PR #33575 完整报告

sgl-project/sglang

[rotary] Rebuild the shared RoPE cache entry when its buffers are dead

合并时间：2026-08-05 08:24

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33575

执行摘要

- 一句话：共享 RoPE 缓存失效自动重建，防查询静默未旋转
- 推荐动作：值得精读。代码量小但揭示了一个典型的进程级共享缓存与模型生命周期相互干扰的问题，_get_live_rope_cache_entry 的设计（用默认设备区分“有意构建的 meta 条目”与“被 teardown 杀死的条目”）有借鉴意义。建议结合测试用例一起阅读，并关注后续对 multimodal_gen 副本的同步修复。

功能与动机

PR body 指出：get_rope/get_rope_cpu 返回进程级共享的 RotaryEmbedding，作为子模块挂到每个请求相同 key 的模型上。当某个模型 teardown 释放 CUDA storages 或把模块树指向 meta 设备时，共享缓存条目对之后所有复用该 key 的模型失效，且失败是静默的——in-place RoPE 算子把 cos/sin 缓存作为参数，meta tensor 会路由到 Meta 后端 no-op，查询保持未旋转而不报错，形成难以诊断的正确性 bug。

实现拆解

1. 在 python/sglang/srt/layers/rotary_embedding/factory.py 中新增 _get_live_rope_cache_entry(key)：从 _ROPE_DICT 读取缓存条目；若 torch.get_default_device() 是 meta，说明正处于 meta 设备构造阶段，直接返回缓存以保持多层共享；否则遍历 cached.buffers()，当任一 buffer 在 meta 设备或 buf.untyped_storage().nbytes() == 0 时，记录 warning、删除 _ROPE_DICT[key] 并返回 None。
2. 将 get_rope 与 get_rope_cpu 的缓存命中逻辑从 if key in _ROPE_DICT: return _ROPE_DICT[key] 改为 cached = _get_live_rope_cache_entry(key); if cached is not None: return cached，使死条目自然落入后续重建分支。
3. 新增 test/registered/rotary/test_rope_cache_invalidation.py（81 行），注册到 base-a-test-cpu 套件，覆盖 4 个语义：meta 化后重建、storage 释放后重建、活条目保持共享、meta 构造期间共享且真实构建不继承 meta 条目。
4. 提交过程中 merrymercy 先合并 main，再提交 “Fix RoPE cache test on CPU CI” 修复该测试在 CPU CI 上的问题，并通过 /tag-and-rerun-ci 重跑确认通过。
5. 无配置或部署配套改动；python/sglang/multimodal_gen/runtime/layers/rotary_embedding/factory.py 中独立复制的 _ROPE_DICT 模式未被处理，PR body 标注为合理后续工作。

关键文件：

- python/sglang/srt/layers/rotary_embedding/factory.py（模块 RoPE 缓存；类别 source；类型 core-logic；符号 _get_live_rope_cache_entry）：核心修复位置：新增 _get_live_rope_cache_entry 并在 get_rope/get_rope_cpu 中使用，是缓存失效检测与重建逻辑的载体。
- test/registered/rotary/test_rope_cache_invalidation.py（模块 缓存测试；类别 test；类型 test-coverage；符号 TestRopeCacheInvalidation, test_rebuilds_after_meta_invalidation, test_rebuilds_after_storage_release, test_live_entry_is_still_shared）：新增 CPU 回归测试，覆盖 meta 化与存储释放两种失效路径，并验证活条目共享与 meta 构造语义。

关键符号：_get_live_rope_cache_entry

关键源码片段

python/sglang/srt/layers/rotary_embedding/factory.py

核心修复位置：新增 _get_live_rope_cache_entry 并在 get_rope/get_rope_cpu 中使用，是缓存失效检测与重建逻辑的载体。

```
def _get_live_rope_cache_entry(key: Tuple) -> Optional[RotaryEmbedding]:
    # 返回 key 对应的缓存模块；若其缓冲区已"死亡"（meta 设备或零大小存储），
    # 则删除缓存条目并返回 None，强制调用方走重建路径。
    cached = _ROPE_DICT.get(key)
    if cached is None:
        return None

    # meta 设备构造阶段会刻意创建 meta 缓冲区并共享同一条目（所有层共用一份），
    # 因此当默认设备为 meta 时不做失效检查，避免每层都重建。
    if torch.get_default_device().type == "meta":
        return cached

    # 遍历所有缓冲区：任一 buffer 位于 meta 设备，或其底层存储已被释放
    #  （nbytes() == 0），都说明该共享条目已被某个模型的 teardown 杀死，
    # 必须丢弃并让后续 get_rope 调用重建。
    for buf in cached.buffers():
        if buf.device.type == "meta" or buf.untyped_storage().nbytes() == 0:
            logger.warning(
                "Discarding dead RoPE cache entry (key=%s): buffer on %s. "
                "A shared RotaryEmbedding was freed by its owner.",
                key,
                buf.device,
            )
            del _ROPE_DICT[key]
            return None
    return cached
```

