

PR #38730 完整报告

sgl-project/sglang

Fix custom logit processor params when num_tokens_in_batch is used

合并时间: 2026-09-11 08:52

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/38730>

执行摘要

- 一句话: 修复多 token 批量下 custom logit 处理器参数未重复展开的 bug
- 推荐动作: 值得快速精读, 尤其是使用 speculative decoding 与自定义 logit 处理器组合的团队。核心启发: 当把采样 mask 从请求粒度扩展到 token 粒度时, 所有配套的参数列表必须同步扩展, 这类长度同步缺陷在批量采样路径中很容易被漏掉。测试用手工构造 SamplingBatchInfo 的方式隔离了模型加载与 server 依赖, 写法简洁、可复用, 可作为同类采样逻辑单测的模板。

功能与动机

PR body 明确说明这是一个 bug fix: Custom logit processing is not handling things correctly if num_tokens_in_batch > 1。即当每个请求在一次采样批量中占据多个 token 行 (如 speculative decoding 的校验阶段) 时, 自定义 logit 处理器拿到的参数列表没有随之扩展, 导致参数个数与 logits 行数不匹配、处理器内部无法正确按行配对。修复方式为 Params are manually repeated when num_tokens_in_batch > 1, 即把请求级参数在传给处理器前按 token 粒度手动重复。

实现拆解

1. 定位缺陷: 在 python/sglang/srt/layers/sampler.py 的 apply_custom_logit_processor 中, batch_mask 已通过 torch.repeat_interleave(batch_mask, num_tokens_in_batch) 从请求粒度扩展到 token 粒度, 但传给 processor 的 custom_params 列表仍只按 batch_indices 生成一份。只要 num_tokens_in_batch > 1, custom_params 的条数就小于 logits[batch_mask] 的行数, 处理器内部按行 zip 或索引时会错位、缺参或抛异常。
2. 修复实现: 将 custom_params 的列表推导改为两层循环——外层遍历命中的 batch_indices, 内层 for _ in range(num_tokens_in_batch) 将同一请求的参数连续重复 num_tokens_in_batch 次。由于 torch.repeat_interleave 同样是每个元素连续重复 n 次, 二者顺序严格一致; num_tokens_in_batch 默认为 1 时行为与原先完全等价, 默认路径零回归。
3. 测试配套: 在 test/registered/unit/sampling/test_custom_logit_processor.py 中新增 TestApplyCustomLogitProcessor 测试类, 手工构造 SamplingBatchInfo (3 个请求、batch_mask 为 [True, False, True]、num_tokens_in_batch=2), 断言处理器收到的 custom_params 为 [params[0], params[0], params[2], params[2]], 并校验 logits 最终填值结果, 覆盖了参数重复与 mask/logits 行对齐两个关键点; 同时为测试文件补充了

apply_custom_logit_processor 与 SamplingBatchInfo 的导入。

4. 合入过程：仅 2 个提交（修复 + merge main），无 review 评论，merrymercy 直接 approve。CI 曾由作者通过 /tag-and-rerun-ci、/rerun-failed-ci 触发重跑，extra 与 AMD ROCm 10 任务存在失败记录后合入。

关键文件：

- python/sglang/srt/layers/sampler.py（模块 采样器；类别 source；类型 core-logic；符号 apply_custom_logit_processor）：核心修复文件：apply_custom_logit_processor 在 num_tokens_in_batch > 1 时未将 custom_params 按 token 粒度重复，导致与扩展后的 batch_mask 不对齐；此处修复直接决定所有自定义 logit 处理器在多 token 批量下的正确性。
- test/registered/unit/sampling/test_custom_logit_processor.py（模块 自定义处理器；类别 test；类型 test-coverage；符号 TestApplyCustomLogitProcessor, test_repeats_request_params_for_each_token）：新增单测，构造 3 请求 × 2 token 的 SamplingBatchInfo 场景，验证 custom_params 按 token 重复和 logits 写入结果；该测试是本次修复唯一的正确性保障。

