

PR #7874 完整报告

verl-project/verl

[trainer] fix: detach model_output from autograd graph

合并时间: 2026-09-16 11:43

原文链接: <http://prhub.com.cn/verl-project/verl/pull/7874>

执行摘要

- 一句话: detach model_output 修复多引擎逐 micro-batch 显存泄漏
- 推荐动作: 值得精读。建议关注四点:
 1. 根因定位方法: _record_memory_history 栈归因锁定
p2p_communication.py::create_tensor_recv_prev, 并用 forward_only (0.76 MiB/micro-batch) 与 PP=1 (0.31 MiB/micro-batch) 两组对照实验把“图保留”机制钉死, 是排查训练显存泄漏的范本。
 2. detach_tree 设计: 惰性 detach (已 detached 张量原样返回) 避免逐 key 创建 TensorImpl; 递归支持 dict / list / tuple, 从而覆盖 jagged nested tensors 这类新型输出结构。
 3. 回归测试设计: CPU-only 下用最小 checkpoint 模型 + weakref 复现“图被钉住”机制, 再配合 AST 静态检查 4 个引擎调用点, 绕开“必须 PP>=2 才可复现”与“无法在 CI 断言绝对显存”的双重困境。
 4. 工程陷阱提醒: “detach 位置错误导致梯度静默归零且 loss 曲线正常”说明此类内存修复必须用梯度可达性断言 (requires_grad 契约) 双保险, 而不能只靠数值曲线。

功能与动机

issue #6696/#6698 报告了统一引擎中逐 micro-batch 的 GPU 显存泄漏: forward_step 返回的 model_output 未 detach, 导致 forward_backward_batch 的 output_lst 在整个 batch 结束前一直持有每 micro-batch 的 autograd 图, LoRA + 长序列场景下首次 update_actor 即 OOM。FSDP 侧由 #6699 修复, 但 Megatron 侧的 postprocess_micro_batch_func 把相同模式的 dict 交给 forward_data_store 后同样泄漏: PR 作者实测 96 层 MoE、TP=1 PP=4、1024 个 micro-batch 时, last-stage rank 每 micro-batch 增长 45.4 MiB, 整 mini-batch 累计 +45.43 GiB, 并以 ncclUnhandledCudaError 形式崩溃 (NCCL 的 cudaMalloc 绕过缓存分配器, 掩盖了真实 OOM 形态)。

实现拆解

1. 新增共享工具 detach_tree (verl/workers/engine/utils.py): 递归处理 dict / list / tuple, 对 requires_grad=True 的张量调用 .detach(), 已 detached 的张量原样返回, 避免每个 micro-batch 每个 key 都创建多余的 TensorImpl。该文件已承载消费这些 dict 的 postprocess_batch_func, 把“发布前 detach”与“批量后处理”收敛到同一模块, 形成数据契

约的单一事实来源。

2. 修复 Megatron 发布点 (verl/workers/engine/megatron/transformer_impl.py) : 在 postprocess_micro_batch_func 构造返回 dict 处, 将 output["model_output"] 从直接赋值改为 detach_tree(model_output)。该 dict 每 micro-batch 被追加进 Megatron 的 forward_data_store, 直到整个 mini-batch 结束才清空; 剥离 autograd 图后, 1F1B 下 create_tensor_recv_prev 分配的 P2P 接收缓冲不再被间接钉住。detach 必须位于 loss_function(...) 之后: 梯度经第一个返回值 scaled_loss 流动, 若提前 detach 会让梯度静默归零 (loss 值仍由 teacher/label 侧算出, 曲线看似正常)。
3. 统一其余三个引擎 (fsdp / automodel / torchtitan) : forward_step 中的 model_output 发布全部改为 detach_tree。其中 FSDP 是把 #6699 已合入的手写 dict comprehension 重构为共享 helper (语义基本等价), automodel / torchtitan 属于同机制防护 (它们同样把逐 micro-batch 输出累积进 output_lst)。metrics 保持原样未 detach, 按 #6698 的修正说明, Metric.append 已对标量做 detach().item(), metrics 本就不保留图。
4. 回归测试与配套 (tests/workers/test_engine_detach_tree_on_cpu.py) : CPU-only 环境无法导入 mcore / torchtitan / nemo_automodel, 因此测试分两层: 行为层用 _TinyCheckpointedLM (冻结 embedding + checkpoint 块) 复现“detach 后 checkpoint 保存激活可被 weakref 回收、梯度仍正常”的机制; 源码头用 ast 静态解析 4 个引擎文件, 断言所有 model_output 发布点都经过 detach_tree。测试刻意不依赖真实 PP=2 环境 (PR 作者指出 PP=1 下泄漏无法复现), 也未做绝对显存断言。

