PR #33962 完整报告

sgl-project/sglang

enable TRT-LLM for MiniMax M3 by preserving SwiGLU params

合并时间：2026-08-10 14:30

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33962

执行摘要

- 一句话：MiniMax M3 启用 TRT-LLM MoE 后端，保留 SwiGLU 参数
- 推荐动作：值得精读。该 PR 是一个典型的 " 模型新后端使能 " 案例：核心设计决策是把模型级标量配置在权重加载阶段物化为 per-expert 张量，再通过 MoeQuantInfo 数据契约透传到 custom op wrapper，这种 " 整数 / 浮点配置到张量参数 " 的物化模式在多个 MoE 后端间可复用。同时应关注两个后续风险点：fp8_utils 的 cute-dsl 路由需要补充 SM120 防护，以及移除 autotune workaround 后的长尾硬件验证。建议合入后跟进毫米波的后续 --fp8-gemm-backend=flashinfer_cutedsl PR，把这次的小 M 路由逻辑统一收口。

功能与动机

PR body 的目标是让 --moe-runner-backend flashinfer_trtllm_routed 能在 MiniMax M3 上运行。M3 的 gated MoE 使用带 swiglu_alpha、swiglu_limit 等参数的 SwiGLU 激活，而 FlashInfer TRT-LLM FP8 kernel 需要按专家传入这些参数才能正确复现激活行为；此前链路中这些参数没有保留，导致 TRT-LLM 后端对 M3 不可用。作者还提到之前为规避 FlashInfer mxfp8_gemm autotuning IMA 而加的跳过逻辑（来自 #29462）已不再需要，因此一并清理并重新启用 autotune。

实现拆解

该变更从配置到内核调用打通了一条完整参数链路，按以下步骤实现：

1. 补齐配置字段：在 moe_runner/base.py 的 MoeRunnerConfig 中新增 gemm1_beta 字段；在 fused_moe_triton/layer.py 的 FusedMoE.__init__ 中增加同名参数并透传；在 minimax_m3.py 的 MiniMaxM3MoE.__init__ 中传入 gemm1_beta=1.0（M3 checkpoint 没有 swiglu_beta 配置，默认 beta 为 1.0）。这样 gemm1_alpha、gemm1_beta、gemm1_clamp_limit 三个标量就能随 model config 进入 MoE runner 配置。
2. 激活参数物化：在 layers/quantization/fp8.py 的 process_weights_after_loading 中，当 MoE runner 为 flashinfer_trtllm 或 flashinfer_trtllm_routed 时调用新增的 _prepare_flashinfer_trtllm_activation_params，把三个标量（或 None）以 torch.full 展开成 [num_local_experts] 形状的 float32 张量，挂到 layer 上（属性名为 _flashinfer_trtllm_gemm1_alpha 等），供后续 apply 阶段读取。
3. kernel 参数透传：在 layers/moe/moe_runner/flashinfer_trtllm.py 的 FlashInferTrtllmFp8MoeQuantInfo 中新增 gemm1_alpha、gemm1_beta、

gemm1_clamp_limit 三个可选张量字段，并在 fused_experts_none_to_flashinfer_trtllm_fp8 的两条调用路径（routed 与非 routed）中把 quant_info 中的参数传给 wrapper；同时 layers/moe/flashinfer_trtllm_moe.py 中两个 custom op wrapper（trtllm_fp8_block_scale_moe_out_wrapper 与 trtllm_fp8_block_scale_routed_moe_out_wrapper）新增同名参数并放入 kwargs 转发给 FlashInfer kernel。

4. autotune 与性能配套：model_executor/runner/flashinfer_autotune.py 将 FLASHINFER_AUTOTUNE_WORKAROUND_SKIPS 从 {"mxfp8_gemm"} 清空为 frozenset()，恢复 mxfp8_gemm 的 autotune；layers/quantization/fp8_utils.py 的 flashinfer_mxfp8_blockscaled_linear 在 backend == "cutlass" 且 q_input.shape[0] <= 64 时自动切换到 cute-dsl，利用其在 decode 小 batch 下 2-5 倍的 kernel 优势。
5. 测试与验证配套：本 PR 没有新增单元测试文件；验证依赖 PR body 中的 GSM8K（97%）与 GB300 TP4 上的速度数据（bs1 解码从 126.0 提升到 158.4 tok/s，加 dense sparse decode 后 167.7 tok/s）。

