

PR #3091 完整报告

radixark/miles

feat(megatron): reset optimizer state by optimizer class

合并时间: 2026-09-05 04:32

原文链接: <http://prhub.com.cn/radixark/miles/pull/3091>

执行摘要

- 一句话: 重写优化器状态重置逻辑, 按类派发并递归遍历嵌套优化器。
- 推荐动作: 值得精读。这是一个重要的 bugfix 和重构, 解决了关键训练功能的多个缺陷。
PR 的设计决策清晰: 采用基于类的派发 (而非易变的字符串参数) 和递归遍历, 使逻辑更健壮、可维护。新代码的“快速失败”策略 (对未知优化器 / 状态抛出异常) 是良好的防御性编程实践。配套的测试桩更新和详尽的单元测试为变更提供了强有力的质量保证。关注点在于其对 Megatron 内部优化器结构的隐式依赖, 以及最终在真实硬件上的端到端验证情况。

功能与动机

PR #2769 的原始实现存在问题, 其 `dist_muon` 分支因 Megatron 的 `validate_args` 在训练开始前将优化器重命名为 `muon` 而永远无法执行; 同时, 它拒绝了 `--optimizer-offload-fraction` 低于 1.0 时的 `HybridDeviceOptimizer` (AMD DSv4 配方使用 0.75); 此外, 新引入的顶层导入破坏了 `test_model_initialize.py` 中的桩 (stub), 导致基础阶段的测试无法通过。本 PR 旨在彻底解决这些问题, 提供一个健壮的重置实现。

实现拆解

1. 新增核心重置模块(`miles/backends/megatron_utils/optimizer_state_reset.py`): 创建独立模块, 实现基于优化器类的派发重置逻辑。核心函数 `reset_optimizer_states` 接收顶层优化器, 通过 `_leaves` 递归遍历 `chained_optimizers`、`sub_optimizers` 和 `.optimizer` 属性以到达所有叶子优化器 (如 `Adam`, `CPUAdam`, `TensorParallelMuon`, `TensorParallelAdaptiveMuon`)。`_history_keys` 根据每个叶子优化器的类返回需要清零的状态键 (如 `Adam` 的 `exp_avg`, `exp_avg_sq`; `Muon` 的 `momentum_buffer`; `Adaptive Muon` 的 `momentum_buffer` 和 `moment2_buffer`)。`_reset` 函数负责将状态历史清零, 并跳过 `master_param` 键。`UnsupportedOptimizerState` 异常用于在遇到未知优化器或状态键时快速失败。为了兼容没有 `emerging_optimizers` 模块的 NPU 环境, 对 `Muon` 类的导入采用了惰性导入 (在匹配到 `Muon` 类型时才导入)。
2. 在模型初始化中集成(`miles/backends/megatron_utils/model.py`): 在 `train` 函数中, 当 `--reset-optimizer-states` 启用时, 调用新的 `reset_optimizer_states(optimizer)` 函数替代旧的内联循环, 并将 `print` 语句改为 `logger.info`。
3. 更新测试桩以兼容(`tests/fast/backends/megatron_utils/test_model_initialize.py`): 由于新模块引入了新的顶层导入, `test_model_initialize.py` 中用于模拟 Megatron 环境的桩需要更新。在 `megatron.core.optimizer` 桩中添加了 `Adam` 和 `CPUAdam`; 新增

megatron.core.optimizer.emerging_optimizers 桩以提供 TensorParallelMuon；在 megatron.core.utils 桩中添加了 unwrap_model；在 miles.backends.megatron_utils.model_provider 桩中添加了 LinearForLastLayer。这些更新确保了基础测试 test_initialize_does_not_step_scheduler_restored_from_checkpoint 在新的导入链下仍能通过。

4. 新增全面测试(tests/fast/backends/megatron_utils/test_optimizer_state_reset.py)：为新模块添加了详尽的单元测试，覆盖：嵌套优化器树（dist_muon 场景）的重置、TensorParallelAdaptiveMuon 双缓冲区重置、在缺少 emerging_optimizers 模块时 Adam 的重置（NPU 兼容性）、HybridDeviceOptimizer（部分卸载）的 CPU 和 GPU 子优化器重置、步数状态为张量或组级别的重置、空优化器跳过、首次步数前的无操作重置，以及对未知状态键和未知优化器类的拒绝。

