

# PR #2840 完整报告

radixark/miles

[multi-lora v2 1/7] LoRA model extension and slot actor

合并时间: 2026-09-16 16:24

原文链接: <http://prhub.com.cn/radixark/miles/pull/2840>

## 执行摘要

- 一句话: 新增多 LoRA 槽位执行器与每租户独立优化器
- 推荐动作: 建议精读。这是整套 multi-LoRA v2 的地基 (1/7), 后续 6 个 PR 的损失选择、槽位 checkpoint、快照发布和网关 runtime 都建立在它的命令面与优化器所有权模型上, 改动虽然只有 +453/-352, 但触及所有 Megatron 训练路径。推荐阅读顺序: 先看 megatron\_utils/model.py 的抽函数是否严格等价 (尤其被丢掉的 no\_sync\_func 断言), 再看 lora/model.py 的 4 个槽位操作函数, 最后细读 lora/optimizer.py 的 SlotOptimizer 与 step\_slot\_optimizers。需要重点确认三个设计决策: (1) 是否接受“pre-collective 失败 = cell 失败”的粗粒度恢复; (2) nonfinite 跳过时直接丢弃梯度是否符合网关的重放语义; (3) 每租户独立优化器的显存增长是否在容量预算内。同时留意 review 点名但未补齐的双槽位 / 两 rank 回归测试, 以及 body 中“无新增测试文件”与实际 diff 的口径差异。

## 功能与动机

PR body 给出的目标是 Expose command-level training on MultiLoRA TrainRayActor in megatron\_utils/lora/actor.py: 让 Tinker 风格的网关可以像发命令一样驱动训练 (累积梯度的 forward\_backward、按槽位下发 Adam 超参的 optim\_step、只做打分的 forward\_only\_logprobs, 以及槽位级 load/unload), 而不是沿用整批 rollout 驱动的训练流程。它是堆叠在 #2839 之上的 1/7 连发, 后续损失选择 (#2841)、单槽位 checkpoint (#2842)、快照发布 (#2843)、协议层 (#2844) 与 runtime (#2845) 都依赖这里的命令面与优化器所有权模型。动机的第二层来自 review: nanjiangwill 指出池化优化器上的 reload\_model\_params() 会顺带把其他活跃槽位的 FP32 master 取整, 破坏租户隔离, 这直接催生了“每槽位一个独立 SlotOptimizer、随槽位销毁”的设计, 也解释了为什么本 PR 连带删除 zero\_optimizer\_state\_for\_adapter 及其测试——状态不再跨租户复用, 手工复位逻辑失去意义。

## 实现拆解

1. 抽出共享执行原语 (miles/backends/megatron\_utils/model.py): 把 train\_one\_step 里的 pipeline FWD/BWD 主体抽成新的 run\_forward\_backward\_pass(args, dumper\_phase\_util, data\_iterator, model, num\_microbatches, num\_rollouts), 只返回 losses\_reduced、不碰优化器; 把 train() 内联的 DDP hook 配置抽成 setup\_train\_iteration\_config(args, model, optimizer, disable\_optimizer), 统一设置 grad\_scale\_func、no\_sync\_func、grad\_sync\_func、param\_sync\_func 与

finalize\_model\_grads\_func。train\_one\_step 与 train 改为调用这两个符号，行为等价，使多 LoRA 命令路径能复用同一套并行与梯度归约语义；注意抽取时去掉了原 overlap\_grad\_reduce 分支里的 assert config.no\_sync\_func is None 前置校验。

