PR #2781 完整报告

radixark/miles

Add --colocate-memory-peak-device: overlap the trainer/rollout handoff on the GPU

合并时间：2026-09-01 11:17

原文链接：http://prhub.com.cn/radixark/miles/pull/2781

执行摘要

- 一句话：gpu 模式重排交接顺序，缓解 GB300 主机内存峰值
- 推荐动作：值得精读。核心看点是 train.py 中 handoff 重排的先后顺序论证，以及 rollout_manager 细粒度 offload API 如何与 offload_rollout_level 配合达到语义自洽。由于没有单测，建议结合 #2706 的 e2e 验证上下文阅读，并关注后续是否补强测试，尤其是 cpu 模式下 onload_weights 新增 guard 的兼容性测试。

功能与动机

PR body 明确指出目标场景：在 --offload-train --offload-rollout 的 colocate 模式下，step 边界交接会同时在主机 RAM 中持有 engine 的权重镜像与 trainer 的权重备份；GB300 节点主机内存（898 GiB cgroup）小于两者之和，成为训练瓶颈。因此作者把 run_deepseek_v4.py 在 GB300 上默认切换到 gpu 模式，把峰值重叠转移到 GPU 显存。PR 同时说明该改动从 #2706 拆出，与 DeepSeek-V4 双后端工作正交，保证默认 cpu 行为完全不变。

实现拆解

该 PR 按以下 5 个步骤落地：

1. 新增参数入口（miles/utils/arguments.py）：在 cluster 参数区新增 --colocate-memory-peak-device，choices=['cpu', 'gpu']、默认 cpu。帮助文本明确两种策略：cpu 是两边各自先 offload 再 onload，host 上两份副本短暂共存；gpu 是先 onload 对方再 offload 自己，把共存移到 GPU。
2. 细粒度 offload API（miles/ray/rollout/rollout_manager.py）：原 offload(tags) 只能一次性卸载，无法表达“只卸载 KV cache、权重镜像先留在 GPU”的中间状态。新增 offload_kv()（CUDA graph 始终卸载，KV cache 按 offload_rollout_level 判断）和 offload_weights()（按 level 中是否含 weight 判断），并给 onload_weights() 加上同样的 guard，保证 offload/onload 调用语义自洽。
3. 初始化路径调整（miles/ray/placement_group.py）：create_rollout_manager 里原来的 offload.remote() 全量卸载，gpu 模式改为 offload_kv.remote()，让 engine 权重镜像在 trainer init 期间驻留 GPU，避免 init 阶段主机内存峰值。
4. 训练主循环重排（train.py）：gpu 模式下三处改动——(a) 初始化时跳过 onload_weights（权重已在 GPU）；(b) generate 之后按 offload_kv → actor_model.onload() →

offload_weights 顺序交接；(c) update_weights 之前按 actor_model.clear_memory() → onload_weights → offload_train() 顺序让位。初版曾引入 offload_grad_buffer 显式释放 grad buffer，后因全参数训练下是 no-op 而删除（提交 40b1106），改用 clear_memory()。入口处新增三组前置拦截：要求 --offload-train 与 --offload-rollout 同时开启、拒绝 critic 路径、拒绝 LoRA（仅支持全参数训练）。

5. 一键脚本自动启用（scripts/run_deepseek_v4.py）：在 _train 的 misc_args 组装处，当解析出的 --hardware == 'GB300' 时自动追加 --colocate-memory-peak-device gpu，并加注释说明原因。

测试配套：本次 5 个变更文件全部为源码，未发现对应的单元测试或 e2e 测试变更；作者依赖 #2706 的 8×GB300 DSv4 50+ step A/B 运行（jobs 2695/2703）做验证。

