

PR #2839 完整报告

radixark/miles

[multi-lora v2 0/7] Remove the multi-LoRA v1 implementation

合并时间: 2026-09-16 16:24

原文链接: <http://prhub.com.cn/radixark/miles/pull/2839>

执行摘要

- 一句话: 删除 multi-LoRA v1 实现, 仅保留槽位原语
- 推荐动作: 值得精读, 但重点不是 5000 行删除本身, 而是删除与保留的边界如何划定: 建议先读 miles/backends/megatron_utils/lora/slots.py (保留原语的新家, 含跨 rank shard 写入者选举与 expert-axis-safe 的 rank 切分) 与 miles/utils/multi_lora.py (收缩后的公共工具面), 再回看 miles/ray/multi_lora/registry.py、backend.py 被删掉的注册期校验与状态机, 逐条确认 v2 (#2844 的 miles/tinker/core/scheduler.py/service.py、#2840 的 slot actor) 是否有等价实现。三个最值得关注的设计决策: 一是槽位原语与控制面彻底解耦 (docstring 里显式声明不掺控制面 / 数据面逻辑); 二是注册期 fail-fast 校验 (data 路径、reward 配置、batch 形状整除) 整体消失后由谁承担; 三是 per-slot LR 调度与 adapter YAML 契约被删, v2 是否真的不再需要 per-adapter 调度。阅读时注意 head 只有一个 squash commit, MoE 专家路径的恢复要直接核对 head_sha 958e1ca 的文件内容。

功能与动机

按 PR body 的表述, 动因是 v2 要转向以 Tinker 命令执行器为中心的槽位化多租户形态, 而 v1 是另一套完整体系: “Remove the dataset-driven multi-LoRA v1 driver, per-adapter data sources and scheduling, adapter controller, and legacy checkpoint flow.” 若两套控制面 / 数据面并存, v2 的改造无法收敛, 所以选择一次性删除; “Retain the slot primitives under miles/backends/megatron_utils/lora/ ... The Tinker command executor builds on these primitives in #2840” 划定了保留边界。作者在 issue 评论中进一步说明保留清单被修正过: MoE expert adapter 的 build-time 部分 (#1794 的 targets_expert_leaves、permute-fusion disable、post-finalize MoE geometry 校验) 最初在删除范围内, 最终 “restored in the trailing commit”, 因为专家路径是跨版本复用的能力而非 v1 特有逻辑。

实现拆解

1. 删除 v1 启动入口与示例脚本: train_multi_lora_async.py (-109, 含 main、_is_empty_batch_timeout)、examples/multi_lora/run_multi_lora.py (-204, 含 ScriptArgs、prepare/train/full_train/serve 四个 typer 子命令)、examples/multi_lora/service_smoke.py (-170, 含 ServiceClient、wait_for_step、register_when_allowed) 全部移除。原因是这套入口拼装的 --multi-lora-adapter/--multi-lora-disable-service-mode/--multi-lora-api-port 参数组合绑定了 v1 控制面; 影响是原先的

multi-LoRA 使用路径在 v2 落地前没有替代入口，必须按 PR 栈顺序合入。

2. 拆除控制面 (Ray actor + HTTP + registry + backend) :

`miles/ray/multi_lora/registry.py` (-252) 删除

`AdapterState/AdapterRecord/AdapterRegistry` 的生命周期状态机与

`mark_batch_trained` 记账逻辑; `miles/ray/multi_lora/backend.py` (-189) 删除注册期校

验 `resolve_adapter_config` (rank/alpha 上限、rollout_batch_size 与

`min_groups_per_dp_split` 的整除约束、data 路径存在性、reward 配置非空、save 目录

冲突) 与 `abort_adapter_requests`; `miles/ray/multi_lora/http_server.py` (-129) 删除

`/adapter_runs` REST 路由与 `RegisterAdapterRequest`;

`miles/ray/multi_lora/controller.py` (-123) 删除 `MultiLoRAController` 与 `AdaptersCache`。

这些能力在 v2 中由 `miles/tinker/core/scheduler.py`、`miles/tinker/core/service.py` (

#2844) 承接，槽位本身的簿记不再由控制面持有。

3. 拆除数据面 (per-adapter 数据源与完全异步 rollout) : `miles/rollout/multi_lora/data_source.py` (-137) 删除 `MultiLoRAAsyncDataSource.reconcile/create_source/get_samples` 的轮询取样本、`resolve_num_step` 的 `num_epoch`→`num_step` 推导;