关键文件：

- miles/backends/megatron_utils/optimizer_state_reset.py（模块 优化器状态重置；类别 source；类型 core-logic；符号 UnsupportedOptimizerState, reset_optimizer_states, _leaves, _history_keys）：新增的核心模块，实现了按优化器类派发、递归遍历的重置逻辑，是本 PR 的功能核心。
- tests/fast/backends/megatron_utils/test_optimizer_state_reset.py（模块 重置逻辑测试；类别 test；类型 test-coverage；符号 stepped_adamw, GroupStepAdamW, muon_with_momentum, wrap）：为新模块提供全面的单元测试，覆盖了嵌套结构、各种优化器类型、NPU 兼容性以及错误处理场景。
- miles/backends/megatron_utils/model.py（模块 模型训练；类别 source；类型 entrypoint）：集成了新的重置逻辑，替换了旧的内联循环，是变更在实际训练流程中的集成点。
- tests/fast/backends/megatron_utils/test_model_initialize.py（模块 初始化测试；类别 test；类型 test-coverage）：更新了测试桩以兼容新模块的导入，确保基础测试在新的代码结构下仍然通过。

关键符号：reset_optimizer_states, _leaves, _history_keys, _reset, _reset_step

关键源码片段

miles/backends/megatron_utils/optimizer_state_reset.py

新增的核心模块，实现了按优化器类派发、递归遍历的重置逻辑，是本 PR 的功能核心。

```
# miles/backends/megatron_utils/optimizer_state_reset.py
# 核心重置逻辑：根据优化器类型派发，并递归遍历嵌套结构
```

```
def reset_optimizer_states(optimizer) -> None:
    """入口函数：遍历所有叶子优化器并重置其状态。"""
    for leaf in _leaves(optimizer):
        _reset(leaf, _history_keys(leaf))
```

```
def _leaves(optimizer) -> Iterator[torch.optim.Optimizer]:
    """递归下降 Megatron 的包装器（如 ChainedOptimizer,
```

```

HybridDeviceOptimizer), 找到持有状态的 torch 优化器。"""
if optimizer is None:
    return
# 优先检查并遍历 'chained_optimizers' 和 'sub_optimizers' 属性 (Megatron 常见包装器)
for children in ("chained_optimizers", "sub_optimizers"):
    if hasattr(optimizer, children):
        for child in getattr(optimizer, children):
            yield from _leaves(child)
    return
# 检查通用的 '.optimizer' 属性 (如 Float16 包装器)
if hasattr(optimizer, "optimizer"):
    yield from _leaves(optimizer.optimizer)
else:
    # 到达叶子节点 (如 Adam, TensorParallelMuon)
    yield optimizer

def _history_keys(optimizer) -> frozenset[str]:
    """根据优化器实例的类, 返回其状态中需要被清零的历史键。
    此处使用 isinstance 进行类派发, 而非 args.optimizer 字符串 (后者在 Megatron
    内部会被重写)。"""
    # Adam 类路径 (CPUAdam 是其子类)
    if isinstance(optimizer, (Adam, CPUAdam)):
        return frozenset({"exp_avg", "exp_avg_sq"})
    # Muon 类路径: 必须在此处惰性导入, 以避免在没有 emerging_optimizers 模块的 NPU
    环境下引入依赖。
    from megatron.core.optimizer.emerging_optimizers import TensorParallelAdaptiveMuon,
    TensorParallelMuon
    # 匹配子类 TensorParallelAdaptiveMuon (它有 moment2_buffer)
    if isinstance(optimizer, TensorParallelAdaptiveMuon):
        return frozenset({"momentum_buffer", "moment2_buffer"})
    # 匹配父类 TensorParallelMuon
    if isinstance(optimizer, TensorParallelMuon):
        return frozenset({"momentum_buffer"})
    # 未知优化器, 快速失败
    raise UnsupportedOptimizerState(f"no reset schema for {type(optimizer).__name__}")

def _reset(optimizer, history: frozenset[str]) -> None:
    """执行重置: 遍历参数组 and 状态, 清零历史键并处理步数。"""
    for group in optimizer.param_groups:
        _reset_step(group)
    for state in optimizer.state.values():
        # 验证状态键是否都在已知范围内 (历史键 + 'step' + 'master_param'), 否则拒绝
        unknown = set(state) - history - {"step", "master_param"}
        if unknown:
            raise UnsupportedOptimizerState(f"unknown state keys {sorted(unknown)} on
            {type(optimizer).__name__}")
        _reset_step(state)
        # 将历史键对应的张量清零
        for key in history & set(state):

