

PR #39678 完整报告

sgl-project/sglang

[misc] Merge FlashInfer autotune caches across spec workers, pad MXFP4 TP shards, drop dead ngram attrs

合并时间: 2026-09-16 16:50

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/39678>

执行摘要

- 一句话: 合并 autotune 缓存并修复 MXFP4 TP 对齐等六项杂项
- 推荐动作: 值得精读, 但只需重点看三个文件: `mxfp4_flashinfer_trtllm_moe.py` 的 `_pad_intermediate_size` (补齐布局与 `scale` 填充值的选择)、`flashinfer_autotune.py` 的 `flashinfer_autotune_context` (对 FlashInfer AutoTuner load/save 语义的绕过方式)、`fp8_utils.py` 的 `block_supported` 守卫 (tuple/list 比较这种容易写错的细节)。`scheduler.py`、`fp8.py`、`weight_updater.py` 三处属于清理项, 浏览即可。若你是 SM100 MXFP4 TP 配置的使用者, 建议按 `print_warning_once` 的提示实测补齐后的吞吐, 并与 TP/EP 组合对比; 若你是维护者, 建议把被删掉的 MXFP4 TP padding 回归用例以更轻量形式补回, 并在文档里明确“补齐只发生在权重加载后、形状契约以 `intermediate_size_per_partition` 为准”。

功能与动机

PR body 自述这是 “Six independent small fixes, one per commit”, 没有关联 Issue, 动机来自各模块在真实负载下暴露的具体缺陷: (1) spec decode 中 `target` 与 `draft` 会先后进入 `autotune`, 而 `autotune(cache=...)` 在进入时会清空所有从文件加载的 `tactic`, 后进入的一方会把先加载的 `tactic` 抹掉; (2) TRT-LLM MXFP4 MoE 内核按 128 宽度对齐处理 expert 权重, TP 切分后本地 `intermediate` 常常不是 128 的倍数, 会让配置直接不可用; (3) DeepGEMM 的 `block-FP8` 快路径此前只看 `weight.shape` 与 `dtype`, 未校验 `block_size`, 非 128x128 的量化块会进入只在 128x128 上验证过的内核; (4) 其余三处分别对应无人读取的 `scheduler ngram` 属性、E2M1 负零未编码以及一个没有占位符的 f-string。

实现拆解

1. FlashInfer autotune 缓存跨 worker 合并 (`python/sglang/srt/model_executor/runner/flashinfer_autotune.py`): `flashinfer_autotune_context` 把 `envs.SGLANG_FLASHINFER_AUTOTUNE_CACHE.get()` 抽成局部变量 `reuse_cache`; 随后不再把缓存路径交给 `autotune(cache=...)`, 而是先取 `AutoTuner.get()`, 命中已有缓存文件时手工 `tuner.load_configs(...)`, 用 `autotune(True, cache=None if reuse_cache else str(autotune_cache), skip_ops=skip_ops)` 进入内核调优, 退出 `yield` 后再 `tuner.save_configs(...)`。这样 `target` 与 `draft` 两个调优阶段在同一进程 / 同一缓存文件上做并集合并, 不再互相覆盖。