`miles/rollout/multi_lora/async_rollout.py` (-584, 本次最大单文件删除) 删除

`GroupBuffer` (`drop_foreign/drop_stale`)、`remaining_groups` 的 batch 封顶、

`process_group` 的 `slot_version/registration_id` 打戳、`MultiLoRAWorkerMetrics` 的

per-adapter step 统计。删除原因是“按数据集驱动多租户并发训练”这一整套语义由 v2 重

新设计; 风险在于权重版本 staleness 与同名重注册的样本隔离这两条不变量需要在 v2 里

等价重建。

4. 保留并搬迁槽位原语: 新增 `miles/backends/megatron_utils/lora/slots.py` (+140), 承接原 `miles/backends/megatron_utils/multi_lora_utils.py` (-490) 中被判定为跨版本复用的部分: `create_multi_lora_instance`、`megatron_shard_name`、`adapter_shard_topology`、`zero_optimizer_state_for_adapter`、`slice_lora_to_rank`; 文件 docstring 明确“Kept deliberately free of any control-plane or data-plane logic”。

`miles/utils/multi_lora.py` 收缩为 +17/-112, 保留 `AdapterSpec`、

`EmptyBatchTimeoutError`、`define_new_adapter_metrics`、`slot_lora_name`、`make_rid`、

`parse_adapter`、`min_groups_per_dp_split`, 即路由命名、RID 拼接与 per-adapter 指标

命名等仍然共用的工具。被判定为 v1 特有的 `miles/utils/adapter_config.py` (-93,

`AdapterRunConfig/AdapterRun/parse_adapter_run_yaml`) 与

`miles/backends/megatron_utils/multi_lora_scheduler.py` (-88, per-slot LR/WD 调度

`build_slot_scheduler/step_slot_schedulers`) 则整体删除, 等于宣告 per-adapter YAML

配置契约与 per-adapter 独立学习率调度不在 v2 计划内。

5. 测试、文档与校验配套: 同步删除 6 个以上 fast 测试文件, 包括

`tests/fast/ray/multi_lora/test_controller_backend.py` (-392)、

`tests/fast/ray/rollout/test_multi_lora_batch_collection.py` (-313)、

`tests/fast/ray/multi_lora/test_controller_http.py` (-239)、

`tests/fast/test_train_multi_lora_async.py` (-189)、

`tests/fast/backends/megatron_utils/test_multi_lora_scheduler.py` (-114)、

`tests/fast/ray/rollout/test_multi_lora_train_data.py` (-106), 它们与源码同批移除以避免

import error; PR body 声明 multi-LoRA 文档同步更新, 且保留了 prefill 批处理断言 (

即 `async_rollout.py` 引用的 `recompute_samples_rollout_logprobs_via_prefill` 一线)，说明该断言并非 v1 专用。合入 main 时额外做了两处冲突对齐：保留 rollout session-metrics 的 import，移除已废弃的 `AdapterRef` 引用（与 #2598 的 metrics 改动相关）。本 PR 明确“adds no test cases”，验证仅为 `pre-commit run --all-files` 与组件级 `git diff --check`，限制与后续验证记录在 #2846。