2. 优化器所有权改为每槽位一个实例 (lora/optimizer.py)：删除 build\_multi\_lora\_optimizer、step\_adapter\_slots、zero\_adapter\_slot\_grads 这套“池化 + 按 slot 索引 chained children”的实现，新增 validate\_multi\_lora\_optimizer\_args，把 use\_distributed\_optimizer、fp16、非 Adam、optimizer\_cpu\_offload、stream\_optimizer\_state\_to\_disk、rematerialize\_param\_from\_master\_weight 等不支持项前移为启动期断言；setup\_model\_and\_optimizer 的多 LoRA 分支现在只做校验并直接 return model, None, None，即没有池化优化器与 scheduler。新增 SlotOptimizer 类：构造时由 \_build\_slot\_base\_optimizers 在 \_only\_slot\_trainable 上下文里为单槽位建 base optimizer（临时把 config.bf16 置 False 以拿到未包装的 torch optimizer），再包一层 LayerWiseDistributedOptimizer，并把每个 child 的 grad\_stats\_parallel\_group 置 None；对外提供 apply\_adam\_params、prepare\_grads、clip\_and\_step、allgather\_params、reload\_masters、zero\_grads。新增 step\_slot\_optimizers 统一编排“按固定顺序 apply → prepare → step → zero → allgather”，注释写明执行期异常一律让 cell 失败。
3. 新增槽位操作层 (lora/model.py, 新增 73 行)：run\_forward\_backward 通过 get\_data\_iterator 取数据、setup\_train\_iteration\_config(disable\_optimizer=True) 配置本轮、reset\_grad\_metadata\_keep\_grads 只重置 DDP 记账而保留梯度累积窗口，并断言“一个 work unit 就是一次 FWD/BWD”，仅在 PP 末级聚合 loss；optim\_step 只把 adam\_params\_by\_slot 中出现的槽位交给 step\_slot\_optimizers (batch size 1，不做额外归一化，归一化交给客户端 loss 权重)；load\_slot 调 bridge 的 init\_adapter\_slot 后创建 SlotOptimizer 并清零梯度；unload\_slot 调 clear\_adapter\_slot，由调用方丢弃实例、优化器状态随之消亡。
4. Actor 与 worker 装配 (lora/actor.py、actor.py、ray/specs/train.py、ray/train/group.py)：新增 MultiLoRATrainRayActor，在 \_init\_training\_state 里建 slot\_optimizers 注册表，并实现 5 个带 @with\_logs 与 \_heartbeat.bump() 的命令方法；forward\_only\_logprobs 只在 PP 末级返回，并把结果 ray.put 成 CPU tensor 盒子。基类 MegatronTrainRayActor 把权重更新器初始化抽成 \_init\_training\_state() 钩子，多 LoRA 子类覆写它从而不初始化普通 WeightUpdater。\_compute\_spec\_trainer 仅在 train\_backend == megatron、role == actor 且 is\_multi\_lora\_enabled(args) 时替换 worker\_class。miles/ray/train/group.py 新增 \_execute\_slots 与 6 个转发方法，要求 Tinker 训练组恰好一个存活 cell。
5. 测试、清理与配套：删除 tests/fast/backends/megatron\_utils/test\_multi\_lora\_slot\_cleanup.py（被测的 zero\_optimizer\_state\_for\_adapter 已随设计变更移除），新增 tests/fast/backends/megatron\_utils/test\_step\_adapter\_slots.py，用 \_FakeSlotOptimizer 覆盖“未知 step 失败在报告成功前逃逸且不 all-gather”“nonfinite grad norm 跳过步进”“prepare 阶段失败逃逸”；扩展 tests/fast/ray/specs/test\_train.py 的 worker class 选择用例并保留 critic 的后端选择；miles/utils/multi\_lora.py 为 worker 选择提供 is\_multi\_lora\_enabled 判定。body 声明无新增其它测试文件、pre-commit 与 git diff --check 通过、fresh CPU/GPU CI pending，验证与限制记录在 #2846 与 #3198。