2. MXFP4 TP 分片补齐到 128 对齐 (python/sglang/srt/layers/quantization/mxftp4_flashinfer_trtllm_moe.py) : 新增模块级函数 `_pad_intermediate_size(layer)`, 在 `Mxftp4FlashinferTrtllmMoEMethod.process_weights_after_loading` 中紧跟 `_mega_moe_weights_built` 早退判断之后、`reorder_w1w3_to_w3w1` 之前调用。逻辑为: `intermediate_size = layer.w13_weight.shape[1] // 2` 向上取整到 128 的倍数; 对 `w13_weight` (补 0) 与 `w13_weight_scale_inv` (补 1) 重分配为 `(num_experts, 2 * padded_size, width)`, 把 gate 半段与 up 半段分别放到各自起始位置, 两段后各留零尾; 对 `w2_weight` (元素粒度 2) 与 `w2_weight_scale_inv` (block 粒度 32) 按最后一维补零; 最后把 `layer.intermediate_size_per_partition` 同步为 `padded_size`, 并用 `print_warning_once` 提示补齐会浪费算力与显存。
3. DeepGEMM block-FP8 只保留 128x128 (python/sglang/srt/layers/quantization/fp8_utils.py) : 在 `deepgemm_w8a8_block_fp8_linear_with_fallback` 中新增 `block_supported = list(block_size) == [128, 128]`, 与原有 `shape_supported`、`dtype_supported` 一起决定是否走 `w8a8_block_fp8_matmul_deepgemm`; 不满足时走既有 Triton 回退 (含 UE8M0 packed scale 的解包)。用 `list()` 而非直接比较, 是因为 `block_size` 可能以 tuple 传入。
4. FP4 反量化表编码负零 (python/sglang/srt/layers/quantization/fp8.py) : `DSV4_DEQUANT_FP4_TABLE` 第 8 项由 0.0 改为 -0.0, 与 E2M1 编码 `0b1000` 的负零一致; 该表被 `cast_e2m1fn_to_e4m3fn` 查表使用, 改动让反量化结果在符号位上保持 bit-exact。
5. 删除 scheduler 失效 ngram 属性 (python/sglang/srt/managers/scheduler.py) : `Scheduler.maybe_init_ngram_embedding` 只保留 `ngram_embedding_manager` 与 `use_ngram_embedding`, 删掉 `token_table`、`ngram_embedding_n`、`ngram_embedding_k` 三个赋值 (表仍可从 manager 上取)。
6. 去掉多余 f-string (python/sglang/srt/model_executor/model_runner_components/weight_updater.py) : `_update_bucketed_weights_from_distributed` 的成功返回串由 f-string 改为普通字符串, 纯风格修复, 不改语义。
7. 测试与 CI 配套: 分支期间先新增了 MXFP4 TP padding 回归用例、并把同类断言并入 `test/registered/unit/layers/quantization/test_mxftp4_sm100_trtllm_gen.py`, 随后在 CI 清理中被删除 (commit `remove added MXFP4 TP padding test` 由 BBuf 提交, 另有 `drop autotune cache test`)。作者用 `/rerun-test` 显式复跑了 5 个用例, 结果全绿: 4-gpu-b200 的 `test_mxftp4_sm100_trtllm_gen.py`, ubuntu-latest 上的 `test_flashinfer_autotune_sync.py`、`test_deepgemm_ue8m0_requant.py`、`test_flashinfer_trtllm_fp8_fallback.py`, 以及 1-gpu-5090 的 `test_basic_sanity.py`。最终 commit 为 `687226a` (tuple-safe block gate; sync padded intermediate size)。

关键文件:

- python/sglang/srt/layers/quantization/mxftp4_flashinfer_trtllm_moe.py (模块 专家量化; 类别 source; 类型 core-logic; 符号 `_pad_intermediate_size`, `Mxftp4FlashinferTrtllmMoEMethod.process_weights_after_loading`) : 本 PR 体量最大、行为影响最直接的文件: 新增 `_pad_intermediate_size` 并在 `process_weights_after_loading` 中调用, 让 TP 分片后的 flashinfer_mxftp4 TRT-LLM MoE 权重满足 128 对齐要求, 同时改写 layer 的形状元数据。