关键文件：

- `miles/backends/megatron_utils/lora/slots.py`（模块 槽位管理；类别 source；类型 core-logic；符号 `create_multi_lora_instance`, `megatron_shard_name`, `adapter_shard_topology`, `zero_optimizer_state_for_adapter`）：本 PR 唯一新增文件，是被保留下来、供 v2 Tinker 命令执行器复用的槽位原语集合（模型插桩、shard 寻址、per-slot 优化器状态清理、rank 轴切分）。它定义了 v1 与 v2 的能力边界，是理解整个删除清单的钥匙。
- `miles/backends/megatron_utils/multi_lora_utils.py`（模块 槽位管理；类别 source；类型 deletion；符号 `create_multi_lora_instance`, `megatron_shard_name`, `adapter_shard_topology`, `all_megatron_checkpoints_exist`）：被删除的 490 行 v1 宿主文件：其中的槽位原语被搬进 `lora/slots.py` 保留，而 `save_multi_lora_checkpoints`（双格式落盘）与 `_register_adapter`（按 adapter 加载 checkpoint）属于 v1 checkpoint 流程，被 #2842/#2843 的新方案取代。对照这份文件能看清“哪些缩进了 `lora/` 目录、哪些彻底消失”。
- `miles/rollout/multi_lora/async_rollout.py`（模块 异步采样；类别 source；类型 deletion；符号 `iter_group_samples`, `first_sample`, `group_adapter_name`, `group_sample_count`）：本次最大单文件删除（-584）：完全异步 rollout 的 per-adapter 缓冲、staleness 过滤、注册隔离与 step 统计全部消失。这里的 `drop_stale/drop_foreign/remaining_groups` 是 v1 数据面的核心不变量，也是判断 v2 是否等价重建的主要依据。
- `miles/ray/multi_lora/registry.py`（模块 租户注册；类别 source；类型 deletion；符号 `AdapterState`, `AdapterRecord`, `AdapterRegistry`, `register`）：被删除的控制面核心（-252）：`AdapterState/AdapterRecord` 定义租户生命周期与 step 记账，`mark_batch_trained` 是唯一推进 `accumulated_groups/step` 的位置。v2 把这套状态机交给 Tinker 网关会话 / 调度器，理解它是理解 #2844 的前提。
- `miles/ray/multi_lora/backend.py`（模块 控制面；类别 source；类型 deletion；符号 `MultiLoRABackend`, `init`, `close`, `validate_adapter`）：被删除的注册期校验与引擎侧 abort（-189）。`resolve_adapter_config` 里集中了所有批量形状与前置条件校验（rank 上限、rollout_batch_size 整除、data 路径存在、reward 配置非空、save 目录冲突），删除后 v2 是否有等价的 fail-fast 校验是主要风险点。
- `miles/utils/multi_lora.py`（模块 公共工具；类别 source；类型 refactor；符号 `AdapterSpec`, `EmptyBatchTimeoutError`, `define_new_adapter_metrics`, `slot_lora_name`）：保留面在此收缩（+17/-112）：`AdapterSpec`、`EmptyBatchTimeoutError`、`slot_lora_name`、`make_rid`、`parse_adapter`、`min_groups_per_dp_split`、`define_new_adapter_metrics` 留下来成为 v2 的公共契约面，其余 v1 专用工具被清除。
- `miles/rollout/multi_lora/data_source.py`（模块 数据源；类别 source；类型 deletion；符号 `fetch_snapshot`, `sampleable`, `MultiLoRAAsyncDataSource`, `reconcile`）：被删除的数

据面调度 (-137) : 轮询取样本、按 adapter 建 RolloutDataSource、
num_epoch→num_step 推导。它决定了 v1 的“数据集驱动多租户”语义, 删除意味着 v2 不再以数据源轮询驱动 adapter 训练。

- miles/ray/multi_lora/controller.py (模块 控制面; 类别 source; 类型 deletion; 符号 get_multi_lora_controller, AdaptersCache, get_snapshot, get_all) : 被删除的 Ray 控制面 actor 与 AdaptersCache (-123) , 是 v1 中 rollout 侧读取 adapter 快照的唯一入口 (async_rollout 通过它打 slot_version/registration_id 戳) 。
- miles/ray/multi_lora/http_server.py (模块 控制面; 类别 source; 类型 deletion; 符号 RegisterAdapterRequest, MultiLoRAHTTPServer, actual_api_port, create_app) : 被删除的控制面 REST 层 (-129) : /adapter_runs 注册 / 查询 / 注销路由与异常映射。它是 v1 对外最显式的接口面, 也是外部集成最直接的 breaking 点。
- miles/backends/megatron_utils/multi_lora_scheduler.py (模块 调度器; 类别 source; 类型 deletion; 符号 _SlotParamGroups, build_slot_scheduler, install_slot_scheduler, drop_slot_scheduler) : 被删除的 per-slot LR/WD 调度 (-88) : 每个 adapter 一条 OptimizerParamScheduler, 按各自已训样本数定位。删除等于宣告 v2 不再保留 per-adapter 独立学习率行程, 是需要产品 / 训练侧确认的决策。
- miles/utils/adapter_config.py (模块 配置契约; 类别 source; 类型 data-contract; 符号 AdapterRunConfig, adapter_global_batch_size, AdapterRun, parse_adapter_run_yaml) : 被删除的配置契约 (-93) : AdapterRunConfig/AdapterRun/parse_adapter_run_yaml 是 adapter YAML 与 HTTP 注册的公共数据模型, 删除后 add 与 API 两侧同时失去类型基础, 说明 v2 不复用该 YAML 契约。
- train_multi_lora_async.py (模块 启动入口; 类别 source; 类型 entrypoint; 符号 _is_empty_batch_timeout, main) : 被删除的仓库根入口 (-109) : v1 multi-LoRA 训练的启动脚本, 含空批次超时判定。它是最直接的 breaking 点——从此 main 上不存在 v1 训练入口。
- tests/fast/ray/multi_lora/test_controller_backend.py (模块 测试用例; 类别 test; 类型 test-coverage; 符号 make_args, make_backend, make_config, register_and_promote) : 最大的被删测试 (-392) , 覆盖 RID 往返、注册后 PENDING→ACTIVE 提升、snapshot 词表、slot 版本跨复用单调性等不变量。作为删除清单中测试的代表, 它说明 v1 的回归保护被整体移除, 需由 v2 重建。

