

PR #2740 完整报告

radixark/miles

refactor(update-weight): backend-neutral HfWeightIterator API

合并时间: 2026-09-03 12:14

原文链接: <http://prhub.com.cn/radixark/miles/pull/2740>

执行摘要

- 一句话: 重构权重更新迭代器为后端无关 API, 统一基础模型与 LoRA 导出接口。
- 推荐动作: 这是一个高质量、设计清晰的架构重构 PR。它成功地将分散的逻辑收敛到后端无关的抽象层, 为未来扩展打下了坚实基础。值得精读, 特别是 `hf_weight_iterator.py` 的基础类设计、`MegatronHfWeightIteratorBase` 的 PP 收集实现, 以及消费者代码如何被简化。关注点: `resolve_placement` 的简洁设计、`get_hf_lora_weights` 模板方法中的验证守卫、以及 `_gather_pp_full_adapter` 的具体广播实现。

功能与动机

本 PR 的动机源于 issue #1360 (权重更新重构), 目标是为全模型和 LoRA 权重导出提供统一的 API 接口, 并定义清晰的放置协议 (`WeightUpdatePlacement`)。这旨在解耦具体后端实现, 使得未来添加新的权重更新后端 (如 FSDP) 只需实现两个抽象钩子 (`iter_hf_base_weights` 和 `_export_lora_named_tensors`), 即可继承统一的分桶、验证和 PP 组装逻辑。

实现拆解

1. 引入后端无关基础层: 在 `miles/backends/training_utils/weight_update/hf_weight_iterator.py` 中新增核心抽象类 `HfWeightIteratorBase`, 定义了 `iter_hf_base_weights` (流式基础权重) 和 `get_hf_lora_weights` (完整 LoRA 导出) 两个主要接口。同时引入了 `WeightUpdatePlacement` 数据类 (`gather_pp/tp/ep`) 和 `resolve_placement` 函数, 用于协商协议要求的放置策略与迭代器强制的放置策略。`get_hf_lora_weights` 作为模板方法, 内嵌了空权重和缺少 LoRA 名称的通用验证逻辑。
2. 适配并重构 Megatron 实现: 新增 `miles/backends/megatron_utils/update_weight/hf_weight_iterator.py` 作为 Megatron 后端的共享基类 `MegatronHfWeightIteratorBase` 和工厂函数 `get_hf_weight_iterator`。该基类强制 `gather_pp=True`, 并实现了 `_export_lora_named_tensors` 方法, 其核心是调用具体的 PP 本地导出钩子 (`_export_pp_local_lora`) 后, 通过 `_gather_pp_full_adapter` 函数完成 PP 维度的完整适配器收集。原有的 `hf_weight_iterator_base.py` (含静态工厂 `create` 方法) 被删除。
3. 迁移具体迭代器实现: `HfWeightIteratorBridge` 和 `HfWeightIteratorDirect` 均改为继承自新的 `MegatronHfWeightIteratorBase`。它们的 `get_hf_weight_chunks` 方法被拆分并重写为 `iter_hf_base_weights` (仅处理基础权重) 和 `_export_pp_local_lora` (处理 LoRA 导出)。原位于两个调用点 (`update_weight_from_tensor` 和分布式混合器) 的 LoRA 积累循环、

PP 组装 (`_pp_assemble_full_adapter`)、适配器槽位处理 (`expose_adapter_slot`) 和守卫逻辑，均被移入迭代器的实现中。

4. 更新消费者与简化调用：`update_weight_from_tensor.py` 和 `update_weight_from_distributed/mixin.py` 被简化，直接使用新的工厂函数 `get_hf_weight_iterator` 和 `get_hf_lora_weights` API，移除了本地的 PP 组装、积累循环和守卫代码。
5. 测试覆盖与配套：新增 `tests/fast/backends/training_utils/weight_update/test_hf_weight_iterator.py` 测试新基础 API 的放置解析和 LoRA 导出模板方法。更新了 `test_lora_weight_sync_validation.py`、`test_lora_hf_weight_iterator.py` 等测试文件，以适配新 API 并将验证逻辑的测试集中到基础类测试中。

