 系统：ModelOptMixedPrecisionConfig 新增 FP8_PB_WO 配置分支，影响所有使用 ModelOpt 混合精度检查点的模型；flashinfer_trtllm.py 的 scale 计算逻辑变更，需关注对其他激活（silu/gelu）的无回归影响。
   • 团队：为后续 ModelOpt 混合精度检查点（如其他模型的 NVFP4+FP8 组合）提供了可复用的模式。
   • 风险标记：核心路径变更，涉及数据契约，缺少性能测试，依赖 FlashInfer 语义

关联脉络

• PR #35630 [AMD] Enable Mori-EP on kimi-k3: 同为 Kimi-K3 模型的量化后端适配，涉及 MoE 与量化配置，说明 K3 的量化支持在多硬件后端持续演进。
• PR #36237 [MegaMoE] Respect padded MXFP8 scale row strides in pre-dispatch: 同为 MoE 量化 scale 的布局与步长处理，属于 MoE 量化正确性修复的同主题工作。
• PR #36097 Fix MXFP8 MoE weight sizing for non-gated models: 同为 MoE 量化权重尺寸 /scale 处理，说明 MoE 量化路径近期有多个正确性修复。