PR #6447 完整报告

verl-project/verl

[megatron] fix: Fix GPU memory leak in ref model offload by explicitly releasing storage

合并时间：2026-06-01 18:09

原文链接：http://prhub.com.cn/verl-project/verl/pull/6447

执行摘要

- 一句话：修复 Megatron ref 模型 GPU 显存泄漏
- 推荐动作：建议精读该 PR，特别是 _can_safely_resize_storage 函数的逻辑和同步拷贝 + resize_(0) 的使用模式。该修复对于避免大模型训练中的显存泄漏至关重要。但建议补充单元测试覆盖非 DDP 路径的卸载场景。

功能与动机

修复在非 DDP 路径下（用于参考模型），将参数和梯度移到 CPU 时 GPU 显存未及时释放的问题。旧实现 param.data = param.data.to("cpu", non_blocking=True) 导致旧的 GPU 张量仍然被引用直到 Python GC 运行，造成显存尖峰并可能触发 OOM。同时异步拷贝后再释放存储存在数据损坏风险。

实现拆解

1. 新增 _can_safely_resize_storage 辅助函数：在 verl/utils/megatron_utils.py 中新增该函数，用于检查一个张量是否独占其整个存储（无共享、无视图偏移、连续），确保调用 storage().resize_(0) 是安全的。
2. 改造非 DDP 路径的 CPU 卸载逻辑：在 offload_megatron_model_to_cpu 函数的 else 分支（非 DDP 路径）中，先将 param.data 和 param.grad 的旧引用保存到 old_data 和 old_grad，然后使用同步 .to("cpu") 拷贝（默认 non_blocking=False），确保数据拷贝完成后才释放 GPU 存储。
3. 显式释放 GPU 存储：在同步拷贝后，调用 old_data.storage().resize_(0) 和 old_grad.storage().resize_(0) 立即释放 GPU 显存（仅在 _can_safely_resize_storage 返回 True 时执行）。
4. 辅助内存回收：添加 gc.collect() 和 torch.cuda.empty_cache() 进一步清理 Python 垃圾回收和缓存碎片。

关键文件：

- verl/utils/megatron_utils.py（模块 工具类；类别 source；类型 core-logic；符号 _can_safely_resize_storage, offload_megatron_model_to_cpu）：修复 GPU 显存泄漏的核心文件，新增安全检查函数并改造卸载逻辑。

关键符号：_can_safely_resize_storage, offload_megatron_model_to_cpu

关键源码片段

verl/utils/megatron_utils.py

修复 GPU 显存泄漏的核心文件，新增安全检查函数并改造卸载逻辑。

```
def _can_safely_resize_storage(tensor: torch.Tensor) -> bool:
    """Check whether it is safe to call ``storage().resize_(0)`` on *tensor*.

    Resizing the underlying storage to zero immediately frees the GPU memory
    but also invalidates **every** tensor that shares the same storage
    (e.g. views, tied weights stored as different Python objects, or slices
    of a DDP flat buffer).  This function returns True only when the tensor
    exclusively owns its entire storage, making ``resize_(0)`` safe.
    """
    return (
        # Storage holds exactly the elements of this tensor — no room for
        # other tensors sharing the same storage.
        tensor.storage().size() == tensor.numel()
        # Tensor starts at the beginning of the storage — not a slice/view
        # offset into a larger buffer.
        and tensor.storage_offset() == 0
        # Tensor is contiguous in memory — rules out transposed or
        # non-contiguous views that only occupy part of the storage layout.
        and tensor.is_contiguous()
    )


@torch.no_grad()
def offload_megatron_model_to_cpu(models):
    # ... 内部 DDP 路径代码不变 ...
    else:
        # we need this for ref module
        for _, param in model_chunk.named_parameters():
            old_data = param.data
            param.data = param.data.to("cpu") # 同步拷贝，确保完成
            if _can_safely_resize_storage(old_data):
                old_data.storage().resize_(0) # 立即释放 GPU 存储
            if param.grad is not None:
                old_grad = param.grad
                param.grad = param.grad.to("cpu")
                if _can_safely_resize_storage(old_grad):
                    old_grad.storage().resize_(0)
        gc.collect()
        get_torch_device().empty_cache()
```

评论区精华

gemini-code-assist[bot] 指出直接对单个参数和梯度调用 storage().resize_(0) 存在风险，因为如果参数共享底层存储（如权重绑定、视图），则会立即失效所有其他张量。虽然 DDP 路径

操作的是整个缓冲区所以安全，但非 DDP 路径需要确保参数不共享存储。作者通过引入 _can_safely_resize_storage 函数来防范此风险。ETOgaosion 批准了 PR。

- storage().resize_(0) 安全性 (correctness): 作者通过新增 _can_safely_resize_storage 函数检查存储独占性来降低风险。

风险与影响

- 风险：主要风险在于 storage().resize_(0) 的正确使用。新增的 _can_safely_resize_storage 函数通过检查存储大小、偏移量和连续性来降低风险，但若存在未预期的共享视图（如高级索引或自定义存储分配）仍可能误判。此外，没有新增测试用例覆盖该新逻辑。
- 影响：直接影响 Megatron 训练框架下的参考模型（ref module）卸载流程，消除约 1-4GB 显存泄漏，尤其对 NPU 环境（显存紧张）影响显著。不会影响 DDP 路径（该路径已正确处理）。
- 风险标记：核心路径变更，缺少测试覆盖

关联脉络

- PR #6506 [megatron, trainer] fix: preserve BSHD top-k distillation shape: 同为对 verl/utils/megatron_utils.py 的修改，虽涉及不同功能但共享同一文件。