- python/sglang/srt/model_executor/runner/flashinfer_autotune.py (模块 自动调优; 类别 source; 类型 core-logic; 符号 flashinfer_autotune_context) : 修复 spec decode 下 target 与 draft 两级 autotune 缓存互相覆盖的问题: 把缓存 load/save 从 FlashInfer 的 autotune(cache=...) 手里接管过来, 实现 tactic 合并。
- python/sglang/srt/layers/quantization/fp8_utils.py (模块 块状量化; 类别 source; 类型 core-logic; 符号 deepgemm_w8a8_block_fp8_linear_with_fallback) : 为 DeepGEMM block-FP8 快路径加上 128x128 量化块守卫, 避免非 128x128 的 block 进入只在该形状上验证过的内核; 用 list(block_size) 保证 tuple 传入也能命中。
- python/sglang/srt/layers/quantization/fp8.py (模块 反量化; 类别 source; 类型 data-contract; 符号 DSV4_DEQUANT_FP4_TABLE, cast_e2m1fn_to_e4m3fn) : DSV4_DEQUANT_FP4_TABLE 第 8 项由 0.0 改为 -0.0, 让 E2M1 的负零编码在查表反量化时保持 bit-exact, 被 cast_e2m1fn_to_e4m3fn 使用。
- python/sglang/srt/managers/scheduler.py (模块 调度器; 类别 source; 类型 dead-code-removal; 符号 maybe_init_ngram_embedding) : maybe_init_ngram_embedding 删除已无人读取的 token_table、ngram_embedding_n、ngram_embedding_k 三个属性赋值, 属于纯清理 (表仍可从 ngram_embedding_manager 取)。
- python/sglang/srt/model_executor/model_runner_components/weight_updater.py (模块 权重更新; 类别 source; 类型 style-cleanup; 符号 _update_bucketed_weights_from_distributed) : _update_bucketed_weights_from_distributed 的成功返回串由无占位符的 f-string 改为普通字符串, 纯风格清理, 不改行为。

关键符号: `_pad_intermediate_size`, `flashinfer_autotune_context`, `deepgemm_w8a8_block_fp8_linear_with_fallback`, `maybe_init_ngram_embedding`, `cast_e2m1fn_to_e4m3fn`, `_update_bucketed_weights_from_distributed`

关键源码片段

[python/sglang/srt/layers/quantization/mx_fp4_flashinfer_trtllm_moe.py](#)

本 PR 体量最大、行为影响最直接的文件: 新增 `_pad_intermediate_size` 并在 `process_weights_after_loading` 中调用, 让 TP 分片后的 flashinfer_mx_fp4 TRT-LLM MoE 权重满足 128 对齐要求, 同时改写 layer 的形状元数据。

```
# 在 MoE 权重加载完成后原地补齐: 权重补 0、scale 补 1,
# 使本地 intermediate 维度满足 TRT-LLM MXFP4 内核的 128 宽度对齐要求。
def _pad_intermediate_size(layer: Module) -> None:
    intermediate_size = layer.w13_weight.shape[1] // 2 # w13 把 gate / up 拼在同一维
    padded_size = (intermediate_size + 127) // 128 * 128
    if padded_size == intermediate_size:
        return # 已对齐则完全不动权重, 避免无谓的拷贝与显存占用

    # gate 与 up 各占一半, 所以补零必须分两段做:
    # 前半段 (gate) 放在起始位置, 后半段 (up) 整体后移到 padded_size
    # 处, 中间与末尾各留零尾。
```

```

for name, fill_value in (
    ("w13_weight", 0), # 权重补 0, 不贡献任何输出
    ("w13_weight_scale_inv", 1), # scale 补 1, 避免出现 0 scale 影响后续 e8m0 换算
):
    param = getattr(layer, name)
    num_experts, _, width = param.shape
    padded = torch.full(
        (num_experts, 2 * padded_size, width),
        fill_value,
        dtype=param.dtype, # 沿用参数自身 dtype, 兼容 fp32 与 e8m0 两种 scale 存储
        device=param.device,
    )
    padded[:, :intermediate_size] = param[:, :intermediate_size]
    padded[:, padded_size : padded_size + intermediate_size] = param[
        :, intermediate_size:
    ]
    param.data = padded

# w2 的最后一维按打包粒度换算: FP4 每 2 个值压进 1 列, scale 每 32 个值共享 1 个 block。
for name, elements_per_column, fill_value in (
    ("w2_weight", 2, 0),
    ("w2_weight_scale_inv", 32, 1),
):
    param = getattr(layer, name)
    num_experts, hidden_size, width = param.shape
    padded = torch.full(
        (num_experts, hidden_size, padded_size // elements_per_column),
        fill_value,
        dtype=param.dtype,
        device=param.device,
    )
    padded[:, :, :width] = param
    param.data = padded

# 下游 MoE runner / 通信按该属性推导形状, 必须同步成补齐后的值,
# 否则权重实际形状与 layer 元数据不一致 (最后一个 commit 专门修了这一点)。
layer.intermediate_size_per_partition = padded_size
print_warning_once(
    f"flashinfer_mxfp4 MoE padded the local intermediate size from "
    f"{intermediate_size} to {padded_size} for 128-element kernel alignment "
    "after TP weight loading. Padding adds unused channels and may waste "
    "compute and memory. Use this TP MoE configuration with caution and "
    "benchmark it against a TP/EP configuration that avoids padding."
)