关键文件：

- miles/backends/megatron\_utils/lora/optimizer.py (模块 优化器; 类别 source; 类型 core-logic; 符号 validate\_multi\_lora\_optimizer\_args, SlotOptimizer, apply\_adam\_params, prepare\_grads) : 多 LoRA 优化器所有权的核心改造: 删除池化的 build\_multi\_lora\_optimizer/step\_adapter\_slots, 改为每槽位一个 SlotOptimizer 与全局排序的 step\_slot\_optimizers, 并新增启动期参数校验。
- miles/backends/megatron\_utils/lora/model.py (模块 模型执行; 类别 source; 类型 core-logic; 符号 run\_forward\_backward, optim\_step, load\_slot, unload\_slot) : 新增的槽位操作层, 是 actor 命令与底层 optimizer/ 模型执行之间唯一的粘合点, 定义了梯度累积、按槽位步进与槽位装载语义。
- miles/backends/megatron\_utils/model.py (模块 训练主循环; 类别 source; 类型 refactor; 符号 run\_forward\_backward\_pass, setup\_train\_iteration\_config, train\_one\_step) : 抽出 run\_forward\_backward\_pass 与 setup\_train\_iteration\_config, 多 LoRA 与常规训练共用; 所有 Megatron 训练路径都会经过这里, 回归面最大。
- miles/backends/megatron\_utils/lora/actor.py (模块 训练执行器; 类别 source; 类型 entrypoint; 符号 MultiLoRATrainRayActor, \_init\_training\_state, forward\_backward, optim\_step) : 新增 MultiLoRATrainRayActor, 是命令级训练接口 (forward\_backward/optim\_step/forward\_only\_logprobs/load\_slot/unload\_slot) 的唯一入口, 也是后续 Tinker 网关调用的落点。
- miles/backends/megatron\_utils/actor.py (模块 训练基类; 类别 source; 类型 refactor; 符号 \_init\_training\_state) : 把权重更新器初始化抽成 \_init\_training\_state() 钩子, 使多 LoRA 子类能够替换训练状态而不复用普通 WeightUpdater, 是继承式扩展的关键接缝。
- miles/ray/train/group.py (模块 训练组; 类别 source; 类型 core-logic; 符号 \_execute\_slots, forward\_backward, optim\_step, forward\_only\_logprobs) : 新增 \_execute\_slots 与 6 个槽位命令转发, 是网关侧调用训练 cell 的通道, 同时用断言把多 LoRA 训练组约束为单 cell。
- miles/ray/specs/train.py (模块 进程规格; 类别 source; 类型 dependency-wiring) : worker class 选择逻辑: 仅 megatron 多 LoRA actor 走 MultiLoRATrainRayActor, 其余后端与 critic 保持原选择。
- tests/fast/backends/megatron\_utils/test\_step\_adapter\_slots.py (模块 优化器测试; 类别 test; 类型 test-coverage; 符号 \_FakeSlotOptimizer, test\_an\_unknown\_step\_failure\_escapes\_before\_success\_is\_reported, test\_a\_nonfinite\_grad\_norm\_skips\_the\_step, test\_an\_unknown\_preparation\_failure\_escapes) : 新增测试覆盖失败传播与 nonfinite 跳过语义, 但基于 fake 优化器, 不涉及真实 LayerWise 与 collective 行为。
- miles/backends/megatron\_utils/lora/slots.py (模块 槽位管理; 类别 source; 类型 dead-code-removal; 符号 zero\_optimizer\_state\_for\_adapter) : 删除 zero\_optimizer\_state\_for\_adapter, 因为槽位状态不再跨租户复用, 手工复位 Adam 动量与 step 时钟的逻辑失去意义。
- tests/fast/backends/megatron\_utils/test\_multi\_lora\_slot\_cleanup.py (模块 槽位测试; 类别 test; 类型 test-removal; 符号 test\_group\_level\_fused\_adam\_clock\_resets\_only\_for\_the\_retired\_slot, test\_tensor\_valued\_group\_clock\_resets\_in\_place,

test\_per\_param\_adamw\_fallback\_clock\_resets)：被删除的测试文件，标志着槽位优化器状态复位逻辑随设计变更退场（其被测函数已不存在）。

- tests/fast/ray/specs/test\_train.py（模块 规格测试；类别 test；类型 test-coverage；符号 test\_the\_backend\_selects\_the\_worker\_class）：更新 worker class 选择用例，锁定多 LoRA actor 分支并保留 critic 的后端选择，防止 worker 装配回归。
- miles/utils/multi\_lora.py（模块 开关工具；类别 source；类型 core-logic）：为 worker class 选择提供 is\_multi\_lora\_enabled 判定的小幅改动，被 miles/ray/specs/train.py 直接导入。