关键文件：

- python/sglang/srt/layers/quantization/fp8.py（模块 量化层；类别 source；类型 core-logic；符号 _prepare_flashinfer_trtllm_activation_params）：核心改动文件：新增 _prepare_flashinfer_trtllm_activation_params 将标量 SwiGLU 参数物化为 per-expert 张量，并在 apply 阶段写入 FlashInferTrtllmFp8MoeQuantInfo，是整个参数链路的源头。
- python/sglang/srt/layers/moe/moe_runner/flashinfer_trtllm.py（模块 MoE 运行时；类别 source；类型 core-logic；符号 FlashInferTrtllmFp8MoeQuantInfo, fused_experts_none_to_flashinfer_trtllm_fp8）：FlashInferTrtllmFp8MoeQuantInfo 新增三个可选张量字段，并在 routed 与非 routed 两条内核调用路径中透传，是数据契约的关键环节。
- python/sglang/srt/layers/moe/flashinfer_trtllm_moe.py（模块 内核封装；类别 source；类型 core-logic；符号 trtllm_fp8_block_scale_moe_out_wrapper, trtllm_fp8_block_scale_routed_moe_out_wrapper）：两个 custom op wrapper 新增 gemm1_alpha/beta/clamp_limit 参数并转发给 FlashInfer kernel，是 torch.compile 兼容包装层的必要改动。
- python/sglang/srt/layers/quantization/fp8_utils.py（模块 量化工具；类别 source；类型 core-logic；符号 flashinfer_mxfp8_blockscaled_linear）：MXFP8 线性层在 M <= 64 时自动从 cutlass 切换到 cute-dsl，带来约 25% decode 吞吐提升，但引入 SM120 不支持的兼容性风险，是讨论焦点。
- python/sglang/srt/model_executor/runner/flashinfer_autotune.py（模块 自动调优；类别 source；类型 data-contract；符号 get_flashinfer_autotune_skip_ops）：清空 mxfp8_gemm autotune 跳过集合，恢复该算子的 autotune，是解除 #29462 workaround 的关键动作。
- python/sglang/srt/models/minimax_m3.py（模块 模型定义；类别 source；类型 data-contract；符号 MiniMaxM3MoE）：为 MiniMax M3 的 MoE 层传入 gemm1_beta=1.0，补齐 TRT-LLM SwiGLU 参数链路的模型端入口。
- python/sglang/srt/layers/moe/fused_moe_triton/layer.py（模块 Triton 层；类别 source；类型 core-logic；符号 FusedMoE）：FusedMoE 构造函数增加 gemm1_beta 参数并透

传，保证 Triton 与 TRT-LLM 等后端共享同一配置契约。

- python/sglang/srt/layers/moe/moe_runner/base.py（模块 运行配置；类别 source；类型 core-logic；符号 MoeRunnerConfig）：MoeRunnerConfig 新增 gemm1_beta 字段，是所有 MoE runner 共享配置的数据契约变更。

关键符号：_prepare_flashinfer_trtllm_activation_params, trtllm_fp8_block_scale_moe_out_wrapper, trtllm_fp8_block_scale_routed_moe_out_wrapper, flashinfer_mxfp8_blockscaled_linear, fused_experts_none_to_flashinfer_trtllm_fp8, get_flashinfer_autotune_skip_ops