关键文件：

- miles/ray/rollout/rollout_manager.py（模块 推理服务；类别 source；类型 core-logic；符号 offload_kv, offload_weights, onload_weights）：新增 offload_kv / offload_weights 细粒度卸载方法，并给 onload_weights 增加 offload_rollout_level guard，是 gpu 交接模式得以成立的基础 API。
- train.py（模块 训练入口；类别 source；类型 dependency-wiring；符号 train, offload_train）：训练主循环的交接顺序在 gpu 模式下被整体重排，并加入双 offload、critic、LoRA 三组前置校验；默认 cpu 路径保持原样。
- miles/utils/arguments.py（模块 参数解析；类别 source；类型 configuration）：新增 --colocate-memory-peak-device 参数，是本次功能的配置入口，帮助文本清晰描述两种内存峰值策略的取舍。
- miles/ray/placement_group.py（模块 资源编排；类别 source；类型 core-logic；符号 create_rollout_manager）：create_rollout_manager 在 gpu 模式下改用 offload_kv 代替全量 offload，让 engine 权重镜像在 trainer 初始化期间驻留 GPU，避免 init 阶段主机峰值。
- scripts/run_deepseek_v4.py（模块 训练脚本；类别 source；类型 configuration）：在硬件为 GB300 时自动追加 --colocate-memory-peak-device gpu，是本次优化真正落到实际训练场景的开关。

关键符号：offload_kv, offload_weights, onload_weights, train, offload_train, create_rollout_manager

关键源码片段

miles/ray/rollout/rollout_manager.py

新增 offload_kv / offload_weights 细粒度卸载方法，并给 onload_weights 增加 offload_rollout_level guard，是 gpu 交接模式得以成立的基础 API。

```
# 细粒度 offload/onload 接口：原来只有一个全量 offload(tags)，无法表达
# " 只卸载 KV cache、权重镜像先留在 GPU" 这类中间状态。
# gpu 交接模式需要三步走，因此拆出 offload_kv / offload_weights，并给
# onload_weights 加上 offload_rollout_level 的 guard，保证调用语义自洽。

async def onload_weights(self):
```

```
    # 权重没有参与 offload 级别时，说明它从未被卸载，onload 直接跳过。
    if 'weight' not in self.args.offload_rollout_level:
        return
    await self.onload(tags=[GPU_MEMORY_TYPE_WEIGHTS])

async def onload_kv(self):
    # KV cache 与 CUDA graph 共用显存预算，总是成对加载。
    await self.onload(tags=[GPU_MEMORY_TYPE_KV_CACHE, GPU_MEMORY_TYPE_CUDA_
    GRAPH])

async def offload_kv(self):
    # CUDA graph 空间小，始终释放；KV cache 是否卸载由 offload_rollout_level 决定。
    tags = [GPU_MEMORY_TYPE_CUDA_GRAPH]
    if 'kv_cache' in self.args.offload_rollout_level:
        tags.append(GPU_MEMORY_TYPE_KV_CACHE)
    await self.offload(tags=tags)

async def offload_weights(self):
    # 与 onload_weights 对称：只有 weight 参与 offload 级别时才真正卸载。
    if 'weight' not in self.args.offload_rollout_level:
        return
    await self.offload(tags=[GPU_MEMORY_TYPE_WEIGHTS])
```

train.py

训练主循环的交接顺序在 gpu 模式下被整体重排，并加入双 offload、critic、LoRA 三组前置校验；默认 cpu 路径保持原样。

```
# 主循环里的交接重排（gpu 模式）。核心思路：既然主机 RAM 是最紧的预算，
# 就把两份权重副本的重叠从 host 搬到 GPU——先让一方上 GPU，再让另一方下 GPU，
# host 上任意时刻只有一份权重副本。

if args.offload_rollout:
    if args.colocate_memory_peak_device == 'gpu':
        # step 1: 只卸载 KV cache，给 trainer 腾出显存；engine 权重镜像仍驻留 GPU。
        await rollout_manager.offload_kv.remote()
        # step 2: trainer 权重备份 onload 到 GPU，此时两份权重在显存中短暂共存。
        await actor_model.onload()
        # step 3: engine 权重镜像卸载到 host，host 上只剩 trainer 那一份副本。
        await rollout_manager.offload_weights.remote()
    else:
        # cpu 默认路径保持原行为：一次性卸载全部 tag，host 上两份副本短暂共存。
        offload_tags = [GPU_MEMORY_TYPE_CUDA_GRAPH]
        if 'kv_cache' in args.offload_rollout_level:
            offload_tags.append(GPU_MEMORY_TYPE_KV_CACHE)
        if 'weight' in args.offload_rollout_level:
            offload_tags.append(GPU_MEMORY_TYPE_WEIGHTS)
        await rollout_manager.offload.remote(tags=offload_tags)

# ... 训练与保存逻辑 ...
```

```
# update_weights 之前（gpu 模式）：把 trainer 清出 GPU，让 engine 权重镜像回来。
if args.colocate_memory_peak_device == 'gpu':
    # 释放 grad buffer 与激活等训练侧显存（全参数路径下等价于清空训练态）。
    await actor_model.clear_memory()
    # engine 权重镜像回到 GPU，供 update_weights 直接写入。
    await rollout_manager.onload_weights.remote()
    # trainer 权重备份再卸载回 host，此时 host 峰值只来自单一份权重。
    await offload_train()
else:
    # cpu 默认路径：先卸载 trainer，再 onload engine 权重，顺序不变。
    await offload_train()
    if args.offload_rollout:
        await rollout_manager.onload_weights.remote()
```