关键符号: create_multi_lora_instance, adapter_shard_topology, megatron_shard_name, zero_optimizer_state_for_adapter, slice_lora_to_rank, AdapterRegistry.mark_batch_trained, MultiLoRABackend.resolve_adapter_config, MultiLoRAAsyncDataSource.get_samples, GroupBuffer.drop_stale, GroupBuffer.drop_foreign, remaining_groups, build_slot_scheduler, min_groups_per_dp_split

关键源码片段

miles/backends/megatron_utils/lora/slots.py

本 PR 唯一新增文件, 是被保留下来、供 v2 Tinker 命令执行器复用的槽位原语集合 (模型插槽、shard 寻址、per-slot 优化器状态清理、rank 轴切分) 。它定义了 v1 与 v2 的能力边界

, 是理解整个删除清单的钥匙。

[miles/backends/megatron_utils/lora/slots.py](#) (本 PR 新增): v1 保留下来、由 v2 复用的槽位原语。

```
# 模块 docstring 就写明了设计边界: 只放模型插桩、shard 寻址与 per-slot 优化器状态清理,  
# 不掺任何控制面 (registry / HTTP) 或数据面 (data source / rollout) 逻辑,  
# 因为 v2 会把上层整个换成 Tinker 命令执行器 (#2840)。
```

```
# 整个 run 期间拓扑固定, 模块级缓存住一次 gloo all-gather 的结果。  
_shard_topology: tuple[bool, tuple[tuple[int, int, int], ...]] | None = None
```

```
def megatron_shard_name(tp_rank: int, pp_rank: int, ep_rank: int, ep_size: int) -> str:  
    """一个 (tp, pp, ep) 坐标对应的 adapter shard 文件名。  
  
    ep_size == 1 时省略 _ep 后缀, 专家并行开启之前写下的老 checkpoint 因此仍可加载。  
    """  
  
    name = f"adapter_megatron_tp{tp_rank}_pp{pp_rank}"  
    if ep_size > 1:  
        name += f"_ep{ep_rank}"  
    return name + ".pt"
```

```
def adapter_shard_topology() -> tuple[bool, tuple[tuple[int, int, int], ...]]:  
    """返回 (本 rank 是否负责写自己的 shard, 实际存在的 (tp, pp, ep) 坐标集合)。
```

同一次 all-gather 顺带解决“谁写盘”: 同一个坐标可能落在多个 DP rank 上,
只选 rank 号最小的进程写, 避免多个进程并发写同一个 shard 文件。

```
    """  
  
    global _shard_topology  
    if _shard_topology is not None:  
        return _shard_topology  
    parallel_state = get_parallel_state()  
    coords = (parallel_state.tp.rank, parallel_state.pp.rank, parallel_state.ep.rank)  
    if not dist.is_initialized():  
        # 未初始化分布式时退化为“自己写自己的坐标”, 方便单卡调试。  
        _shard_topology = (True, (coords,))  
        return _shard_topology  
  
    current_rank = dist.get_rank()  
    group = get_gloo_group()  
    gathered: list[object] = [None] * dist.get_world_size(group=group)  
    dist.all_gather_object(gathered, (coords, current_rank), group=group)  
    is_writer = current_rank == min(rank for entry_coords, rank in gathered if entry_coords ==  
    coords)  
    _shard_topology = (is_writer, tuple(sorted({entry_coords for entry_coords, _ in gathered})))  
    return _shard_topology
```

```
def zero_optimizer_state_for_adapter(optimizer, model, idx: int) -> None:
```