关键符号：apply_custom_logit_processor, test_repeats_request_params_for_each_token

关键源码片段

python/sglang/srt/layers/sampler.py

核心修复文件：apply_custom_logit_processor 在 num_tokens_in_batch > 1 时未将 custom_params 按 token 粒度重复，导致与扩展后的 batch_mask 不对齐；此处修复直接决定所有自定义 logit 处理器在多 token 批量下的正确性。

```
def apply_custom_logit_processor(
    logits: torch.Tensor,
    sampling_batch_info: SamplingBatchInfo,
    num_tokens_in_batch: int = 1,
):
    """对 logits 应用自定义 logit 处理器，原地修改。

    num_tokens_in_batch 用于支持 speculative decoding 场景：每个请求
    在 batch 中占据多个 token 行，因此 logits 的总行数等于请求数乘
    num_tokens_in_batch。
    """
    assert logits.shape[0] == len(sampling_batch_info) * num_tokens_in_batch, (
        f"The batch size of logits ({logits.shape[0]}) does not match the batch size of "
        f"sampling_batch_info ({len(sampling_batch_info)}) x num_tokens_in_batch "
        f"({num_tokens_in_batch})"
    )

    for _, (processor, batch_mask) in sampling_batch_info.custom_logit_processor.items():
        # 先按请求粒度取出需要处理的 batch 下标，例如 [0, 2]
        batch_indices = batch_mask.nonzero(as_tuple=True)[0]
```

```

assert batch_mask.shape[0] == len(sampling_batch_info), (
    f"The number of batch mask ({batch_mask.shape[0]}) does not match the number of "
    f"sampling_batch_info ({len(sampling_batch_info)})"
)
# 把 mask 从请求粒度扩展到 token 粒度：每个选中的请求连续展开成
# num_tokens_in_batch 个 True，与 logits 的行布局对齐
batch_mask = torch.repeat_interleave(batch_mask, num_tokens_in_batch)

# 关键修复：custom_params 必须与 batch_mask 同步按 token 粒度重复，
# 否则处理器拿到的参数个数（len(batch_indices)）会少于
# logits[batch_mask] 的行数，多 token 批量下无法按行配对。
# 两层循环 = 先按请求顺序，再对每个请求连续重复 num_tokens_in_batch 次
custom_params = [
    sampling_batch_info.custom_params[i]
    for i in batch_indices
    for _ in range(num_tokens_in_batch)
]

# 传入的 custom_params 现在与 logits 中选中的行严格一一对应
logits[batch_mask] = processor(logits[batch_mask], custom_params)

logger.debug(f"Custom logit processor {processor.__class__.__name__} is applied.")

```

test/registered/unit/sampling/test_custom_logit_processor.py

新增单测，构造 3 请求 × 2 token 的 SamplingBatchInfo 场景，验证 custom_params 按 token 重复和 logits 写入结果；该测试是本次修复唯一的正确性保障。

```

class TestApplyCustomLogitProcessor(CustomTestCase):
    def test_repeats_request_params_for_each_token(self):
        # 构造 3 个请求，其中第 0、2 个请求命中 batch_mask；
        # num_tokens_in_batch = 2 模拟 speculative decoding 的批量解码阶段，
        # 每个请求在 logits 中占据 2 行
        batch_size = 3
        num_tokens = 2
        params = [{"value": 1.0}, {"value": 2.0}, {"value": 3.0}]

    def processor(logits, custom_params):
        # 关键断言：每个命中请求的参数必须按 token 粒度重复，
        # 即 [ 第 0 个请求 x2, 第 2 个请求 x2 ]，与 batch_mask 展开一致
        self.assertEqual(
            custom_params, [params[0], params[0], params[2], params[2]]
        )
        for row, param in zip(logits, custom_params, strict=True):
            row.fill_(param["value"])
        return logits

sampling_info = SamplingBatchInfo(
    temperatures=torch.ones(batch_size, 1),
    top_ps=torch.ones(batch_size),