关键源码片段

python/sglang/srt/layers/quantization/fp8.py

核心改动文件：新增 _prepare_flashinfer_trtllm_activation_params 将标量 SwiGLU 参数物化为 per-expert 张量，并在 apply 阶段写入 FlashInferTrtllmFp8MoeQuantInfo，是整个参数链路的源头。

```
# 权重加载完成后，把 TRT-LLM SwiGLU 的标量参数物化为 per-expert tensor。
# 背景：MiniMax M3 的 gated MoE 携带 gemm1_alpha / gemm1_beta /
# gemm1_clamp_limit 等激活参数，FlashInfer TRT-LLM FP8 kernel 需要以
# [num_experts] 形状的张量接收，而不是 Python 标量。
def _prepare_flashinfer_trtllm_activation_params(self, layer: Module) -> None:
    """Materialize optional TRT-LLM SwiGLU parameters once per expert."""
    num_experts = int(layer.num_local_experts)
    device = layer.w13_weight.device
    # 每个参数要么是 None（kernel 用默认值），要么展开成与专家数量对齐的
    # float32 张量，挂到 layer 上供 apply 阶段读取。
    for name, value in (
        ("gemm1_alpha", self.moe_runner_config.gemm1_alpha),
        ("gemm1_beta", self.moe_runner_config.gemm1_beta),
        ("gemm1_clamp_limit", self.moe_runner_config.gemm1_clamp_limit),
    ):
        tensor = (
            None
            if value is None
            else torch.full(
                (num_experts,),
                float(value),
                dtype=torch.float32,
                device=device,
            )
        )
        setattr(layer, f"_flashinfer_trtllm_{name}", tensor)

# 调用点在 process_weights_after_loading 中，与权重 layout 对齐（
# align_fp8_moe_weights_for_flashinfer_trtllm）同属 flashinfer_trtllm
# 后端的加载流程，保证 apply 阶段可以安全直接访问上面的属性。
if (
```

```
        get_moe_runner_backend().is_flashinfer_trtllm()
        or get_moe_runner_backend().is_flashinfer_trtllm_routed()
    ):
        self._prepare_flashinfer_trtllm_activation_params(layer)
```

python/sglang/srt/layers/moe/flashinfer_trtllm_moe.py

两个 custom op wrapper 新增 gemm1_alpha/beta/clamp_limit 参数并转发给 FlashInfer kernel，是 torch.compile 兼容包装层的必要改动。

```
# TRT-LLM routed MoE 的 custom op 封装：把 sglang 侧的 per-expert
# SwiGLU 参数（gemm1_alpha / gemm1_beta / gemm1_clamp_limit）透传给
# FlashInfer kernel。此前这些参数缺失，导致 MiniMax M3 无法走该后端。
@register_custom_op(
    fake_impl=_fake_fp8_block_scale_routed_moe_out,
    mutates_args=["output"],
)
def trtllm_fp8_block_scale_routed_moe_out_wrapper(
    topk_ids: torch.Tensor,
    routing_bias: Optional[torch.Tensor],
    hidden_states: torch.Tensor,
    hidden_states_scale: torch.Tensor,
    gemm1_weights: torch.Tensor,
    gemm1_weights_scale: torch.Tensor,
    gemm1_alpha: Optional[torch.Tensor],
    gemm1_beta: Optional[torch.Tensor],
    gemm1_clamp_limit: Optional[torch.Tensor],
    gemm2_weights: torch.Tensor,
    gemm2_weights_scale: torch.Tensor,
    num_experts: int,
    top_k: int,
    n_group: Optional[int],
    topk_group: Optional[int],
    intermediate_size: int,
    local_expert_offset: int,
    local_num_experts: int,
    routed_scaling_factor: Optional[float],
    output: torch.Tensor,
    routing_method_type: int = 0,
    use_shuffled_weight: bool = False,
    weight_layout: int = 0,
    enable_pdl: Optional[bool] = None,
    tune_max_num_tokens: int = 8192,
    fp8_quantization_type: Optional[int] = None,
    activation_type: Optional[int] = None,
) -> None:
    try:
        from flashinfer.fused_moe import trtllm_fp8_block_scale_routed_moe
    except ImportError as e:
        raise ImportError(
```

```
            "Can't import trtllm_fp8_block_scale_routed_moe from flashinfer. "
            "Please check flashinfer version."
        ) from e

    kwargs = {
        "topk_ids": topk_ids,
        "routing_bias": routing_bias,
        "hidden_states": hidden_states,
        "hidden_states_scale": hidden_states_scale,
        "gemm1_weights": gemm1_weights,
        "gemm1_weights_scale": gemm1_weights_scale,
        # 新增参数：即使为 None 也要显式传递，FlashInfer kernel 内部
        # 会退回默认 SwiGLU 行为（alpha=1 / beta=0 / 不 clamp）。
        "gemm1_alpha": gemm1_alpha,
        "gemm1_beta": gemm1_beta,
        "gemm1_clamp_limit": gemm1_clamp_limit,
        "gemm2_weights": gemm2_weights,
        "gemm2_weights_scale": gemm2_weights_scale,
        "output": output,
        "num_experts": num_experts,
        "top_k": top_k,
        "n_group": n_group,
        "topk_group": topk_group,
        "intermediate_size": intermediate_size,
        "local_expert_offset": local_expert_offset,
        "local_num_experts": local_num_experts,
        "routed_scaling_factor": routed_scaling_factor,
        "routing_method_type": routing_method_type,
        "use_shuffled_weight": use_shuffled_weight,
        "weight_layout": weight_layout,
        "enable_pdl": enable_pdl,
        "tune_max_num_tokens": tune_max_num_tokens,
    }
    if fp8_quantization_type is not None:
        from flashinfer.fused_moe import Fp8QuantizationType

        kwargs["fp8_quantization_type"] = Fp8QuantizationType(fp8_quantization_type)

    if activation_type is not None:
        from flashinfer.fused_moe.core import ActivationType

        kwargs["activation_type"] = ActivationType(activation_type)

    trtllm_fp8_block_scale_routed_moe(**kwargs)
```