关键符号：MultiLoRATrainRayActor.forward\_backward,  
MultiLoRATrainRayActor.optim\_step, MultiLoRATrainRayActor.forward\_only\_logprobs,  
MultiLoRATrainRayActor.load\_slot, MultiLoRATrainRayActor.unload\_slot,  
SlotOptimizer.init, SlotOptimizer.clip\_and\_step, SlotOptimizer.zero\_grads,  
step\_slot\_optimizers, validate\_multi\_lora\_optimizer\_args, run\_forward\_backward,  
run\_forward\_backward\_pass, setup\_train\_iteration\_config,  
MegatronTrainRayActor.\_init\_training\_state, TrainRayGroup.\_execute\_slots

## 关键源码片段

### miles/backends/megatron\_utils/lora/model.py

新增的槽位操作层，是 actor 命令与底层 optimizer/ 模型执行之间唯一的粘合点，定义了梯度累积、按槽位步进与槽位装载语义。

```
"""Multi-LoRA slot operations extending the shared Megatron model execution."""
```

```
def run_forward_backward(args, batch_id: int, model, rollout_data) -> dict:
    """一个 work unit 就是一次 forward/backward；梯度在此累积，不做 optimizer step。"""
    data_iterator, num_microbatches = get_data_iterator(args, model, rollout_data)
    assert len(num_microbatches) == 1, "a work unit is a single forward/backward pass"

    for iterator in data_iterator:
        iterator.reset()
    for model_chunk in model:
        model_chunk.train()
    # disable_optimizer: bf16 下 loss scaling 是恒等的，这次 pass 不需要 optimizer
    setup_train_iteration_config(args, model, None, disable_optimizer=True)
    # 只重置 DDP 的 per-iteration 记账，保留跨 batch 的梯度累积窗口
    reset_grad_metadata_keep_grads(model)

    dumper_phase_util = DumperMegatronUtil(args, model, DumperPhase.FWD_BWD, rollout_id=batch_id)
    losses_reduced = run_forward_backward_pass(
        args, dumper_phase_util, data_iterator, model, num_microbatches[0], num_rollouts=None
    )
    dumper_phase_util.finalize(model)

    if get_parallel_state().is_pp_last_stage:
```

```

        return aggregate_train_losses(losses_reduced, None)
    return {}

```

```

def optim_step(args, slot_optimizers: dict, adam_params_by_slot: dict) -> dict[int, dict]:
    # batch size 为 1: 梯度按累积值步进, 归一化交给客户端的 loss 权重
    stepped = {slot: slot_optimizers[slot] for slot in adam_params_by_slot}
    return step_slot_optimizers(stepped, adam_params_by_slot, clip_grad=args.clip_grad)

```

```

def load_slot(args, model, slot: int, rank: int, alpha: float) -> "SlotOptimizer":
    from megatron.bridge.peft.multi_lora_layers import init_adapter_slot

    init_adapter_slot(model, slot, rank=rank, alpha=alpha) # 在模型上挂载该槽位的 adapter
    slot_optimizer = SlotOptimizer(args, model, slot) # 优化器状态随槽位一起新建
    slot_optimizer.zero_grads()
    return slot_optimizer

```

```

def unload_slot(model, slot_optimizer) -> None:
    """调用方随后会丢弃该实例, 其状态随之消亡。"""
    from megatron.bridge.peft.multi_lora_layers import clear_adapter_slot

    clear_adapter_slot(model, slot_optimizer.slot)
    slot_optimizer.zero_grads()

