PR #2296 完整报告

THUDM/slime

fix(train): skip optimizer and scheduler for eval-only

合并时间：2026-08-21 10:50

原文链接：http://prhub.com.cn/THUDM/slime/pull/2296

执行摘要

- 一句话：eval-only 跳过优化器与调度器构建
- 推荐动作：该 PR 值得精读，尤其是“为什么不用 train_iters=1”的设计权衡。建议在合入后补一条 CPU 侧测试：断言 num_rollout == 0 时 setup_model_and_optimizer 返回 (model, None, None)、create_training_models 不分配 critic，并验证正常训练路径仍构造 optimizer 与 scheduler。

功能与动机

PR body 明确指出 eval-only 是预期功能：train.py 已有特殊分支在 --num-rollout 0 时调用 rollout_manager.eval，train loop 是 range(..., num_rollout) 所以实际零训练步。但 bring-up 阶段 setup_model_and_optimizer 总是构造 Megatron 的 optimizer 与 OptimizerParamScheduler，默认 --lr-decay-iters 下 slime 计算 train_iters=0 → lr_decay_steps=0，触发 Megatron 断言 lr_decay_steps > 0，导致永远走不到 eval 分支。作者还解释了为何不用 #2109 的 train_iters=1 方案：那是对调度器撒谎，且 lr_warmup_steps >= lr_decay_steps 时仍会崩溃。

实现拆解

变更入口是 slime/backends/megatron_utils/model.py 的 setup_model_and_optimizer 和 slime/ray/placement_group.py 的 create_training_models。

1. setup_model_and_optimizer 在 get_model 之后新增 early return：当 args.num_rollout == 0 时，设置 args.no_load_optim = True 并返回 (model, None, None)。这一步绕过了 OptimizerConfig 组装、Adam 补丁、get_megatron_optimizer 和 get_optimizer_param_scheduler 等全部训练栈构造，同时通过 no_load_optim 让 checkpoint 加载跳过 optimizer 状态，与 ref/teacher 使用 optimizer=None 的既有路径保持一致。
2. 同步更新 setup_model_and_optimizer 和 initialize_model_and_optimizer 的返回类型标注，将 optimizer 与 scheduler 类型改为 MegatronOptimizer | None、OptimizerParamScheduler | None，明确新契约。
3. create_training_models 将 critic 创建条件从 args.use_critic 收紧为 args.use_critic and args.num_rollout != 0，避免 eval-only 时白白分配 critic 训练组；随后 start_rollout_ids 的来源判断从 args.use_critic 改为 critic_model is not None，避免

use_critic 与 num_rollout 组合时引用未定义的 critic_start_rollout_ids。

4. 测试配套：PR 最终未包含自动化测试。作者在第二笔 commit 中曾加入针对 train.py 控制流、critic 分配和 setup_model_and_optimizer 的 CPU 测试，但在第三笔 commit 中将其删除，理由是那只是本地验证脚本而非 slime 应保留的测试。

关键文件：

- slime/backends/megatron_utils/model.py（模块 后端训练；类别 source；类型 data-contract；符号 setup_model_and_optimizer, initialize_model_and_optimizer）：核心修复点：setup_model_and_optimizer 在 num_rollout == 0 时跳过 optimizer 与 scheduler 构造，避免 Megatron 因 lr_decay_steps == 0 断言失败；返回类型同步变为 Optional。
- slime/ray/placement_group.py（模块 训练编排；类别 source；类型 core-logic；符号 create_training_models）：create_training_models 在 num_rollout == 0 时跳过 critic 分配，并将 start_rollout_ids 的判断依据从 args.use_critic 改为 critic_model is not None，避免 eval-only 组合时引用未定义变量。

关键符号：setup_model_and_optimizer, initialize_model_and_optimizer, create_training_models