python/sglang/srt/layers/quantization/fp8_utils.py

MXFP8 线性层在 M <= 64 时自动从 cutlass 切换到 cute-dsl，带来约 25% decode 吞吐提升，但引入 SM120 不支持的兼容性风险，是讨论焦点。

```
# 小 batch 场景（M <= 64）下，CUTLASS 持久化 kernel 比 CuTe-DSL
# swap-AB / split-K kernel 慢 2-5 倍，且两者消费相同的 swizzled 1D scale，
# 因此在 decode 阶段自动切到 cute-dsl，换取约 25% 吞吐提升。
#
# 注意：这是一个有硬件边界的优化，SM120 不支持 cute-dsl，后续需要
# 在 backend 路由逻辑中补充硬件过滤（见 review 讨论）。

# 前置：输入被量化为 q_input 与其缩放 x_scale_u8，输出 dtype 已确定。
# 在保留 swizzled scale layout 的前提下切换 kernel 后端是安全的。
if backend == "cutlass" and q_input.shape[0] <= 64:
    backend = "cute-dsl"

# 两种后端都消费同一个 swizzled 1D scale；TRT-LLM 后端保持原有
# scale.view(-1) 路径不变。
if backend == "trtllm":
    weight_scale_t = weight_scale.view(-1)
else:
    weight_scale_t = weight_scale.t() if weight_scale.ndim == 2 else weight_scale

output = flashinfer_mm_mxfp8(
    q_input,
    weight.t(),
    x_scale_u8,
    weight_scale_t,
    out_dtype=output_dtype,
    use_8x4_sf_layout=False,
    backend=backend,
)
```

评论区精华

review 中主要有三处交锋：

1. mmangkad 建议 minimax_m3.py 中 gemm1_beta=1.0 改为 getattr(config, "swiglu_beta", 1.0)，以兼容未来 checkpoint 显式携带 swiglu_beta 的情况；最终合入版本仍为硬编码 1.0，因为 M3 当前 config 无此字段。
2. mmangkad 建议 fp8.py 的 apply 阶段用 getattr(layer, "_flashinfer_trtllm_gemm1_alpha", None) 做兜底，避免属性缺失时 AttributeError；最终合入版本仍为直接属性访问，依赖 _prepare_flashinfer_trtllm_activation_params 在权重加载流程中保证属性存在。
3. mmangkad 对 fp8_utils.py 的 cute-dsl 小 M 路由要求 benchmark 数据，zcnrex 给出 GB300 TP4 上 158.4 vs 126.0 tok/s（+25.7%）；mmangkad 随后指出他自己也正在做 --fp8-gemm-backend=flashinfer_cutedsl 相关工作，且 cute-dsl 通常到 M <= 256 都优于 cutlass，但该路由改动会破坏不支持 cute-dsl 的 SM120，属于遗留风险。