评论区精华

PR 没有公开的逐行 review 评论（review_comments_count = 0），Zhichenzzz 与 maocheng23 均直接 APPROVED，claude[bot] 仅提示仓库配置了手动 review。实现过程中的关键设计决策记录在提交历史里：

- 提交 40b1106 移除 offload_grad_buffer：作者说明其“全参数训练下是 no-op，且 LoRA 路径从不 resume 或 double-pause 该区域”。说明初版曾试图在 update_weights 前显式释放 grad buffer，最终确认对目标场景无用而删除，并顺带清理了冗余 guard。
- 提交 6ec8933 明确“拒绝 LoRA，仅支持全参数训练”，结合 PR body 里“拒绝 critic 路径”的说明，gpu 模式被严格收窄为“全参数、无 critic、双 offload”场景，规避了大量未布线的分支。
- 提交 86a028c 补充解释 LoRA 场景下的 grad-buffer 行为并移除冗余 guard，说明作者在范围收窄过程中持续复查过这些分支。
- 暂无高价值评论线程

风险与影响

- 风险：
  1. 缺少测试覆盖：5 个变更文件全部为源码，没有任何对应单测；train.py 主循环的控制流重排只能靠 #2706 的 e2e 运行间接验证，后续改动容易回归。
  2. 默认路径 guard 语义变化：onload_weights() / offload_weights() 新增了 offload_rollout_level 中是否含 'weight' 的 guard，但 cpu 模式下 create_rollout_manager 仍然是全量 offload.remote()（无 tags）。如果用户显式把 offload_rollout_level 配置为不含 weight，两个路径对“权重是否曾卸载”的假设是否一致需要确认，否则可能出现权重永不 onload 的隐患。
  3. GPU 显存余量依赖：gpu 模式把峰值搬到显存，交接期两份权重同时在 GPU；这依赖 --sglang-mem-fraction-static 0.7 与 --train-memory-margin-bytes 3221225472 等余量设置，若显存预算计算有误会直接 OOM，且比 host 内存峰值更难排查。
  4. 适用范围窄：critic 与 LoRA 路径被显式拒绝，未来若要在这些场景使用需重新布线；run_deepseek_v4.py 只按硬件自动启用，其他模型脚本需要手动传参。 - 影响：对用户

：GB300 上跑 DeepSeek-V4 训练会自动切换到 gpu 模式，主机 RAM 峰值下降（避免 898 GiB cgroup 超限）；其他硬件与未显式开启的用户完全不受影响，默认 cpu 路径逐字节兼容。对系统资源：提供“主机内存峰值与 GPU 显存峰值”之间的显式权衡旋钮，交接期资源压力从 host 转移到 GPU。对团队与代码结构：rollout_manager 新增 3 个 offload/onload 语义接口，train.py 主循环多出一个分支；新参数需要文档与排障指南跟进，否则后续维护者容易混淆两种策略的适用条件。 - 风险标记：缺少测试覆盖，训练主循环控制流调整，offload_rollout_level 语义依赖，仅覆盖全参数无 critic 场景

关联脉络

- 暂无明显关联 PR