关键文件：

- `miles/backends/training_utils/weight_update/hf_weight_iterator.py`（模块 权重更新；类别 `source`；类型 `core-logic`；符号 `WeightUpdatePlacement`, `resolve_placement`, `HfWeightIteratorBase`, `get_hf_lora_weights`）：引入后端无关的核心抽象 `HfWeightIteratorBase` 和放置协议 `WeightUpdatePlacement`，是本次重构的基础和新契约定义。
- `miles/backends/megatron_utils/update_weight/hf_weight_iterator.py`（模块 Megatron 权重更新；类别 `source`；类型 `core-logic`；符号 `MegatronHfWeightIteratorBase`, `get_hf_weight_iterator`, `_export_lora_named_tensors`, `_gather_pp_full_adapter`）：定义了 Megatron 后端的具体基类 `MegatronHfWeightIteratorBase`、工厂函数 `get_hf_weight_iterator` 和关键的 PP 维度适配器收集函数 `_gather_pp_full_adapter`，是新架构在具体后端上的落地实现。
- `miles/backends/megatron_utils/update_weight/hf_weight_iterator_bridge.py`（模块 Megatron 权重更新；类别 `source`；类型 `core-logic`；符号 `HfWeightIteratorBridge`, `iter_hf_base_weights`, `_export_pp_local_lora`, `_export_current_adapter`）：展示了 Bridge 迭代器如何适配新接口：将旧 `get_hf_weight_chunks` 拆分为 `iter_hf_base_weights` 和 `_export_pp_local_lora`，并复用 `MegatronHfWeightIteratorBase` 的 PP 收集逻辑。

关键符号：`iter_hf_base_weights`, `_export_lora_named_tensors`, `get_hf_lora_weights`, `_export_pp_local_lora`, `_gather_pp_full_adapter`, `get_hf_weight_iterator`, `resolve_placement`

关键源码片段

`miles/backends/training_utils/weight_update/hf_weight_iterator.py`

引入后端无关的核心抽象 `HfWeightIteratorBase` 和放置协议 `WeightUpdatePlacement`，是本次重构的基础和新契约定义。

```
"""Backend-neutral API for streaming training-side weights as HF-named tensors."""
```

```
import dataclasses
from abc import ABC, abstractmethod
from typing import ClassVar
```

```

import torch
from miles.utils.lora import is_lora_weight_name

@dataclasses.dataclass(frozen=True)
class WeightUpdatePlacement:
    """Which training-side parallel dims the iterator gathers before yielding.

    A gathered dim: every yielded tensor is full along that dim on every rank
    of that dim's group. A non-gathered dim: each rank yields its own shard of
    the param set.
    """

    gather_pp: bool
    # Always gathered today; explicit so a future protocol can relax them.
    gather_tp: bool = True
    gather_ep: bool = True

def resolve_placement(required: WeightUpdatePlacement, forced: WeightUpdatePlacement |
None) -> WeightUpdatePlacement:
    """Join of the protocol's required placement and the iterator's forced one:
    a dim is gathered if either side gathers it."""
    if forced is None:
        return required
    return WeightUpdatePlacement(
        gather_pp=required.gather_pp or forced.gather_pp,
        gather_tp=required.gather_tp or forced.gather_tp,
        gather_ep=required.gather_ep or forced.gather_ep,
    )

class HfWeightIteratorBase(ABC):
    """Streams a training model's weights as HF-named tensors.

    Collective: every training rank must drive the iterators in lockstep.
    Yielded tensors are freshly allocated per bucket and stay valid while the
    caller holds a reference.
    """

    # Placement this implementation can produce; None means any.
    forced_placement: ClassVar[WeightUpdatePlacement | None] = None

    def __init__(self, args, model, *, placement, model_name, quantization_config) -> None:
        self.args = args
        self.model = model
        self.placement = placement
        self.model_name = model_name
        self.quantization_config = quantization_config