- gemm1_beta 取值的健壮性 (design): 维持 gemm1_beta=1.0：当前 M3 checkpoint 没有 swiglu_beta 字段，且 swiglu_beta 语义上等同于 beta 缩放系数，默认 1.0 是正确的。

- layer 属性访问是否需要 getattr 兜底 (style): 直接访问属性：_prepare_flashinfer_trtllm_activation_params 在权重加载流程中保证属性一定存在，getattr 兜底会掩盖真实 bug。
- cute-dsl 小 M 路由的 benchmark 依据 (performance): 数据充分，路由改动被接受；mmangkad 补充说明他也在做 --fp8-gemm-backend=flashinfer_cutedsl，发现 cute-dsl 通常到 M <= 256 都优于 cutlass。
- cute-dsl 路由对 SM120 的兼容性影响 (correctness): 未在本 PR 中修复，作为已知风险遗留，留待后续 fp8-gemm-backend 统一工作收口。
- 移除 mxfp8_gemm autotune 跳过是否安全 (question): 双方均未复现 IMA，同意清空 FLASHINFER_AUTOTUNE_WORKAROUND_SKIPS，恢复 autotune。

风险与影响

- 风险：
  1. SM120 兼容性回归（fp8_utils.py）：flashinfer_mxfp8_blockscaled_linear 在 M <= 64 且 backend 为 cutlass 时无条件切到 cute-dsl，而 SM120 设备不支持 cute-dsl，可能导致该设备上 MXFP8 小块 GEMM 直接失败或回退，reviewer mmangkad 明确指出了这一点。
  2. 隐式属性契约（fp8.py）：apply 阶段直接读取 layer._flashinfer_trtllm_gemm1_alpha 等属性，依赖 _prepare_flashinfer_trtllm_activation_params 一定在 process_weights_after_loading 中执行；若未来有其他代码路径绕过该函数（如直接构造 layer 后 apply），会因缺少属性而报错。reviewer 的 getattr 兜底建议未被采纳。
  3. autotune workaround 移除（flashinfer_autotune.py）：清空 mxfp8_gemm 的跳过集合后，可能重新引入 #29462 描述的 autotune IMA；作者与 reviewer 均表示未复现，但覆盖硬件和 FlashInfer 版本范围有限，不能完全排除。
  4. 缺少测试覆盖：8 个文件改动没有任何对应单元测试，参数链路（config -> layer -> quant_info -> wrapper -> kernel）的回退行为完全依赖手工 benchmark 与 GSM8K 验证。
  5. gemm1_beta 硬编码：若未来 M3 checkpoint 引入非 1.0 的 swiglu_beta，硬编码会导致静默数值偏差。 - 影响：用户侧：MiniMax M3（MXFP8）用户现在可以在 flashinfer_trtllm_routed 后端下推理，配合 SGLANG_OPT_USE_MINIMAX_DENSE_SPARSE_DECODE=1 解码吞吐可达 167.7 tok/s，GSM8K 精度 97%；系统侧：fp8_utils.py 的小 M 路由影响所有使用 MXFP8 且走 cutlass 后端的模型，decode 小 batch 场景普遍受益（约 25% 提升），但同时给 SM120 用户带来兼容性风险；团队侧：该 PR 清理了遗留的 autotune workaround，为后续 --fp8-gemm-backend=flashinfer_cutedsl 的统一后端选择工作提供了实践依据，但本 PR 未附带测试，需要后续补齐。 - 风险标记：SM120 兼容性回归，缺少测试覆盖，autotune workaround 移除风险，隐式加载顺序契约

关联脉络

- PR #32229 fix(minimax): use routed TRT-LLM for NVFP4 MoE auto on SM100: 同为 MiniMax 系列模型在 TRT-LLM MoE 后端上的路由修复，属于同一功能线，本 PR 是其向

M3 + FP8 的延伸。

- PR #34217 [misc] Pass FP8 scales in FlashInfer SWA prefill, autotune fp8 on SM120, and tighten is_image_understandable_model: 同为 FlashInfer FP8 与 autotune 相关改动，且涉及 SM120 的 fp8 autotune 支持，与本 PR 的 autotune workaround 清理和 SM120 风险点直接相关。