"""清理某个 slot 退役后的优化器状态，让下一个租户从零开始（bias correction 重新计时）。

v1 靠“复用同一个优化器 + 清零状态”实现 slot 复用；v2 改由每租户独立优化器承担（#2840 的 miles/backends/megatron_utils/lora/optimizer.py），所以这段逻辑虽然保留，但不再是 slot 复用的唯一保障。

"""

```
from megatron.bridge.peft.multi_lora_layers import MultiLoRALinear, _iter_multi_lora_modules
```

```
target_main_params = set()
```

```
for module in _iter_multi_lora_modules(model):
```

```
    if not isinstance(module, MultiLoRALinear):
```

```
        continue
```

```
    adapter = module.adapters[idx]
```

```
    for param in adapter.parameters():
```

```
        # distributed optimizer 会把权重挂到 main_param 上，要按 main_param 的 id 匹配
```

```
        main = getattr(param, "main_param", None)
```

```
        target_main_params.add(id(main if main is not None else param))
```

```
chained = getattr(optimizer, "chained_optimizers", [optimizer])
```

```
for chained_optimizer in chained:
```

```
    inner = getattr(chained_optimizer, "optimizer", chained_optimizer)
```

```
    if inner is None:
```

```
        continue
```

```
    # TE/apex FusedAdam 的 Adam step 挂在 param group 上而不是单个参数上，
```

```
    # 所以要按 group 上打的 miles_multi_lora_slot 标记重置。
```

```
    for group in inner.param_groups:
```

```
        if group.get("miles_multi_lora_slot") == idx and "step" in group:
```

```
            if isinstance(group["step"], torch.Tensor):
```

```
                group["step"].zero_()
```

```
            else:
```

```
                group["step"] = 0
```

```
    for param, state in inner.state.items():
```

```
        if id(param) not in target_main_params:
```

```
            continue
```

```
        if "exp_avg" in state:
```

```
            state["exp_avg"].zero_()
```

```
        if "exp_avg_sq" in state:
```

```
            state["exp_avg_sq"].zero_()
```

```
        # 下一个租户的 bias correction 从这里重新开始
```

```
        if "step" in state:
```

```
            if isinstance(state["step"], torch.Tensor):
```

```
                state["step"].zero_()
```

```
            else:
```

```
                state["step"] = 0
```

miles/rollout/multi_lora/async_rollout.py

本次最大单文件删除 (-584)：完全异步 rollout 的 per-adapter 缓冲、staleness 过滤、注册隔离与 step 统计全部消失。这里的 drop_stale/drop_foreign/remaining_groups 是 v1 数据面的核心不变量，也是判断 v2 是否等价重建的主要依据。

[miles/rollout/multi_lora/async_rollout.py](#) (本 PR 整文件删除)：v1 数据面的两条关键不变量，v2 需要确认是否等价覆盖。

```
# 后台生产者线程按 adapter 往有界 FIFO 里塞 group，训练侧轮询取批；
# 训练完全停摆时靠这个上限兜底内存，与 fully_async 的 queue.Queue(maxsize=1000) 同一约定。
MAX_BUFFERED_GROUPS = 1000
EMPTY_BATCH_TIMEOUT_S = 30.0
```

```
class GroupBuffer:
    """单个 adapter 的已完成 prompt group FIFO；有界，满了丢最老的。"""

    def __init__(self) -> None:
        self._groups: deque[Group] = deque(maxlen=MAX_BUFFERED_GROUPS)

    def get(self, n_groups: int) -> list[Group]:
        """取走最老的 n 个 group (queue.Queue 风格的接口)。"""
        return [self._groups.popleft() for _ in range(n_groups)]

    def drop_foreign(self, registration_id: str) -> int:
        """丢弃被同名 adapter 的另一次注册打过标记的 group。

        旧租户在途的生成可能在 buffer 被重置后才落地，如果不清掉，新租户会拿到别人的样本；
        提交采样时还看不到 adapter 视图、因此没打标记的 group 则保留。
        """
        if not self._groups:
            return 0
        kept: deque[Group] = deque(maxlen=MAX_BUFFERED_GROUPS)
        dropped = 0
        for group in self._groups:
            stamped = first_sample(group).metadata.get("registration_id")
            if stamped is not None and stamped != registration_id:
                dropped += 1
            else:
                kept.append(group)
        self._groups = kept
        return dropped

    def drop_stale(self, current_version: int, max_staleness: int | None) -> list[int]:
        """丢弃生成时权重版本过旧的 group，返回各自 staleness 用于打点。

        v1 用 --max-weight-staleness 控制 on-policy 程度：slot_version 在提交采样时打戳，
        被丢的样本先 reset_for_retry() 再计数。
        """
        if max_staleness is None or not self._groups:
```

```

    return []
    kept: deque[Group] = deque(maxlen=MAX_BUFFERED_GROUPS)
    dropped: list[int] = []
    for group in self._groups:
        stamped = first_sample(group).metadata.get("slot_version")
        staleness = current_version - stamped if stamped is not None else 0
        if stamped is not None and staleness > max_staleness:
            for sample in iter_group_samples(group):
                sample.reset_for_retry()
            dropped.append(staleness)
        else:
            kept.append(group)
    self._groups = kept
    return dropped