```

## miles/backends/megatron\_utils/lora/actor.py

新增 MultiLoRATrainRayActor, 是命令级训练接口 (forward\_backward/optim\_step/forward\_only\_logprobs/load\_slot/unload\_slot) 的唯一入口, 也是后续 Tinker 网关调用的落点。

```

class MultiLoRATrainRayActor(MegatronTrainRayActor):
    """命令级训练 actor: 每个命令一次远程调用, 梯度跨调用累积。"""

    def _init_training_state(self) -> None:
        # 覆写基类钩子: 多 LoRA 没有池化优化器, 只有按槽位注册的 SlotOptimizer
        self.slot_optimizers: dict[int, SlotOptimizer] = {}

    @with_logs
    def forward_backward(self, batch_id: int, rollout_data_ref: Box) -> dict:
        self._heartbeat.bump()
        with ExitStack() as stack:
            rollout_data, store_get_result = get_rollout_data(self.args, rollout_data_ref)
            stack.enter_context(store_get_result)
            return lora_model.run_forward_backward(self.args, batch_id, self.model, rollout_data)

    @with_logs
    def optim_step(self, adam_params_by_slot: dict[int, dict]) -> dict[int, float]:
        # 只有请求里出现的槽位会被步进, 其余槽位的梯度继续保留
        return lora_model.optim_step(self.args, self.slot_optimizers, adam_params_by_slot)

```

```

@with_logs
def forward_only_logprobs(self, batch_id: int, rollout_data_ref: Box) -> Box | None:
    self._heartbeat.bump()
    with ExitStack() as stack:
        rollout_data, store_get_result = get_rollout_data(self.args, rollout_data_ref)
        stack.enter_context(store_get_result)
        data_iterator, num_microbatches = get_data_iterator(self.args, self.model, rollout_data)
        outputs = self.compute_log_prob(data_iterator, num_microbatches, rollout_id=batch_id)
    if not get_parallel_state().is_pp_last_stage: # 非 PP 末级不返回结果
        return None
    # 只把 CPU tensor 放进 Ray 对象存储, 避免序列化 GPU 张量
    return Box(ray.put({key: [t.cpu() for t in tensors] for key, tensors in outputs.items()}))

@with_logs
def load_slot(self, slot: int, rank: int, alpha: float) -> None:
    self.slot_optimizers[slot] = lora_model.load_slot(self.args, self.model, slot, rank, alpha)

@with_logs
def unload_slot(self, slot: int) -> None:
    # pop 出来后交给 unload_slot, 实例被丢弃时它的优化器状态一起消亡
    lora_model.unload_slot(self.model, self.slot_optimizers.pop(slot))

```

## 评论区精华

本 PR 只有两条实质性行内意见，均来自 nanjiangwill、状态均为 COMMENTED，作者未在评论区回复，处理方式只能从最终代码反推：

关于（早期版本的）`miles/backends/megatron_utils/lora/executor.py`：“optimizer is the outer LayerWiseDistributedOptimizer, so `reload_model_params()` walks every child optimizer. Each BF16 child copies its model parameter back into its FP32 master. Loading or unloading slot A therefore rounds the FP32 masters for every live slot, changing their later Adam updates. Reload only for child in `_slot_children(optimizer, slot)` here and in `unload_slot`, and add a two-slot regression test where slot B’s FP32 master differs from its BF16 model value.”

关于 `lora/optimizer.py` 的 `step_adapter_slots`：“`_step_one_slot()` can fail on one rank during `prepare_grads()` or batch-size scaling before `get_grad_norm_fp32()` enters the world all-reduce. That rank catches the exception here and moves to the Gloo outcome gather while healthy ranks block in the norm all-reduce, turning per-slot recovery into a collective timeout. Add rank consensus after local preparation and before the norm collective, or treat any pre-collective failure as cell-fatal.”

