

PR #3245 完整报告

radixark/miles

fix(megatron): respect --no-load-optim when resuming a LoRA adapter

合并时间: 2026-09-15 10:55

原文链接: <http://prhub.com.cn/radixark/miles/pull/3245>

执行摘要

- 一句话: 修复 LoRA 恢复时忽略 --no-load-optim 导致旧优化器状态误载
- 推荐动作: 值得精读, 但定位是精准的小范围 bugfix。重点关注三点: 一是 load_optimizer 参数默认值选择 (True) 如何保证向后兼容; 二是 --no-load-optim 语义被严格限定为“只跳过 optimizer, 保留 iteration 与 scheduler”, 与 Megatron 原生行为对齐; 三是通过 entry 存在性守卫实现与 #3083 的无序合入, 这是一种低成本的跨 PR 兼容设计。若团队后续会扩展 LoRA 恢复功能, 可继续关注 load_lora_adapter 的参数演进。

功能与动机

PR body 指出: `_load_training_state` 只要发现 adapter 旁有 `training_state_rank*.pt` 就恢复 optimizer, 而 `load_lora_adapter` 从未接收 `args`, 因此 `--no-load-optim` 无法传入。更严重的是, 在非 `--megatron-to-hf-mode bridge` 模式下, `miles_validate_args` 会在 `--load` 没有 tracker 文件时自动设置 `no_load_optim = True`, 即用户本意是从全新 optimizer 开始, 却通过 adapter 把旧的 Adam moments 悄悄带回来。这与 Megatron 原生 `load_checkpoint` 及 `fsdp_utils/checkpoint.py` 对 `--no-load-optim` 的语义不一致, 属于静默状态污染 bug。

实现拆解

1. 调用点接线 (`miles/backends/megatron_utils/checkpoint.py`): 在 `load_checkpoint` 的 LoRA 分支中, 调用 `load_lora_adapter` 时新增 `load_optimizer=not args.no_load_optim`, 把 Megatron 既有 flag 的语义透传到 LoRA 恢复路径, 这是整个修复的入口。
2. 参数透传 (`miles/backends/megatron_utils/lora_utils.py`): `load_lora_adapter` 新增关键字参数 `load_optimizer: bool = True` (默认保持旧行为, 避免影响其他调用方), 并在 native 格式命中分支中把它传给 `_load_training_state`; `_load_training_state` 同样新增 `load_optimizer: bool = True`。
3. 门控逻辑: `_load_training_state` 中, 当 `load_optimizer=False` 时跳过 `optimizer.load_state_dict`, 仅记录 `--no-load-optim: keeping the freshly initialized optimizer` 日志, 但 `opt_param_scheduler` 和 `iteration` 仍照常恢复——这正是 Megatron 自身对 `--no-load-optim` 的定义。
4. 跨 PR 兼容守卫 (第二个 commit): 将 optimizer 恢复改为先检查 `training_state.get("optimizer") is not None` 再执行 `load_state_dict`, 避免 #3083 使 `--no-save-optim` 写出 null optimizer 后, 此处把 None 传给 `load_state_dict`; 该守卫让两个 PR 以任意顺序合入都安全。

5. 测试配套 (tests/fast/backends/megatron_utils/test_lora_utils.py) : 新增 TestLoadTrainingStateOptimizerGate, 两个用例分别验证 _load_training_state(..., load_optimizer=False) 不触达 optimizer 但恢复 scheduler/iteration, 以及 load_lora_adapter(..., load_optimizer=False) 能正确透传 flag; 全程无 GPU。验证结果 : fast 测试 318 passed / 2 skipped, pre-commit 干净, 4x H200 Qwen2.5-0.5B LoRA rank 16 GRPO 端到端手动验证两种 resume 方式符合预期。

关键文件:

- miles/backends/megatron_utils/lora_utils.py (模块 适配器层; 类别 source; 类型 core-logic; 符号 load_lora_adapter, _load_training_state) : 核心修复文件: load_lora_adapter 和 _load_training_state 新增 load_optimizer 参数, 控制 resume 时是否恢复 optimizer, 同时保留 iteration 与 LR scheduler 恢复。
- miles/backends/megatron_utils/checkpoint.py (模块 检查点; 类别 source; 类型 core-logic; 符号 load_checkpoint) : 修复的入口接线点: load_checkpoint 在 LoRA 分支中把 not args.no_load_optim 传给 load_lora_adapter, 使 flag 首次真正到达 LoRA 恢复逻辑。
- tests/fast/backends/megatron_utils/test_lora_utils.py (模块 单元测试; 类别 test; 类型 test-coverage; 符号 TestLoadTrainingStateOptimizerGate, test_no_load_optim_skips_the_optimizer_and_keeps_the_rest, test_load_lora_adapter_forwards_the_flag) : 新增 48 行测试, 覆盖 _load_training_state 门控路径和 load_lora_adapter 参数转发, 是本次修复的主要回归保障。