```

```
def remaining_groups(adapter) -> int:
```

```
    """该 adapter 还差多少 group 才能凑满一个 batch。
```

```
    取批时用它封顶，保证不会为了凑某个 adapter 的 batch 而多拿别人的 group，
    从而让每个 adapter 的 step 与 rollout_batch_size 严格对齐。
```

```
    """
```

```

    remaining = adapter.config.rollout_batch_size - adapter.accumulated_groups
    assert remaining > 0, (
        f"adapter '{adapter.name}' accumulated_groups={adapter.accumulated_groups} >= "
        f"rollout_batch_size={adapter.config.rollout_batch_size}; batch accounting drifted"
    )
    return remaining

```

miles/ray/multi_lora/registry.py

被删除的控制面核心 (-252)：AdapterState/AdapterRecord 定义租户生命周期与 step 记账，mark_batch_trained 是唯一推进 accumulated_groups/step 的位置。v2 把这套状态机交给 Tinker 网关会话 / 调度器，理解它是理解 #2844 的前提。

miles/ray/multi_lora/registry.py (本 PR 整文件删除)：v1 租户生命周期与训练记账的核心状态机，v2 由 Tinker 网关重建。

```

# v1 的租户生命周期：PENDING -> ACTIVE -> RETIRING -> CLEANUP -> COMPLETED。
# slot 会被复用，但 slot_versions 永不重置，保证 (slot, version) 组合不重复出现；
# v2 把这套生命周期挪到 Tinker 网关 (#2844 的 miles/tinker/core/scheduler.py 与 service.py) ，
# 槽位本身仍由 miles/backends/megatron_utils/lora/slots.py 维护。

```

```

class AdapterState(str, Enum):
    PENDING = "PENDING" # 已注册但权重还没推上去
    ACTIVE = "ACTIVE" # 权重就位，可被采样
    RETIRING = "RETIRING" # 已请求下线，但继续服务在途请求
    CLEANUP = "CLEANUP" # 等待 abort 完成，之后才能释放 slot
    COMPLETED = "COMPLETED" # slot 已释放，记录只留作查询

```

```

@dataclass
class AdapterRecord:
    name: str
    slot: int
    config: Any
    step: int = 0
    # num_step 是相对 start_step 计的，checkpoint 续训时把起点 step 存下来
    start_step: int = 0
    # 已提交训练的 prompt group 数；只有 mark_batch_trained（train 成功之后）才推进，
    # 因此 train 调用失败重试不会污染计数器
    accumulated_groups: int = 0
    state: AdapterState = AdapterState.PENDING
    # 每次注册都是新租户：同名再注册时，rollout 侧旧租户留下的状态必须按这个 id 丢弃
    registration_id: str = field(default_factory=lambda: uuid.uuid4().hex)

```

```

def mark_batch_trained(self, rollout_id: int) -> list[str]:
    """结算一个已训练 batch 的 group 数并触发 step；返回本批完成 step 的 adapter。