结论与决策：第一条被采纳为架构级改动——池化优化器整体退场，SlotOptimizer 每槽位持有自己的 LayerWiseDistributedOptimizer，`reload_masters` 只作用于 `self._inner`，租户隔离从“约定”变成结构保证，并连带删除 `zero_optimizer_state_for_adapter`。第二条采纳了作者给出的第二个选项“pre-collective 失败即 cell-fatal”：`step_slot_optimizers` 不再捕获异常、注释写明 `execution failures invalidate the cell`，新测试断言异常向上逃逸且失败槽位不触发 `allgather_params`，代价是单个槽位故障会拖垮整个 cell。两条意见各自点名要求的回归测试

(双槽位 FP32 master 差异、两 rank 在 `get_grad_norm_fp32` 之前失败) 在本次改动中都没有落地，现有测试是单进程的 fake 版本。此外 yushengsu-thu 给了 APPROVED (checked)，claude[bot] 只是仓库的 review 配置提示，nanjiangwill 的两条意见没有收到作者的文字回复。

- 槽位 load/unload 会改写其他租户的 FP32 master (correctness): 采纳并升级为架构级改动：池化优化器被删除，SlotOptimizer 每个槽位持有自己的 LayerWiseDistributedOptimizer，`reload_masters` 只作用于 `self._inner`，连带删除 `zero_optimizer_state_for_adapter` 及其测试；但评论要求的双槽位回归测试未加入。
- 预集合通信失败导致 per-slot all-reduce 死锁 (correctness): 选择了 cell-fatal 路线：`step_slot_optimizers` 不再捕获异常，docstring 写明 `execution failures invalidate the cell`，新测试断言异常在报告成功前逃逸且失败槽位不 `allgather_params`；代价是单个槽位故障会拖垮整个 Tinker cell，且两 rank 测试未落地。
- review 点名要求的回归测试缺失 (testing): 槽位隔离与失败传播目前只被 fake 单测覆盖，真实 LayerWise/collective 行为没有回归保护，需要在后续 PR 或专门测试中补齐；body 的测试声明与 diff 口径也建议向作者确认。

## 风险与影响

- 风险：
  1. 共享路径回归面大：`miles/backends/megatron_utils/model.py` 是所有 Megatron 训练（整模、单 LoRA、critic）都会经过的文件，本次把 `train_one_step` 与 `train` 的代码搬家；抽取后的 `setup_train_iteration_config` 丢掉了原实现里的 `assert config.no_sync_func is None`（`overlap_grad_reduce` 场景），等于移除了“自定义 `no_sync_func` 不该被覆盖”的守卫，属于顺带的行为收窄，需要回归确认不会掩盖配置错误。
  2. 集合通信强耦合：`step_slot_optimizers` 依赖所有 rank 使用相同的 `sorted(adam_params_by_slot)` 顺序与相同槽位集合，否则会在 `get_grad_norm_fp32` 的 world all-reduce 上错位或卡死；当前没有 rank consensus，任何 pre-collective 失败都让 cell 失败——这是 review 两个选项中的一个，但恢复粒度确实变粗，单个坏槽位会拖垮整个 Tinker cell。
  3. 梯度处理语义：`clip_and_step` 在 norm 非有限时返回 `skipped_nonfinite`，随后 `step_slot_optimizers` 仍会对该槽位 `zero_grads()`，即异常 batch 的累积梯度被直接丢弃；上游必须把该 outcome 当作需要重放或显式失败来消费，否则数据会被静默吞掉，这一点要在 #2844/#2845 的网关实现里重点核对。
  4. 显存与启动约束：每个活跃租户各持一套 LayerWise 优化器与 FP32 master，显存随槽位数线性增长；`validate_multi_lora_optimizer_args` 又把 `use_distributed_optimizer`、`optimizer_cpu_offload`、`stream_optimizer_state_to_disk`、`rematerialize_param_from_master_weight` 等省显存 / 省内存开关全部禁掉，多租户并发下的显存上限很可能先于算力成为瓶颈。
  5. 单 cell 假设：`miles/ray/train/group.py` 的 `_execute_slots` 用 `(cell,) = self._cells`，多 LoRA 训练组必须恰好一个存活 cell，否则直接抛 `ValueError/RuntimeError`，与常规 rollout 组多 cell 并发模型不同，部署时需要显式约束。