```

```

@abstractmethod
def iter_hf_base_weights(self, weights, *, materialize=True):
    """Base model weights as size-bounded buckets of HF-named GPU tensors;
    atomic update groups are never split across buckets.

    ``weights``: backend-native named weights to read; None reads the live
    model parameters. ``materialize=False`` joins every collective but
    yields nothing.
    """

def get_hf_lora_weights(self, adapter=None) -> list[tuple[str, torch.Tensor]]:
    """The complete adapter in HF PEFT naming (lora_A/lora_B), fully
    gathered, as one list. ``adapter`` selects a multi-LoRA slot; None is
    the single-LoRA adapter. Collective.
    """

    named_tensors = self._export_lora_named_tensors(adapter)
    if not named_tensors:
        raise RuntimeError(
            f"LoRA weight sync failed: the weight iterator produced zero chunks"
            f"{f' for adapter {adapter!r}' if adapter is not None else ''}. "
            "No adapter weights were sent to the rollout engine. This usually means "
            "the Megatron-Bridge or SGLang version is incompatible."
        )
    if not any(is_lora_weight_name(name) for name, _tensor in named_tensors):
        raise RuntimeError(
            "LoRA weight sync failed: chunk contains no LoRA weights "
            "(no lora_A/lora_B names found). Check weight iterator configuration."
        )
    return named_tensors

@abstractmethod
def _export_lora_named_tensors(self, adapter) -> list[tuple[str, torch.Tensor]]:
    """Backend hook: the complete adapter as HF-named tensors, fully
    gathered along every parallel dim."""

```

评论区精华

PR 本身没有评论区讨论，从提交历史（14 个提交）可以看到这是一个经过精心设计的增量式重构。提交信息清晰地展示了设计演进：从引入基础 API，到调整放置表达式，再到迁移文件路径、定义 PP 收集原语，最后将收集逻辑保持在 Megatron 侧。这表明作者在重构过程中对模块边界、职责划分和扩展性进行了深入的思考。

- 暂无高价值评论线程

风险与影响

- 风险：回归风险：本次重构为大规模代码移动和接口重写，虽声明“无语义变更”，但涉及权重同步的核心路径。风险点包括： 1) `resolve_placement` 的合并逻辑是否在所有场景下正确； 2) LoRA 导出守卫（空导出、无 LoRA 名称）从消费者移入迭代器后，其触发条件和错

误信息是否保持一致； 3) 在分布式路径下，PP 收集行为是否完全复制原逻辑。 架构风险：当前所有 Megatron 迭代器的 `materialize` 参数仅通过 `assert` 校验，真正的非物化迭代器支持留待未来迁移。这属于已知的过渡状态。 兼容性风险：移除了旧的 `HfWeightIteratorBase` 类及其 `create` 工厂，任何直接导入该模块或依赖 `is_lora_weight_name` 旧路径（从 `miles/backends/megatron_utils/lora_utils`）的代码都将中断，需要更新导入。新路径为 `miles/utils/lora.py`。

- 影响：系统影响：权重更新机制是训练与推理引擎交互的枢纽，影响所有模型（包括多 LoRA）的权重同步。本次重构旨在提高代码的可维护性和可扩展性，为后续支持新硬件 / 后端铺平道路。团队影响：为后续权重更新重构（分布式路径迁移、FSDP 支持）建立了清晰的基础架构。开发者在添加新后端或修改权重导出逻辑时，将面对更统一、更少重复的代码。用户影响：无直接功能变更，用户无感知。重构后的系统更易于维护和扩展。
- 风险标记：核心路径重构，架构重大调整，需要集成测试验证，未来迁移点已声明（`materialize` 断言）

关联脉络

- PR #2875 Remove redundant inkling atomic update groups: 同属于 Megatron 权重更新模块的重构，清理了 Inkling 模型冗余的原子更新组，与本 PR 的权重同步逻辑简化有潜在关联。
- PR #3089 fix colocate lora duplicated cpu param backup by keeping TMS only: 涉及 colocate LoRA 权重同步的优化，是权重更新流程的组成部分，与本 PR 重构的 LoRA 导出路径可能共享底层机制。
- PR #2692 fix(disk-delta): reject noncanonical tensor layouts: 修复磁盘增量更新中的张量布局校验，属于 `update_weight` 子系统的 bugfix，与本 PR 重构的权重流管道在概念上相关。
- PR #2653 Stream optimizer main initialization to NVMe: 关于权重 / 参数流式处理的优化（NVMe 流式初始化），展示了仓库在权重更新性能方面的持续投入，与本 PR 提升架构扩展性的目标互补。