关键文件:

- verl/workers/engine/utils.py (模块 引擎工具; 类别 source; 类型 core-logic; 符号 detach_tree) : 新增共享 helper detach_tree, 是所有引擎 detach 的单一事实来源; 递归处理 dict / list / tuple, 且对已 detached 张量原样返回以避免每 micro-batch 每 key 创建多余 TensorImpl。
- verl/workers/engine/megatron/transformer_impl.py (模块 Megatron 引擎; 类别 source; 类型 core-logic; 符号 postprocess_micro_batch_func) : Megatron 泄漏根因所在: postprocess_micro_batch_func 的返回 dict 进入 forward_data_store 且整个 mini-batch 才清空; 此处 detach 是修复生效的关键点, PP>=2 + 1F1B 下会钉住 P2P 接收缓冲。
- tests/workers/test_engine_detach_tree_on_cpu.py (模块 回归测试; 类别 test; 类型 test-coverage; 符号 _TinyCheckpointedLM, test_detach_tree_releases_the_checkpoint_saved_activation, test_detach_tree_preserves_values_and_untouched_objects, test_detach_tree_keeps_nested_tensors_intact) : 新增 CPU 回归测试: weakref + 最小 checkpoint 模型复现“detach 释放 checkpoint 保存激活”机制, 断言梯度仍正常; 并用 AST 静态检查 4 个引擎的 model_output 发布点全部经过 detach_tree, 防止回归。
- verl/workers/engine/fsdp/transformer_impl.py (模块 FSDP 引擎; 类别 source; 类型 dependency-wiring; 符号 forward_step) : 把 #6699 已合入的手写 dict comprehension 统一替换为 detach_tree, 消除重复逻辑并支持嵌套结构, 是本次“统一引擎契约”的一部分。
- verl/workers/engine/automodel/transformer_impl.py (模块 Automodel 引擎; 类别 source; 类型 data-contract; 符号 forward_step) : 该引擎同样把 model_output 累积进

output_lst, 统一 detach 避免同类泄漏; PR 初版原本不打算改, 最终合入统一。

- verl/workers/engine/torchtitan/transformer_impl.py (模块 TorchTitan 引擎; 类别 source; 类型 dependency-wiring; 符号 forward_step) : TorchTitan 引擎统一 detach; 其 jagged nested tensors 输出路径由 detach_tree 递归支持, 测试专门覆盖。

关键符号: detach_tree, postprocess_micro_batch_func, forward_step, test_detach_tree_releases_the_checkpoint_saved_activation, test_backend_publishes_model_output_through_detach_tree

关键源码片段

verl/workers/engine/utils.py

新增共享 helper `detach_tree`, 是所有引擎 detach 的单一事实来源; 递归处理 dict / list / tuple, 且对已 detached 张量原样返回以避免每 micro-batch 每 key 创建多余 TensorImpl。

```
def detach_tree(obj):  
    """剥离报告张量上的 autograd 图, 保留数据本身。
```

各后端把每个 micro-batch 的输出 dict 累积到整个 mini-batch 结束:
单程序引擎的 output_lst、Megatron 的 forward_data_store (megatron/core/pipeline_parallel/schedules.py), 而 postprocess_batch_func 要到所有 backward 都执行完后才消费它们。任何带 grad_fn 的张量进入该结构, 都会把对应 micro-batch 的整张 autograd 图钉在显存里, 使驻留内存随 micro-batch 数量线性增长: 在 PEFT 的 enable_input_require_grads 下会钉住 checkpoint 保存的 embedding 输出及其梯度缓冲; 在 Megatron 1F1B 下还会钉住图的输入, 即 p2p_communication.py::create_tensor_recv_prev 新分配的 P2P 接收缓冲。

梯度不受影响: backward 运行在单独返回的活 loss 上。

```
"""  
  
if isinstance(obj, torch.Tensor):  
    # 已经 detached 的张量原样返回, 避免每个 micro-batch 每个 key  
    # 都创建一个无用的 TensorImpl  
    return obj.detach() if obj.requires_grad else obj  
if isinstance(obj, dict):  
    return {k: detach_tree(v) for k, v in obj.items()}  
if isinstance(obj, list | tuple):  
    return type(obj)(detach_tree(v) for v in obj)  
return obj
```