```

[python/sglang/srt/model_executor/runner/flashinfer_autotune.py](#)

修复 spec decode 下 target 与 draft 两级 autotune 缓存互相覆盖的问题: 把缓存 load/save 从 FlashInfer 的 `autotune(cache=...)` 手里接管过来, 实现 tactic 合并。

```

@contextlib.contextmanager
def flashinfer_autotune_context(model_runner: ModelRunner, *, run_lm_head: bool):
    # gate 与 load_configs 内部使用同一组输入判断是否复用缓存
    from flashinfer.autotuner import AutoTuner, _collect_metadata, autotune

    mr = model_runner
    cache_path = flashinfer_autotune_cache_path(mr)
    sync_group = _autotune_tactic_sync_group(mr.tp_group)
    reuse_cache = envs.SGLANG_FLASHINFER_AUTOTUNE_CACHE.get() # 后面 load/save 与
    autotune 参数都要读
    if reuse_cache:
        autotune_cache = cache_path
        if sync_group is not None:
            # 各 rank 缓存不一致时, 命中缓存的 rank 会跳过 profile 导致归约错位, 先统一丢弃
            _drop_diverged_autotune_cache(cache_path, sync_group, _collect_metadata())
        logger.info("Running FlashInfer autotune with cache: %s", autotune_cache)
    else:
        timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
        runs_dir = cache_path.parent / "runs"
        runs_dir.mkdir(parents=True, exist_ok=True)
        autotune_cache = runs_dir / f"{cache_path.stem}.{timestamp}{cache_path.suffix}"

    mr.forward_stream.wait_stream(torch.cuda.current_stream())
    with torch.get_device_module(mr.device).stream(mr.forward_stream):
        from sglang.srt.layers.logits_processor import autotune_dummy_run_mode

        skip_ops = get_flashinfer_autotune_skip_ops(mr)
        # autotune(cache=...) 进入时会清空所有从文件加载的 tactic: spec decode 里
        # draft worker 复用 target 的缓存路径时会把 target 已加载的 tactic 抹掉。
        # 因此这里改为手动 load / save, 让两个阶段的 tactic 在同一个 AutoTuner 里合并。
        tuner = AutoTuner.get()
        if reuse_cache and autotune_cache.is_file():
            tuner.load_configs(str(autotune_cache))
        with (
            _autotune_process_group(sync_group),
            autotune(
                True,
                cache=None if reuse_cache else str(autotune_cache),
                skip_ops=skip_ops,
            ),
            autotune_dummy_run_mode(run_lm_head=run_lm_head),
        ):
            yield
        if reuse_cache:
            tuner.save_configs(str(autotune_cache))
    torch.cuda.current_stream().wait_stream(mr.forward_stream)
    logger.info("FlashInfer autotune completed.")

```

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

为 DeepGEMM block-FP8 快路径加上 128x128 量化块守卫，避免非 128x128 的 block 进入只在该形状上验证过的内核；用 `list(block_size)` 保证 tuple 传入也能命中。

```
output_dtype = input.dtype
dtype_supported = output_dtype == torch.bfloat16

# TODO: https://github.com/sgl-project/sglang/pull/6890#issuecomment-2943395737
shape_supported = weight.shape[0] % 64 == 0 and weight.shape[1] % 128 == 0
# DeepGEMM 的 block-FP8 路径只在 128x128 量化块上验证过，其它 block 形状必须回退到 Triton。
# block_size 可能以 tuple 传入，直接与 list 比较会恒为 False 导致快路径永不命中，所以先 list()
。
block_supported = list(block_size) == [128, 128]

if not (shape_supported and dtype_supported and block_supported):
    # fall back to triton
    # UE8M0 packed scale (int32) 要先解包成 Triton 期望的 fp32 (N//block_n, K//block_k)
    if weight_scale.dtype == torch.int32:
        weight_scale = _unpack_ue8m0_scale_for_triton(
            weight_scale, weight.shape, block_size
        )
    return triton_w8a8_block_fp8_linear(
        input, weight, block_size, weight_scale, input_scale, bias
    )
```