关键符号: load_lora_adapter, _load_training_state, load_checkpoint

关键源码片段

miles/backends/megatron_utils/lora_utils.py

核心修复文件: load_lora_adapter 和 _load_training_state 新增 load_optimizer 参数, 控制 resume 时是否恢复 optimizer, 同时保留 iteration 与 LR scheduler 恢复。

```
def load_lora_adapter(
    model: Sequence[torch.nn.Module],
    adapter_path: str,
    *,
    optimizer: Any | None = None,
    opt_param_scheduler: Any | None = None,
    load_optimizer: bool = True, # 默认 True 保持旧行为, 只有 load_checkpoint 显式传入 flag
) -> tuple[bool, int | None]:
    """Load LoRA adapter weights from a saved checkpoint into the model."""
    adapter_dir = Path(adapter_path)
    if not adapter_dir.exists():
        logger.warning(f"LoRA adapter path does not exist: {adapter_dir}")
        return False, None

    tp_rank = get_parallel_state().tp.rank
```

```

pp_rank = get_parallel_state().pp.rank

# ---- 优先尝试 Megatron-native 格式 (保 TP/PP 分片, 无需转换) ----
global_rank = dist.get_rank() if dist.is_initialized() else 0
native_path = adapter_dir / f"adapter_megatron_rank{global_rank}.pt"
if not native_path.exists():
    legacy = adapter_dir / f"adapter_megatron_tp{tp_rank}_pp{pp_rank}.pt"
    if legacy.exists():
        logger.warning(f"Using legacy tp/pp-named adapter shard {legacy}; only valid when
            EP<=TP")
        native_path = legacy
if native_path.exists():
    state_dict = torch.load(native_path, map_location="cpu", weights_only=True)
    loaded = 0
    for model_chunk in model:
        for name, param in model_chunk.named_parameters():
            if name in state_dict:
                param.data.copy_(state_dict[name].to(device=param.device))
                loaded += 1
    logger.info(f"Loaded {loaded} adapter tensors from Megatron-native checkpoint: {native_
        path}")

# 把 load_optimizer 透传给训练状态恢复, 决定是否跳过 optimizer
iteration = _load_training_state(adapter_dir, optimizer, opt_param_scheduler, load_
    optimizer)
return True, iteration

# ---- HF PEFT 格式 (尚未支持直接加载) ----
hf_path = adapter_dir / "adapter_model.bin"
if hf_path.exists():
    logger.warning(
        f"Found HF PEFT adapter at {hf_path} but direct HF PEFT loading into "
        f"Megatron is not yet supported. Please save using Megatron-native format "
        f"(adapter_megatron_tp*_pp*.pt files) for checkpoint resume."
    )
    return False, None

logger.warning(f"No adapter checkpoint found at {adapter_dir}")
return False, None

def _load_training_state(
    adapter_dir: Path,
    optimizer: Any | None,
    opt_param_scheduler: Any | None,
    load_optimizer: bool = True,
) -> int | None:
    """Restore optimizer/scheduler state saved alongside a LoRA adapter checkpoint.

```

```

load_optimizer=False 对应 --no-load-optim: 保留全新 optimizer,
但仍然恢复 LR scheduler 与 iteration, 与 Megatron 原生语义一致。
"""

if optimizer is None:
    return None

rank = dist.get_rank() if dist.is_initialized() else 0
state_path = adapter_dir / f"training_state_rank{rank}.pt"
if not state_path.exists():
    return None

# optimizer state dict 可能含非 tensor 对象（如 step 计数、param group 元数据），
# 因此这里必须完整反序列化，不能 weights_only=True。
training_state = torch.load(state_path, map_location="cpu", weights_only=False)

if not load_optimizer:
    # --no-load-optim: 明确跳过，避免旧 Adam moments 污染全新 optimizer
    logger.info("--no-load-optim: keeping the freshly initialized optimizer")
elif training_state.get("optimizer") is not None:
    # entry 存在性守卫：兼容 #3083 的 --no-save-optim 写出 null optimizer 的场景
    optimizer.load_state_dict(training_state["optimizer"])
    logger.info("Restored optimizer state from LoRA checkpoint")

if opt_param_scheduler is not None and training_state.get("opt_param_scheduler") is not
None:
    opt_param_scheduler.load_state_dict(training_state["opt_param_scheduler"])
    logger.info("Restored LR scheduler state from LoRA checkpoint")

iteration = training_state.get("iteration")
if iteration is not None:
    logger.info(f"Resuming LoRA training from iteration {iteration}")
return iteration