```

这里是唯一推进 accumulated_groups / step 的地方，所以 train 失败或重试时 registry 的计数保持不动，不会出现“训了但没记账”或“记了但没训”的漂移。

```

    """
    record_entry = self.batch_records.pop(rollout_id, None)
    if record_entry is None:
        return []
    stepped = []
    reached_num_step = []
    for name, n_groups in record_entry["groups"].items():
        record = self.records.get(name)
        if record is None or record.state not in (
            AdapterState.ACTIVE,
            AdapterState.RETIRING,
            AdapterState.CLEANUP,
        ):
            continue
        record.accumulated_groups += n_groups
        if name in record_entry["step_names"]:
            # 收集侧按 adapter 的剩余 group 数封顶，正常应正好等于 rollout_batch_size
            target = record.config.rollout_batch_size
            if record.accumulated_groups != target:
                logger.warning(
                    f"Adapter '{name}' stepped with accumulated_groups={record.accumulated_groups} "
                    f"!= rollout_batch_size={target}; adapter batch accounting drifted"
                )
            record.step += 1
            record.accumulated_groups = 0
            stepped.append(name)
    # num_step 到点即自动注销；用 step - start_step 支持从 checkpoint 续训

```

```

if (
    getattr(record.config, "num_step", None) is not None
    and record.state is AdapterState.ACTIVE
    and (record.step - record.start_step) >= record.config.num_step
):
    reached_num_step.append(name)
for name in reached_num_step:
    self.deregister(name)
return stepped

```

评论区精华

本 PR 的评审记录非常薄：[claude\[bot\]](#) 只留下仓库级 Code Review 模板评论，[yushengsu-thu](#) 给出 APPROVED 且正文为空，没有任何行内 review comment，因此实质讨论发生在 PR 之外的 issue 评论里。最有价值的一处是作者对保留清单的自我修正：MoE expert adapter 的 build-time 部分（#1794 的 targets_expert_leaves、permute-fusion disable、post-finalize MoE geometry 校验）最初被删，最终在 trailing commit 中恢复保留，理由是該路径属于专家适配器的通用几何约束而非 v1 控制面逻辑；其余专家路径（EP-aware shard 拓扑、expert-axis-safe 的 rank 切片）本来就在保留清单。第二处是 PR body 明示的覆盖缺口：“This update adds no test cases. Current validation and limitations are recorded on #2846.”，即删除 5000 行实现与配套测试后，回归保护完全依赖后续 PR 重建。注意提交列表只有一个 squash 后的 commit (head 958e1ca)，说明“trailing commit”以 amend/squash 形式进入 head，审查时应以 head_sha 为准而非提交历史。

- MoE 专家适配器（#1794）的 build-time 逻辑是删还是留 (design): 最终采用保留方案，在 trailing commit 中恢复；但提交列表只有 1 个 squash commit，需以 head_sha 958e1ca 的文件内容确认恢复完整。
- 删除范围与 v2 落地时序的兼容性风险 (question): 以 PR 栈顺序合入作为隐式约定；该风险未被显式评审，属于需要持续跟踪项。
- 本 PR 不新增测试，验证依赖后续 PR 重建 (testing): 已知限制，验证与限制记录在 #2846；租户生命周期、slot 版本单调性、batch 记账漂移告警等不变量暂时失去回归保护。
- 评审过程本身缺少实质交锋 (other): 该 PR 的实质决策讨论发生在 issue 评论与 #2846，而非 PR 评论线程，后续同类清理型 PR 建议在描述里补充保留 / 删除清单对照表。

风险与影响

- 风险：1) 兼容性 breaking，且本 PR 单独合入时功能缺位：train_multi_lora_async.py、examples/multi_lora/run_multi_lora.py、/adapter_runs HTTP API、miles/utils/adapter_config.py 的 AdapterRunConfig 全部消失，任何依赖 v1 入口或 --multi-lora-adapter YAML 的外部脚本 / 流水线会直接失败；v2 替代实现在 #2840-#2849，因此必须严格按栈序合入，main 在中间窗口内没有可用的 multi-LoRA 路径。2) 删除的验证覆盖形成的回归盲区：test_controller_backend.py、test_multi_lora_batch_collection.py 覆盖的租户生命周期、slot_versions 单调性、batch 记账漂移告警 (accumulated_groups != rollout_batch_size) 等不变量随测试一并删除，