```

```

top_ks=torch.zeros(batch_size, dtype=torch.int32),
min_ps=torch.zeros(batch_size),
is_all_greedy=False,
is_any_greedy=False,
need_top_p_sampling=False,
need_top_k_sampling=False,
need_min_p_sampling=False,
vocab_size=4,
has_custom_logit_processor=True,
custom_params=params,
custom_logit_processor={0: (processor, torch.tensor([True, False, True]))},
device="cpu",
)
logits = torch.zeros(batch_size * num_tokens, 4)

apply_custom_logit_processor(
    logits, sampling_info, num_tokens_in_batch=num_tokens
)

# 命中行被填充为对应参数值，未命中行保持 0：
# 第 0 个请求两行填 1.0，第 1 个请求两行保持 0，第 2 个请求两行填 3.0
expected = torch.tensor(
    [[1.0] * 4, [1.0] * 4, [0.0] * 4, [0.0] * 4, [3.0] * 4, [3.0] * 4]
)
self.assertTrue(torch.equal(logits, expected))

```

评论区精华

本 PR 没有公开的 review 讨论线程：review_comments 为空，merrymercy 直接给出 APPROVED。Issue 评论仅包含作者两次 CI 触发命令（/tag-and-rerun-ci 与 /rerun-failed-ci retry），说明合入前曾针对失败任务做过重跑。整体属于低争议、快速合入的小型正确性修复。

- 暂无高价值评论线程

风险与影响

- 风险：

1. 回归风险低但需注意行为边界：修改仅在 num_tokens_in_batch > 1 时改变 custom_params 的形状，等于 1 时与旧逻辑完全等价；测试仅覆盖 num_tokens_in_batch=2 的正整数场景，未覆盖 batch_indices 为空、custom_params 为 None 等边界分支（原代码同样未特殊处理，不属于本次引入）。
1. CI 稳定性：合并前 PR Test (Extra) 与 AMD ROCm 10 两个任务标记为失败，作者通过 /rerun-failed-ci 重跑后合入；需留意 AMD 平台是否存在同类采样路径的兼容性问题，本函数为 CPU/GPU 通用代码。
2. 参数对象共享引用：custom_params 中同一请求的参数对象被重复引用同一份（未深拷贝），若自定义 processor 就地修改参数对象内容，会影响该请求后续 token 的处理——这是

原有行为延续，未在本 PR 中改变。

3. 性能影响可忽略：多 token 场景仅多了一次长度为 `batch_indices * num_tokens_in_batch` 的列表推导，开销极小。 - 影响：用户 / 功能层面：修复了 custom logit processor（含 OpenAI 兼容 API 的 `logit_bias`、`DisallowedTokens`、`ThinkingBudget` 等内置处理器）与 speculative decoding 多 token 批量同时启用时的参数错位问题；此前该组合可能产生 `IndexError`、错误屏蔽 token 或采样结果不符合预期。系统层面：核心采样路径 `sampler.py` 改动仅 7 行，默认路径行为不变，多 token 场景开销可忽略。团队层面：合并者为维护者 `merrymercy` 直接 approve，无遗留讨论，属于 speculative decoding 兼容性清理的一部分，为后续在多 token 批量路径上安全使用自定义采样扫清障碍。 - 风险标记：采样核心路径变更，测试仅覆盖单处理器场景，CI 曾失败后重跑，参数对象共享引用

关联脉络

- PR #37709 [PD] Transfer the DCP-replicated DSPARK draft KV in DCP1->DCP-N layouts: 同属 speculative decoding 功能线的兼容性修复，说明多 token / draft 路径上持续有配套修正；修改文件不重叠。
- PR #38756 [AMD] aiter: resolve SWA KV pool for draft workers + guard paged decode: 同为 speculative decoding draft worker 路径的修复，与本 PR 一起构成 spec decoding 场景下采样 / 注意力兼容性清理的一部分。