PR #2692 完整报告

radixark/miles

fix(disk-delta): reject noncanonical tensor layouts

合并时间：2026-09-01 01:56

原文链接：http://prhub.com.cn/radixark/miles/pull/2692

执行摘要

- 一句话：disk-delta 基线校验张量 dtype/ 形状，拒绝非规范布局
- 推荐动作：值得精读。核心设计是“规范化校验前置 + 集合通信错误同步”：在分布式训练中，source rank 的校验错误不能立即抛出（会中断 collectives），通过 all_gather_object 聚合后再统一失败，是一个可复用的模式。dtype 映射表的做法（safetensors 不暴露 torch dtype 编码器）也值得注意。建议关注点：移除 fallback 后的兼容性边界、未来新 dtype 的扩展点、以及错误聚合的通信开销。

功能与动机

PR body 解释：disk-delta 在原始字节上编码变更，并在规范 safetensors header 下应用。如果张量缺失或 dtype/shape 不匹配，字节级 XOR 无法修复：它要么在 checkpoint 中没有目标，要么使 header 错误解释负载。在基线捕获时失败可以防止发布损坏或不完整的更新流。原实现遇到 checkpoint 缺失张量时会 fallback 到 trainer 权重（self._snapshot[name] = tensor...），这掩盖了布局不一致，可能导致后续 delta 应用到错误字节。

实现拆解

1. 读取层布局化（miles/utils/disk_delta.py）：_tensor_locations 的返回值从 (path, offset, nbytes) 扩展为 (path, offset, nbytes, dtype, shape)，make_tensor_reader 返回的 read 新增 expected_dtype、expected_shape 关键字参数；不一致时抛 ValueError 并同时展示 checkpoint 与 trainer 两侧值。这给了上层语义校验的钩子，且不改变既有调用方（参数可选）。
2. dtype 代码映射（delta.py）：新增 _SAFETENSORS_DTYPE_BY_TORCH_DTYPE 与 _safetensors_dtype()。safetensors 库不公开 torch dtype 编码器，但 disk-delta 需要精确代码来比对 header，所以手写映射并覆盖了 float8、uint64/uint32/uint16 等条件存在类型；未知 dtype 直接 ValueError。
3. 基线校验重构（delta.py::_capture_baseline）：reader 只在 source rank 构建；seed_bucket 对每个张量按 trainer 的 dtype 与 shape 读取 baseline，KeyError 转成“缺失”错误，并追加字节数一致性检查。所有错误先存 local_error，等 _for_each_hf_bucket 的 gather 全部结束后用 all_gather_object 汇总各 rank 的错误，任何一个 rank 失败则全员抛 RuntimeError，避免在集合通信中途退出导致死锁。
4. 标量张量支持（delta.py::encode_bucket）：展平顺序从 .view(torch.uint8).reshape(-1) 改为 .reshape(-1).view(torch.uint8)，保证 0 维标量也能得到 1 字节序列。

5. 测试配套：新增 tests/fast/utils/test_disk_delta.py::test_tensor_reader_validates_declared_layout，覆盖正确读取、错误 dtype、错误 shape 三种情况；该测试只覆盖 reader 层，没有覆盖跨 rank 错误传播路径。

关键文件：

- miles/backends/megatron_utils/update_weight/update_weight_from_distributed/delta.py（模块 权重同步；类别 source；类型 core-logic；符号 _SAFETENSORS_DTYPE_BY_TORCH_DTYPE, _safetensors_dtype, _capture_baseline, seed_bucket）：disk-delta 的主同步实现，本次修改集中在这里：新增 dtype 映射、改造 _capture_baseline 的基线校验与跨 rank 错误同步、调整 encode_bucket 的展平顺序。
- miles/utils/disk_delta.py（模块 增量工具；类别 source；类型 core-logic；符号 _tensor_locations, make_tensor_reader, read）：提供 layout-aware 的 safetensors 原始字节读取器，是 delta 机制直接操作字节的基础设施；本次扩展了索引与读取时的 dtype/shape 校验。
- tests/fast/utils/test_disk_delta.py（模块 测试；类别 test；类型 test-coverage；符号 test_tensor_reader_validates_declared_layout）：新增对 reader 布局校验的单元测试，覆盖 dtype 与 shape 两种错误路径，是本次修复的回归保障。

关键符号：_safetensors_dtype, _capture_baseline, seed_bucket, encode_bucket, _tensor_locations, read, test_tensor_reader_validates_declared_layout