verl/workers/engine/megatron/transformer_impl.py

Megatron 泄漏根因所在: postprocess_micro_batch_func 的返回 dict 进入 forward_data_store 且整个 mini-batch 才清空; 此处 detach 是修复生效的关键点, PP>=2 + 1F1B 下会钉住 P2P 接收缓冲。

```
def postprocess_micro_batch_func(self, output, data, forward_only, loss_function, local_cp_size=None):  
    # 出于内存效率考虑, entropy 的计算被移到 compute_log_probs (forward_only == True 时)
```

```

device = data["input_ids"].device
model_output = self.prepare_model_outputs(output, data)

if loss_function is not None:
    loss, metrics = loss_function(
        model_output=model_output, data=data, dp_group=self.get_data_parallel_group()
    )
    # 乘上 num_micro_batch: Megatron 的 pp schedule 内部会按 micro-batch 数缩放 loss
    scaled_loss = loss * data["num_micro_batch"]
else:
    assert forward_only, "forward_only must be True when loss_function is None"
    loss = torch.tensor(1.0, device=device)
    scaled_loss = loss
    metrics = {}

# loss 只保留标量；梯度通过第一个返回值 scaled_loss 流动，
# 因此这里剥离 model_output 的图是安全的
output = {"loss": loss.detach().item(), "metrics": metrics}
if forward_only or not self.engine_config.dynamic_context_parallel:
    # 必须在 loss_function(...) 之后再 detach，否则梯度会静默归零
    # (loss 值仍由 teacher/label 侧算出，loss 曲线看起来依然正常)
    # 该 dict 会进入 Megatron 的 forward_data_store，一个 micro-batch
    # 一条，直到整个 mini-batch 结束才清空；带着 grad_fn 会把 1F1B
    # 下的 P2P 接收缓冲 (create_tensor_recv_prev) 也一起钉住
    output["model_output"] = detach_tree(model_output)
if self.engine_config.dynamic_context_parallel:
    output[DCP_SAMPLE_IDS] = tu.get_non_tensor_data(data, key=DCP_SAMPLE_IDS,
        default=None)
    output[DCP_GROUP_LEADER] = tu.get_non_tensor_data(data, key=DCP_GROUP_LEADER,
        default=False)
# ... 后续为 calculate_per_token_loss 模式下的 loss_agg_mode 校验

```

tests/workers/test_engine_detach_tree_on_cpu.py

新增 CPU 回归测试：weakref + 最小 checkpoint 模型复现“detach 释放 checkpoint 保存激活”机制，断言梯度仍正常；并用 AST 静态检查 4 个引擎的 model_output 发布点全部经过 detach_tree，防止回归。

```

class _TinyCheckpointedLM(torch.nn.Module):
    """冻结 embedding + checkpoint 的训练块，模拟 PEFT/LoRA 基座模型
    在梯度 checkpoint 与 enable_input_require_grads 下的行为。"""

    def __init__(self):
        super().__init__()
        self.embed = torch.nn.Embedding(16, 8)
        self.embed.weight.requires_grad_(False) # 冻结基座
        self.proj = torch.nn.Linear(8, 8, bias=False) # 可训练部分
        self.saved_block_input = None # 指向 checkpoint 保存输入的 weakref

    def forward(self, input_ids):

```

```

x = self.embed(input_ids)
# PEFT 的 enable_input_require_grads: embedding 输出需要梯度,
# 否则 checkpoint 下梯度无法流入可训练参数
x.requires_grad_(True)
hidden = torch.utils.checkpoint.checkpoint(self.proj, x, use_reentrant=False)
self.saved_block_input = weakref.ref(x)
return hidden

def test_detach_tree_releases_the_checkpoint_saved_activation():
    module = _TinyCheckpointedLM()
    hidden = module(torch.randint(0, 16, (1, 6)))

    # 模拟后端把 model_output 发布到跨 micro-batch 存活的 dict
    model_output = detach_tree({"log_probs": hidden.sum(-1), "entropy": hidden.mean(-1)})
    assert all(value.grad_fn is None for value in model_output.values())

    loss = hidden.sum()
    loss.backward()
    assert module.proj.weight.grad is not None # 梯度通道依然完好

    saved_input = module.saved_block_input
    assert saved_input() is not None # 活图存在时输入理应存活
    del loss, hidden
    gc.collect()
    # model_output 仍然被持有 (模拟 output_lst / forward_data_store 的行为),
    # 若它没有 detach, checkpoint 保存的 embedding 输出就会存活下来 -> 泄漏
    assert saved_input() is None, (
        "checkpoint-saved block input (embedding output) survived backward: "
        "the per-micro-batch output is retaining the autograd graph"
    )

```

评论区精华

实际 review 评论为空 (仅 CLAassistant 签名检查与维护者 APPROVED), 核心讨论都浓缩在 PR body 的自述论证中:

- 为何只在插入点 detach 就足够: 梯度经第一个返回值 scaled_loss 流动 (这是 Megatron loss_func 用于 backward 的对象), dict 纯粹是报告数据, postprocess_batch_func 在所有 backward 完成后才消费它。
- 位置敏感度: detach 必须发生在 loss_function(...) 之后, 否则梯度会静默归零且 loss 曲线仍正常 (损失值由 teacher/label 侧算出), 极难排查, 这是本修复最重要的工程陷阱。
- 复现条件: 泄漏只在 PP>=2 + 训练模式 (非 forward_only) 下出现; PP=1 时 forward_backward_no_pipelining 立即 backward, 无图存活; dynamic_context_parallel=True 路径本就排除 model_output, 不适合验证。因此任何回归测试必须 PP>=2, 合入的 CPU 测试选择机制性复现而非绝对内存断言。

- 范围演进：PR 初版宣称只改 Megatron，最终合入版本统一了 4 个引擎（提交历史中最后两笔由维护者完成），FSDP 原有手写 detach 也收敛到共享 helper。
- detach 位置与梯度安全性 (correctness): 合入版本把 detach 放在 dict 构造处（`output["model_output"] = detach_tree(model_output)`），scaled_loss 保持 `requires_grad=True`，梯度链路不受影响；测试同时断言 `grad_fn is None` 与 `proj.weight.grad is not None`。
- 回归测试的可行性设计 (testing): 合入 `tests/workers/test_engine_detach_tree_on_cpu.py`，覆盖行为与源码调用点两层；未加入真实 PP>=2 集成测试，属于已知测试空白。
- 修复范围是否扩展到非 Megatron 引擎 (design): 最终合入版本统一 4 个引擎的 `model_output` 发布契约，AST 测试将 4 个调用点全部纳入防回归范围。

风险与影响

- 风险：
 - 位置依赖风险（隐蔽回归）：detach 必须位于 `loss_function(...)` 之后；若前移，梯度静默归零而 loss 曲线仍正常（loss 由 teacher/label 侧算出），难以及时发现。代码注释与测试双重锚定该约束，但未来改动仍可能触碰。
 - 语义等价性风险（偏低）：FSDP 原修复只在 `value.grad_fn is not None` 时 detach，`detach_tree` 改为对 `requires_grad=True` 张量一律 detach，对“requires_grad=True 但无 grad_fn”的叶子张量行为略有扩展；此类张量进入报告 dict 无下游消费，风险可忽略，但严格说存在行为差异。
 - 测试覆盖缺口：合入的是 CPU 机制测试 + AST 静态检查，没有真实 PP>=2 下的显存断言；AST 检查只识别 `output["model_output"] = detach_tree(...)` 与 dict 字面量两种发布模式，若引擎改为中间变量再发布会漏检（此时断言会直接失败，属于保守保护）。
 - 多引擎同步改动风险：4 个引擎文件同时修改 import 与发布点，任一语法或导入错误都会影响对应后端训练；已由 CPU 单元测试与 CI 覆盖。
- 影响：
 - 用户影响：Megatron PP>=2 的大模型训练（大 MoE、长序列、每 mini-batch 上千 micro-batch）不再因逐 micro-batch 保留 P2P 接收缓冲而 OOM；实测 last-stage rank 峰值 allocated 从 62.80 GiB (OOM) 降至 35.77 GiB，每 micro-batch 增长从 45.4 MiB 降至 1.65 MiB。此前的 `ncclUnhandledCudaError` 表象（NCCL 通过 `cudaMalloc` 绕过缓存分配器）也会随之消失。
 - 系统影响：显存驻留从 $O(\text{num_micro_batches})$ 回落到 $O(\text{pipeline_depth})$ ；四个引擎统一“发布到跨 micro-batch 结构前必须 detach”的契约，消费侧 `postprocess_batch_func` 无需改动，返回 dict 的键与数值语义不变（仅剥离 grad_fn）。
 - 团队影响：`detach_tree` 成为可复用工具，后续新引擎可参照；测试范式（weakref 机制复现 + AST 调用点检查）可推广到其他跨 micro-batch 数据结构；PR body 的归因方法论（memory history 栈归因、forward_only / PP=1 对照实验）可作为排查同类显存问题的模板。
 - 风险标记：多引擎同步修改，detach 位置依赖易静默出错，CPU 机制测试未覆盖真实 PP>=2，AST 静态检查有模式局限

关联脉络

- PR #6699 [fsdp, trainer] fix: detach model_output and loss metrics to stop per-micro-batch graph retention in actor update: 本 PR 的直接前身：修复 FSDP 引擎逐 micro-batch 图保留问题并已合并；本 PR 在 Megatron 后端跟进同类修复，并把 FSDP 的手写 detach 统一重构为共享 detach_tree。