```

miles/backends/megatron_utils/checkpoint.py

修复的入口接线点：`load_checkpoint` 在 LoRA 分支中把 `not args.no_load_optim` 传给 `load_lora_adapter`，使 flag 首次真正到达 LoRA 恢复逻辑。

```

# Load LoRA adapter weights if available
if is_lora_enabled(args):
    adapter_path = getattr(args, "lora_adapter_path", None)
    if adapter_path is not None:
        loaded, iteration = load_lora_adapter(
            ddp_model,
            adapter_path,
            optimizer=optimizer,
            opt_param_scheduler=opt_param_scheduler,
            # 把 Megatron 的 --no-load-optim 语义透传进来：
            # False 时保留全新 optimizer，但仍恢复 iteration 与 LR scheduler
            load_optimizer=not args.no_load_optim,

```

```

    )
    if loaded:
        logger.info(f"Successfully loaded LoRA adapter from {adapter_path}")
        if iteration is not None:
            # 用 LoRA 训练状态中的 iteration 覆盖主 checkpoint 的恢复结果
            result = (iteration, result[1])
    else:
        logger.warning(
            f"LoRA is enabled and --lora-adapter-path={adapter_path} was specified, "
            f"but adapter weights could not be loaded. "
            f"Training will start with freshly initialized adapter weights."
        )

    return result

```

评论区精华

该 PR 没有实质代码评讨论：review_comments_count=0，仅有 claude[bot] 的自动提示（仓库配置为手动 review，可 @claude review）和 yueming-yuan 的直接 approve。设计要点全部体现在 PR body 与 commit message 中：作者明确说明恢复被守卫在 entry 存在性上，因此与 #3083 可任意顺序合入；并强调 --no-load-optim 仍恢复 iteration 与 LR scheduler，以对齐 Megatron 原生 load_checkpoint 和 fsdp_utils/checkpoint.py 的既有语义。

- 自动化 review 提示与 approve（无实质技术讨论）(other): 无技术争议，作者自审后合入。

风险与影响

- 风险：

1. 行为变更面：load_lora_adapter 默认参数 load_optimizer=True 保持旧行为，但 load_checkpoint 是唯一调用点且显式传入 flag，因此所有走进 load_checkpoint 的 LoRA 恢复路径都会受 args.no_load_optim 影响——这正是预期修复，但若有外部代码绕过 load_checkpoint 直接调用 load_lora_adapter，仍会默认恢复 optimizer。
2. 属性存在性风险：checkpoint.py 直接读取 args.no_load_optim，若未来有调用路径构造的 args 缺少该属性会抛 AttributeError；当前 miles_validate_args 保证其存在，但测试未覆盖该属性缺失的防御。
3. 跨 PR 依赖：第二个 commit 的 entry 守卫虽让 #3083 与 #3245 顺序无关，但如果 #3083 的写入格式后续再变化（例如 key 改名），此处仍可能静默退化。
4. 测试覆盖局限：新增测试为 fast 单测，端到端验证仅在 4x H200 上手动执行，未固化进 CI；对 LR scheduler 恢复语义的断言依赖 state_dict 直接比较，未覆盖真实 Adam moment 的格式兼容。- 影响：影响范围集中在 Megatron 后端的 LoRA 训练恢复路径：使用 --lora-adapter-path 恢复且同时指定 --no-load-optim（或由 miles_validate_args 自动设置该 flag）的用户，此前会错误恢复旧 optimizer 状态，修复后 optimizer 保持全新，但 iteration 与 LR scheduler 仍正确恢复。对不使用 LoRA 或未开启 --no-load-optim 的运行无行为变化。团队收益在于：LoRA RL 实验的恢复语义与 Megatron/FSDP 主路径统一，避免静默的优化器状态污染导致的实验结论失真。- 风险标记：恢复路径行为变更，依赖 args.no_load_optim 属性存在，跨 PR 兼容依赖

#3083, 端到端验证未固化进 CI

关联脉络

- PR #3083 (PR body 提及的关联改动) --no-save-optim 使 training_state 写入 null optimizer entry: PR body 明确说明 #3083 会让 --no-save-optim 把 optimizer 写为 null ; 本 PR 第二个 commit 通过 entry 存在性守卫保证两个改动以任意顺序合入都安全。历史 PR 数据中未提供 #3083 的完整标题, 此处仅作关联记录。
- PR #3183 [AMD][LoRA] Add a Qwen3-4B LoRA RL launcher: 同属 LoRA 功能线: 该 PR 为 AMD 平台新增 LoRA RL 启动器, 本 PR 修复 LoRA 恢复路径的 optimizer 状态语义, 二者共同完善 LoRA RL 训练与恢复体验。