6. 测试与 CI 缺口：新增测试全部基于 `_FakeSlotOptimizer`，不覆盖真实

`LayerWise/collective` 行为；review 点名要求的双槽位与两 rank 回归测试缺失；body 声明 fresh CPU/GPU CI pending，同时“未新增测试文件”的表述与 diff 中新增 `test_step_adapter_slots.py` 不完全一致，建议向作者确认测试落盘状态。

7. 小问题：`MultiLoRATrainRayActor.optim_step` 的返回类型标注是 `dict[int, float]`，而 `lora_model.optim_step` 实际返回 `dict[int, dict]` (`grad_norm` 或 `skipped_nonfinite`)，标注与真实契约不一致，容易误导后续网关调用方。- 影响：对用户与上层服务：多 LoRA 服务首次具备“命令式训练”接口，Tinker 网关 (#2844-#2849) 可以按请求驱动 `forward_backward/optim_step`，按租户下发 Adam 超参；普通整模与单 LoRA 训练的启动方式、参数与行为不变，但仍会经过被重构的 `model.py` 代码路径，因此回归验证需要覆盖非多 LoRA 场景。

对系统：多 LoRA 模式的启动校验明显收紧（必须 bf16、纯 Adam、关闭 dist opt 与 offload 系列开关），不满足会在初始化阶段直接断言失败，属于有意为之的显式失败；每个槽位的优化器与 FP32 master 使显存占用随活跃租户数线性增长，容量规划需要按最大并发槽位数预留。

对团队与运维：优化器状态的所有权从“池化 + 手工复位”变成“随槽位生命周期”，后续按槽位 checkpoint (#2842)、不可变快照发布 (#2843) 可以直接复用 `slot_optimizers` 注册表；但单 cell 约束、nonfinite 跳过的重放策略、以及 cell 级失败恢复粒度都需要网关侧与运维一起对齐。

影响程度：触及 Megatron 后端核心训练路径（优化器构造、FWD/BWD 抽函数、actor 初始化钩子）并新增一个模块，属于跨模块重构叠加新特性地基，影响面偏大；对非 megatron 后端与不启用 multi-LoRA 的部署无功能影响。

- 风险标记：核心训练路径重构，集合通信顺序强耦合，槽位故障拖垮整个 cell，显存随活跃租户数线性增长，缺少双槽位与多 rank 回归测试，合并前 fresh CI 未跑完

## 关联脉络

- PR #2841 [multi-lora v2 2/7] Per-batch loss selection and Tinker losses: 同一连发的下一环，改 `miles/backends/megatron_utils/lora/model.py` 与 `lora/actor.py`，在本次新增的 `forward_backward/optim_step` 命令面上叠加每 batch 损失选择，直接依赖这里的梯度累积契约。
- PR #2842 [multi-lora v2 3/7] Per-slot checkpoints with optimizer state: 复用本次建立的 `slot_optimizers` 注册表与 `lora/slots.py` 做按槽位 checkpoint，并改 `miles/ray/train/group.py` 转发层；本次把优化器状态绑定到槽位生命周期的决策是它的前提。
- PR #2843 [multi-lora v2 4/7] Publish immutable adapter snapshots to disk: 同样修改 `lora/actor.py` 与 `miles/ray/train/group.py`，槽位权重快照发布依赖本次的 load/unload 与 `SlotOptimizer` 所有权模型。
- PR #2844 [multi-lora v2 5/7] Tinker gateway protocol face: 协议层与每模型调度器需要消费本次定义的命令语义与 per-slot outcome (`grad_norm` / `skipped_nonfinite`)，是命令面的上游消费者。

- PR #2845 [multi-lora v2 6/7] Tinker gateway runtime and entry point: 网关 runtime 通过 miles/ray/train/group.py 的槽位命令转发调用本次新增的 actor 接口，并决定 nonfinite 跳过的重放策略。
- PR #3245 fix(megatron): respect --no-load-optim when resuming a LoRA adapter: 同一片区域的后续 bugfix，说明 LoRA/ 槽位优化器状态的加载与清零一直是高风险点；本次把状态所有权交给槽位生命周期正是对这一类问题的结构性回应。