关键源码片段

miles/utils/disk_delta.py

提供 layout-aware 的 safetensors 原始字节读取器，是 delta 机制直接操作字节的基础设施；本次扩展了索引与读取时的 dtype/shape 校验。

```
def _tensor_locations(ckpt_dir: str) -> dict[str, tuple[str, int, int, str, tuple[int, ...]]]:
    """索引每个张量的字节范围以及 safetensors 头部声明的布局信息。

    返回值从原来的 (file, offset, nbytes) 扩展为 (file, offset, nbytes, dtype, shape)。
    这样上层在读取原始字节前就能核对 trainer 张量与 checkpoint 声明是否一致。
    """
    locations: dict[str, tuple[str, int, int, str, tuple[int, ...]]] = {}
    for path in glob.glob(os.path.join(ckpt_dir, "*.safetensors")):
        with open(path, "rb") as f:
            (header_len,) = struct.unpack("<Q", f.read(8))
            header = json.loads(f.read(header_len))
        for name, info in header.items():
            if name == "__metadata__":
                continue
            begin, end = info["data_offsets"]
            locations[name] = (
                path,
                8 + header_len + begin, # 8 字节头部长度字段 + header + data_offsets 起点
                end - begin,
                info["dtype"],
```

```
                tuple(info["shape"]),
            )
    return locations


def make_tensor_reader(ckpt_dir: str):
    """索引一次 header，返回 layout-aware 的原始字节读取器。"""
    locations = _tensor_locations(ckpt_dir)

    def read(
        name: str,
        *,
        expected_dtype: str | None = None,
        expected_shape: tuple[int, ...] | None = None,
    ) -> np.ndarray:
        path, offset, nbytes, dtype, shape = locations[name]
        # 只要任一期望值与 checkpoint 声明不一致，就拒绝读取，避免把字节错位问题带到后续 diff。
        if (expected_dtype is not None and dtype != expected_dtype) or (
            expected_shape is not None and shape != expected_shape
        ):
            raise ValueError(
                f"Checkpoint tensor {name!r} has dtype={dtype}, shape={shape}; "
                f"trainer emitted dtype={expected_dtype}, shape={expected_shape}"
            )
        with open(path, "rb") as f:
            f.seek(offset)
            return np.frombuffer(f.read(nbytes), dtype=np.uint8)

    return read
```

评论区精华

该 PR 没有实质技术讨论：0 条评论、0 条 review 评论。仅有的记录是 claude[bot] 自动说明这是 fork PR、自动 review 被禁用，以及 yueming-yuan 的批准（无正文）。因此关于“移除 fallback 是否安全”“错误传播是否需要覆盖非 source rank”等设计问题没有公开交锋，只能从代码和 PR body 推断结论。

- 自动 review 在 fork PR 上被禁用 (question): 未触发额外自动审查，由人类维护者 yueming-yuan 直接批准。
- 维护者批准 (other): 批准合并。

风险与影响

- 风险：
  1. 兼容性回归：移除缺失张量回填后，任何 trainer 权重与 canonical checkpoint 布局不一致（哪怕是 megatron->HF 转换修剪 vocab-padding 的合法场景）都会让首次同步直接失败。代码注释里仍提到 round-trip trims vocab-padding rows，这意味着使用 disk-delta 的模型必须保证 name 集合完全一致。

2. dtype 映射完整度：_SAFETENSORS_DTYPE_BY_TORCH_DTYPE 覆盖了常用类型与 float8/uint 系列，但若出现未覆盖 dtype（如 complex128、quint8 或新 float8 变体），_safetensors_dtype 抛错虽比错位安全，但会阻止训练，需要随新 dtype 同步扩展映射表。
3. 错误传播机制：all_gather_object 增加一次全局通信；且 next() 只取第一个失败 rank 的信息，可能掩盖多 rank 的不同错误，但足够定位首个故障。
4. 字节数校验边界：基于 numel * element_size 的字节数相等无法发现 padding/alignment 差异，若 checkpoint 有额外对齐可能误报；safetensors 标准布局下通常一致，但值得留意。
5. 影响范围：delta.py 位于 megatron 权重更新关键路径，改动回归会影响整个 PPO/ 多引擎同步流程。 - 影响：影响范围集中在使用 --update_weight_disk_dir 的 disk-delta 权重同步用户。正面影响：从源头阻止损坏 delta 的发布，提升更新流完整性，减少字节错位导致的隐性推理错误。负面影响：对 checkpoint 与训练权重布局不完全一致的用户，首次同步会直接失败而不是用“近似”权重继续，需要先统一 checkpoint 命名与布局。代码结构上仅 2 个源码文件和 1 个测试文件，但涉及 megatron 更新权重链路，团队回归测试时需要覆盖 disk-delta 的端到端同步场景。 - 风险标记：核心路径变更，移除 fallback 行为，跨 rank 错误同步，dtype 映射覆盖完整性

关联脉络

- PR #2764 perf(megatron): keep policy logits in model precision: 同属 megatron 后端权重 / 精度一致性维护方向，与本次 disk-delta 布局校验一起保障权重同步准确。
- PR #2818 fix(megatron): keep SFT logits in model precision: 同属 megatron 后端正确性修复，改动范围也在 megatron_utils 区域，可视为同一维护方向的后续巩固。