v2 若不重建等价断言，这类漂移会静默发生。3) 数据面不变量的迁移风险：

GroupBuffer.drop_foreign 用 registration_id 隔离同名重注册的残留样本、drop_stale 用 slot_version 与 --max-weight-staleness 控制 on-policy 程度，这两条保证在 v2 中是否等价存在，是后续 review 的关键检查项。4) per-slot 调度与配置契约被放弃：

multi_lora_scheduler.py 的 per-adapter LR/WD 调度与 adapter_config.py 的 YAML 契约删除后不可恢复，若 v2 仍需 per-adapter 学习率行程，需要重新设计而非简单搬运。5) import 残留：miles/utils/multi_lora.py 保留面收窄、AdapterRef 被移除，任何未一并清理的调用方（包括与 #2598 相关的 rollout metrics / sample codec 路径）会 importerror；本 PR “至少 22 个测试文件与源码模块存在名称交集”，交集之外的引用需要靠 CI 兜底。

6) MoE 专家路径的恢复完整性：issue 评论说明 build-time 三处逻辑是“trailingcommit”补回来的，需确认恢复的不是残缺版本（例如几何校验只恢复了调用而没恢复常量或分支），否则 MoE 适配器会静默失去构建期保护。

- 影响：对用户：仍在用 v1 路径的用户（run_multi_lora.py、service_smoke.py、adapter YAML、/adapter_runs 接口）必须等待并迁移到 Tinker 网关形态；单租户 LoRA 与 miles/backends/megatron_utils/lora_utils.py 路径不受影响（#3245/#3083 针对该路径的修复仍然成立）。对系统：仓库净减约 5000 行、72 个文件，multi-LoRA 的复杂度从“驱动 + 控制面 + 数据面 + per-slot 调度”收敛为“槽位原语 + Tinker 网关”两层，长期维护面明显收窄，但短期内 multi-LoRA 能力在 main 上处于真空。对团队：作为 7 PR 栈的第 0 个，它决定了后续评审者对“保留 vs 删除”边界的共同理解，miles/backends/megatron_utils/lora/、miles/utils/multi_lora.py 的保留清单此后就是 v2 的契约面。
- 风险标记：功能删除型 breaking 变更，v1 入口与 HTTP API 缺失替代，删除测试造成回归盲区，数据面不变量（staleness/ 注册隔离）待 v2 等价重建，per-adapter 调度与 YAML 契约不可恢复，AdapterRef 等 import 残留风险，MoE 专家路径恢复完整性待核

关联脉络

- PR #2840 [multi-lora v2 1/7] LoRA model extension and slot actor: PR body 明确 'The Tinker command executor builds on these primitives in #2840'，它直接复用本 PR 保留在 miles/backends/megatron_utils/lora/ 下的槽位原语，并新增每租户独立优化器，取代被删的 zero_optimizer_state_for_adapter 清零方案。
- PR #2842 [multi-lora v2 3/7] Per-slot checkpoints with optimizer state: v2 的 per-slot checkpoint 方案取代本 PR 删除的 v1 checkpoint 流程（multi_lora_utils.py 的 save_multi_lora_checkpoints、_register_adapter 与 find_latest_checkpoint）。
- PR #2843 [multi-lora v2 4/7] Publish immutable adapter snapshots to disk: 本 PR 删掉了 v1 的 adapter checkpoint 落盘与 HF 导出路径，v2 改由不可变快照发布机制承担，属于同一功能线的直接替代。
- PR #2844 [multi-lora v2 5/7] Tinker gateway protocol face: 本 PR 删除的 AdapterRegistry/MultiLoRABackend/MultiLoRAHTTPServer 控制面（租户生命周期、注册校验、/adapter_runs 接口）在 v2 中由 Tinker 网关的 core/service.py、core/scheduler.py 承接，是删除清单的主要去向。
- PR #2849 [multi-lora v2 7/7] Tinker gateway example and tests: 替代本 PR 删除的 examples/multi_lora/run_multi_lora.py 与 service_smoke.py，v2 的示例启动器与端到端

冒烟测试在该 PR 中重建。

- PR #1794 MoE expert adapter support (标题未在本 PR 材料中给出) : issue 评论明确引用其 build-time 三处逻辑 (targets_expert_leaves、permute-fusion disable、post-finalize MoE geometry 校验) , 这些内容最初在删除范围内, 最终恢复保留。