评论区精华

该 PR 没有 review 评论（`review_comments_count = 0`），讨论集中在作者自审式返工与 CI 复跑上：

1. 作者在 PR body 中把这次改动定位为六项独立修复，其中 autotune 的关键背景是“merge target/draft FlashInfer autotune caches instead of letting the second load clear the first”，并在最终 diff 的注释里写明“autotune(cache=...) clears all file-loaded tactics on entry, which would drop the target's tactics when the draft worker loads; load and save them by hand.”。
2. 分支后期走了一轮“先加测试、再删测试”的返工：commit fold mxfp4 padding case into sm100 trtllm test → drop autotune cache test → remove added MXFP4 TP padding test (BBuf)，PR body 以“The newly added MXFP4 TP-padding regression case was removed in follow-up CI cleanup. The production padding fix remains unchanged; the earlier validation can be reproduced ad hoc when needed.”作结，生产修复保留但覆盖被撤。
3. 最后一个 commit tuple-safe block gate; sync padded intermediate size 说明作者自审时发现两个细节：block_size 可能是 tuple，直接与 list 比较会恒为 False（DeepGEMM 快路径永不命中）；padding 后必须把 intermediate_size_per_partition 同步成补齐值。
4. CI 层面作者只做了复跑，未新增稳定的回归用例：/rerun-test
test/registered/unit/layers/quantization/test_mxfp4_sm100_trtllm_gen.py ...
test/registered/core/test_basic_sanity.py 覆盖 5 个用例，bot 回执显示 3 个 workflow

全部通过。

- 新增的 MXFP4 TP padding 回归用例在 CI 清理中被删除 (testing): 生产侧补齐逻辑保持合入, 验证退化为按需 ad hoc 复现, SM100 MXFP4 TP 路径缺少专门的常驻回归用例。
- block_size 的 tuple 兼容与 padded intermediate 同步 (correctness): 两处合入前修正, 最终实现使用 `list(block_size) == [128, 128]` 并回写补齐后的 intermediate size。
- CI 复跑范围与验证结果 (testing): 改动的主要影响面在 PR 内被显式验证过, 但没有留下新的专用回归用例。
- autotune 缓存生命周期改为手动 load/save (design): 通过接管缓存读写实现 target/draft tactic 合并, 代价是缓存文件写入不再由 FlashInfer 统一管理 (异常退出时不会保存)。

风险与影响

• 风险:

1. MXFP4 补齐后的形状契约 (中风险): `_pad_intermediate_size` 改的是 layer 上多个参数的实际 shape, 并把 `layer.intermediate_size_per_partition` 同步为 `padded_size`。任何仍按补齐前尺寸推导缓冲区、通信分片或 MoE runner 配置的调用方都会与权重形状不一致——最终 commit 修的就是这类同步问题, 但材料中无法确认是否还有其他读取点。
2. 补齐粒度硬编码 (低 - 中风险): `w2_weight_scale_inv` 的 `elements_per_column = 32` 硬编码了 FP4 的块大小, `w2_weight` 用 2 表示 FP4 每字节打包两个值。若未来改变 FP4 block size 或 scale 打包布局 (例如 packed int32 scale), 这里的补零长度与 `fill_value=1` 的语义会静默错位。
3. 性能与显存代价 (已知, 低风险): 补齐会引入无效通道, 代码自身用 `print_warning_once` 提示 “Padding adds unused channels and may waste compute and memory”, 且只在 rank0 打印一次, 运维容易漏看, TP 配置可能在没有察觉的情况下损失吞吐。
4. 无回归测试 (中风险): padding 用例与 autotune 缓存用例都被删除, SM100 MXFP4 TP 路径只能靠 `test_mxfp4_sm100_trtllm_gen.py` 等现有用例间接覆盖; autotune 缓存合并的“两次调优写同一文件”场景更是没有常驻用例, 后续重构容易再次踩坑。
5. autotune 缓存生命周期 (中风险): 改为手工 `load_configs / save_configs` 后, 若在 yield 期间进程异常退出, `save_configs` 不会执行, 缓存保留旧内容 (可接受但语义变更); 另外多个 worker 对同一缓存路径的写操作没有文件锁, 正确性依然依赖 `_drop_diverged_autotune_cache` 的 group 内一致性检查。
6. DeepGEMM 回退带来的性能变化 (低风险, 方向正确): 非 128x128 的量化块改走 Triton, 吞吐会低于 DeepGEMM; 但此前这些形状本就没有被验证, 属于修正而非退化。
7. 死代码删除的隐性引用 (低风险): `Scheduler.maybe_init_ngram_embedding` 删除的 `token_table / ngram_embedding_n / ngram_embedding_k` 若仍有外部读取点会直接 `AttributeError`; 从材料看无法全量确认无引用, 需要以 grep 结果为准。- 影响: 用户侧: 在 SM100 (Blackwell) 上使用 `flashinfer_mxfp4 + TRT-LLM MoE` 且采用 TP (非 EP) 切分的部署, 从“内核对齐不满足、无法正常跑”变为可用, 代价是补齐带来的额外算力与显存开销; 使用 DeepGEMM block-FP8 且 block 形状非 128x128 的配置不再有静

默错误结果的风险，但会走更慢的 Triton 路径；使用 FlashInfer autotune 的 specdecode 部署在二次启动时能复用到 target 与 draft 合并后的 tactic 集合，缩短重复调优时间。系统侧：改动集中在 sglang/srt 的量化层、model executor runner 与 scheduler，未触及网络协议、KV 缓存或调度主循环，跨模块面较窄；DSV4_DEQUANT_FP4_TABLE 的负零修正在数值上与 0.0 等价，只在符号位可观测的场景（bit-exact 比对、e4m3 转换）有差异。团队侧：这份 PR 把若干散落的隐患一次性清掉，但同时撤掉了刚写的回归用例，等于把验证责任转移到后续按需复现，长期维护成本略微上升。 - 风险标记：MXFP4 补齐回归用例被删 ,intermediate_size_per_partition 形状契约变更，补齐粒度与 FP4 block size 硬编码耦合，autotune 缓存手动读写无文件锁，非 128x128 block 改走 Triton 回退 ,scheduler ngram 属性删除依赖无外部引用

关联脉络

- PR #39613 Accept MXFP8 dispatch in FlashInfer A2A TRT-LLM MoE: 同样在 FlashInfer TRT-LLM MoE 的量化入口做守卫调整，与本次 flashinfer_mxfp4 的补齐 / 内核对齐属于同一族适配问题。
- PR #39155 [AMD] GLM-5.2 NextN: cast draft fused MoE to per-channel FP8: 同属 draft 草稿侧 MoE 量化路径，涉及 target 与 draft 两侧权重布局差异，与本 PR 的 autotune 缓存合并动机一致。
- PR #35605 [XPU] Use torch scaled_mm for XPU block FP8 linear: 同样改 python/sglang/srt/layers/quantization/fp8_utils.py 的 block FP8 线性路径与回退选择，是本次 block_supported 守卫的同一代码区域。
- PR #38913 [Kernel] Add H20 block-FP8 MoE configs for GLM-5.3-Flash EP4/EP8: 这些调优配置普遍使用 block_shape [128, 128]，本次新增的量化块守卫直接影响它们是否继续命中 DeepGEMM 快路径。
- PR #39646 dsv4.1: standalone kernels and Python wrappers: DSV4.1 内核栈的一部分，DSV4_DEQUANT_FP4_TABLE 服务的 cast_e2m1fn_to_e4m3fn 正是该系列量化路径中的反量化环节。