```

```
state[key].zero_()
```

评论区精华

Codex 自动审查提出了两个主要问题 (P1 和 P2) :

1. P1 (NPU 兼容性): 指出顶层无条件导入 `megatron.core.optimizer.emerging_optimizers` 会破坏没有该模块的 NPU 运行。结论: 已在最终提交中解决, 通过将 `Muon` 类的导入移到 `_history_keys` 函数内部, 使其只在匹配到 `Muon` 类型优化器时才执行导入, 避免了 Adam 路径上的 `ModuleNotFoundError`。
 2. P2 (Adaptive Muon 状态遗漏): 指出 `TensorParallelAdaptiveMuon` 有额外的 `moment2_buffer`, 但初始实现未处理, 可能导致崩溃。结论: 已在最终提交中解决, `_history_keys` 函数现在将 `TensorParallelAdaptiveMuon` 置于 `TensorParallelMuon` 之前进行匹配, 并返回包含 `momentum_buffer` 和 `moment2_buffer` 的键集。最终, PR 获得了人工批准 (LGTM in human review) 。
- NPU 兼容性回归 (P1) (correctness): 已解决。将 `Muon` 类的导入移至 `_history_keys` 函数内部, 使其仅在匹配到 `Muon` 类型优化器时执行, 避免了 Adam 路径上的 `ModuleNotFoundError`。
 - Adaptive Muon 状态键遗漏 (P2) (correctness): 已解决。在 `_history_keys` 中将 `TensorParallelAdaptiveMuon` 置于 `TensorParallelMuon` 之前匹配, 并返回包含 `momentum_buffer` 和 `moment2_buffer` 的键集。
 - 测试桩更新以支持新导入 (testing): 已解决。在桩模块 `megatron.core.optimizer` 和 `megatron.core.utils` 等中补充了新导入所需的符号 (`Adam`, `CPUAdam`, `TensorParallelMuon`, `unwrap_model` 等) 。

风险与影响

- 风险:
 1. 回归风险: 核心逻辑重写, 虽然旨在修复问题, 但新的递归遍历和基于类的派发可能引入新的边界情况 (如未知的包装器属性或嵌套结构), 尽管测试覆盖了主要路径。
 2. 兼容性风险: 依赖于 `megatron.core.optimizer` 下具体优化器类的 `isinstance` 检查和状态键定义, 若 Megatron 未来版本修改了这些类名或状态键, 可能导致重置失败或错误。
 3. 测试覆盖风险: 尽管新增了全面单元测试, 但端到端验证 (如 H200 上的 Qwen3-4B 运行) 是在 PR 描述中提及的, 未在 CI 中作为自动化测试, 对于这种训练关键功能的变更, 自动化的集成测试覆盖更为稳妥。- 影响: 影响范围: 仅影响 Megatron 训练后端在使用 `--reset-optimizer-states` 标志时的行为。影响程度: 高。这是一个关键训练功能的正确性修复, 直接影响从检查点恢复或特定实验配置 (如使用 `dist_muon` 或部分 CPU offload 的 Adam) 下的训练稳定性。变更修复了之前可能静默失败或抛出错误的场景, 使其如预期工作。同时, 更新了测试基础设施以支持新代码的导入, 确保了现有测试集的稳定性。
- 风险标记: 核心路径变更, 优化器兼容性, 测试桩更新

关联脉络

- PR #2769 Reset optimizer state by optimizer class: 本 PR 是其修正版本，修复了其中存在的分支无法运行、拒绝 HybridDeviceOptimizer 和测试兼容性问题。
- PR #1990 Move sglang config resolution into sglang_config.py: 展示了仓库近期对模块职责分离的重构趋势，本 PR 新增的独立 optimizer_state_reset.py 模块遵循了类似的设计哲学，将核心逻辑从训练主流程中解耦。