test/registered/rotary/test_rope_cache_invalidation.py

新增 CPU 回归测试，覆盖 meta 化与存储释放两种失效路径，并验证活条目共享与 meta 构造语义。

```
class TestRopeCacheInvalidation(CustomTestCase):
    # get_rope 返回进程级共享模块；模型 teardown 会把模块树 meta 化或释放存储，
    # 使共享的 cos/sin 缓存失效，后续模型拿到的条目会静默产生“未旋转”输出，
    # 因此必须验证缓存条目能自动重建。
    def setUp(self):
        # 强制走 CPU 分支，避免 GPU 依赖，便于在 CPU CI 上运行。
        cpu_patch = patch("sglang.srt.layers.rotary_embedding.base._is_cpu", True)
        cpu_patch.start()
        self.addCleanup(cpu_patch.stop)
        set_global_server_args_for_scheduler(ServerArgs(model_path="dummy"))
        _ROPE_DICT.clear()

    def test_rebuilds_after_meta_invalidation(self):
        # 场景 1：模型 teardown 把共享条目 .to(device="meta")。
        rope = get_rope(**_ROPE_KWARGS)
        expected = rope.cos_sin_cache.clone()
        rope.to(device="meta")
        rebuilt = get_rope(**_ROPE_KWARGS)
        # 必须得到新实例，且 cos/sin 缓存内容与原先一致。
        self.assertIsNot(rebuilt, rope)
        self.assertNotEqual(rebuilt.cos_sin_cache.device.type, "meta")
        self.assertTrue(torch.equal(rebuilt.cos_sin_cache, expected))

    def test_rebuilds_after_storage_release(self):
        # 场景 2：模型释放底层存储，storage 大小变为 0。
        rope = get_rope(**_ROPE_KWARGS)
        expected = rope.cos_sin_cache.clone()
        rope.cos_sin_cache.untyped_storage().resize_(0)
        rebuilt = get_rope(**_ROPE_KWARGS)
        self.assertIsNot(rebuilt, rope)
        self.assertTrue(torch.equal(rebuilt.cos_sin_cache, expected))
```

评论区精华

设计权衡来自源码注释与 PR body：“死条目与 meta 构造条目在缓冲区层面无法区分——两者都是 meta 且无 storage——因此用当前默认设备来区分”，这是本修复最核心的决策点。merrymercy 先以 COMMENTED 状态给出 “approve”，随后正式 APPROVED；PR 内没有 inline review comment。最后一次 commit “Fix RoPE cache test on CPU CI” 说明测试在 CPU CI 上经历了一次修复迭代，PR 期间两次 /tag-and-rerun-ci 触发重跑。PR body 主动指出 multimodal_gen 存在同样的 _ROPE_DICT 缓存模式副本，未在本 PR 修复，留作 follow-up。

- 失效条目与 meta 构造条目的区分策略 (design)：当 torch.get_default_device() 为 meta 时跳过检查、保持共享；否则遍历 buffers，发现 meta 或零大小存储即删除条目并触发重建。

- multimodal_gen 独立副本是否同步修复 (question): 作者认为在该副本上应用相同防护是合理的后续工作，未在本 PR 范围内解决。
- CPU CI 测试修复 (testing): 测试在 CPU CI 上修复后通过，PR 被批准并合并。

风险与影响

- 风险：每次 get_rope 命中缓存时新增一次缓冲区遍历，通常只有 cos/sin 两个 buffer，开销可忽略，但热路径上多做一次 Python 级迭代，对极高频模型加载场景仍需留意。buf.untyped_storage().nbytes() == 0 对 CUDA storage 释放的判定依赖 PyTorch 的精确语义，若未来出现合法的零字节 buffer 可能被误判重建，但重建结果等价，风险低。测试只在 CPU 上执行，针对 CUDA storage 释放的 GPU 路径缺乏端到端验证，回归只能靠该单元测试间接保证。multimodal_gen 副本仍是同类静默 bug 的潜在来源。
- 影响：对用户：修复多模型加载 / 卸载 / 切换场景下查询“未旋转”导致的静默精度损失。对系统：共享缓存条目生命周期与模型生命周期解耦，失效条目自动重建，避免脏状态跨模型传播。对团队：新增 CPU 回归测试，明确了共享缓存失效的判定与重建语义，为后续副本修复提供依据。
- 风险标记：共享缓存失效检测，meta 设备分支，CPU-only 测试，multimodal_gen 副本待修复

关联脉络

- PR #33536 [diffusion] Fuse DiT FFN tanh-GELU into up-proj GEMM (cublasLt epilogue) behind quality=high (Qwen-Image 1024^2 denoise 12.36 -> 12.05 s on H200): 与本 PR body 提到的 multimodal_gen/runtime 后续修复区域同目录，后续若同步 _ROPE_DICT 副本可参考该目录的既有改动。
- PR #33451 [diffusion] FLUX.2 VAE decoder fast path behind quality=high (H200: 1024^2 97.6->29.2 ms, 2048^2 437.2->168.5 ms): 同样改动 multimodal_gen/runtime 目录，与本 PR 标注的 follow-up 区域相邻，属于同一子系统的演进脉络。