关键源码片段

slime/backends/megatron_utils/model.py

核心修复点：setup_model_and_optimizer 在 num_rollout == 0 时跳过 optimizer 与 scheduler 构造，避免 Megatron 因 lr_decay_steps == 0 断言失败；返回类型同步变为 Optional。

```
def setup_model_and_optimizer(
    args: Namespace, role: str = "actor"
) -> tuple[list[DDP], MegatronOptimizer | None, OptimizerParamScheduler | None]:
    # eval-only 模式（--num-rollout 0）不训练，不应构造优化器与调度器。
    # 若照常构造，Megatron 会因 lr_decay_steps == 0 触发 assert 失败。
    assert not args.moe_use_upcycling
    assert args.load is not None or args.pretrained_checkpoint is not None

    model = get_model(get_model_provider_func(args, role), ModelType.encoder_or_decoder)

    if args.num_rollout == 0:
        # 标记不加载 optimizer 状态，与 ref / teacher 的 checkpoint 加载路径保持一致。
        args.no_load_optim = True
        # optimizer 与 scheduler 均为 None，调用方按"无优化器"路径处理。
        return model, None, None

    # 正常训练路径：组装 OptimizerConfig，再构造 optimizer 与 scheduler。
    kwargs = {}
    for field in dataclasses.fields(OptimizerConfig):
        if hasattr(args, field.name):
            kwargs[field.name] = getattr(args, field.name)
```

```
    config = OptimizerConfig(**kwargs)
    config.timers = None

    # stateless Adam 需要替换 Adam 类并禁用分布式优化器状态初始化。
    optimizer_context = (
        _patch_megatron_adam(StatelessAdam) if args.use_stateless_adam else nullcontext()
    )
    with optimizer_context:
        optimizer = get_megatron_optimizer(
            config=config,
            model_chunks=model,
            use_gloo_process_groups=args.enable_gloo_process_groups,
        )
    if args.use_stateless_adam:
        _disable_distributed_optimizer_state_initialization(optimizer)
    opt_param_scheduler = get_optimizer_param_scheduler(args, optimizer)
    return model, optimizer, opt_param_scheduler
```

slime/ray/placement_group.py

create_training_models 在 num_rollout == 0 时跳过 critic 分配，并将 start_rollout_ids 的判断依据从 args.use_critic 改为 critic_model is not None，避免 eval-only 组合时引用未定义变量。

```
def create_training_models(args, pgs, rollout_manager, actor_cls=None):
    actor_model, actor_start_rollout_ids = create_actor_model(args, pgs, rollout_manager, actor_
    cls=actor_cls)

    critic_model = None
    # eval-only 模式不训练，无需分配 critic 训练组；仅当实际训练时创建。
    if args.use_critic and args.num_rollout != 0:
        from slime.utils.arguments import parse_megatron_role_args

        critic_args = (
            parse_megatron_role_args(args, args.megatron_config_path, role="critic")
            if args.megatron_config_path is not None
            else copy.deepcopy(args)
        )
        if args.megatron_config_path is None:
            critic_args.disable_param_buffers_cpu_backup = False

        critic_model = allocate_train_group(
            args=critic_args,
            num_nodes=args.critic_num_nodes,
            num_gpus_per_node=args.critic_num_gpus_per_node,
            pg=pgs["critic"],
            role="critic",
        )
        critic_start_rollout_ids = critic_model.create(rollout_manager=rollout_manager)
```

```
    # 以 critic_model 是否实际创建为判断依据，避免 use_critic 与 num_rollout 组合时的误判。
    if critic_model is not None:
        start_rollout_ids = critic_start_rollout_ids
    else:
        start_rollout_ids = actor_start_rollout_ids

    assert len(set(start_rollout_ids)) == 1

    if args.start_rollout_id is None:
        args.start_rollout_id = start_rollout_ids[0]

    if args.rollout_global_dataset:
        ray.get(rollout_manager.load.remote(args.start_rollout_id - 1))

    return actor_model, critic_model
```

评论区精华

仓库中未发现针对本 PR 的 review 评论。最有价值的讨论集中在 PR body 对 #2109 方案的反驳：

- 作者指出 train_iters=1 是“对调度器撒谎”，会让 train_iters 失去“实际训练步数”的语义，且 lr_warmup_steps >= lr_decay_steps 时仍然崩溃。
- 结论：eval-only 不应构造训练栈，直接跳过 optimizer 与 scheduler 才是正确方向。
- 该决策最终被合入，说明团队认可“非训练模式不建训练栈”的设计。
- 为何不用 train_iters=1 绕过 scheduler 构造 (design): 不采用 train_iters=1，改为 eval-only 直接不构造优化器与调度器。

风险与影响

- 风险：
  - 返回契约变更：setup_model_and_optimizer 与 initialize_model_and_optimizer 的返回值从固定三元 / 四元组变为可含 None。虽然 ref/teacher 加载路径已有 optimizer=None 的先例，但本 PR 未逐一排查其余调用方，若某处直接使用返回的 optimizer 属性（如 optimizer.param_groups）会触发 AttributeError。
  - 缺少测试覆盖：最终 PR 无自动化测试，--num-rollout 0 的 eval-only 启动与正常训练回归都依赖人工验证，存在回归风险。
  - critic 跳过逻辑：若配置 --use-critic 且 --num-rollout 0，create_training_models 不再创建 critic，start_rollout_ids 将退回 actor 的 id；若上层逻辑仍按 use_critic 期待 critic 的 rollout id，可能产生不一致。
  - 影响是正面的：eval-only 不再创建 optimizer、scheduler 和 critic，显存与启动开销下降；正常训练 num_rollout > 0 分支未改动，风险可控。
- 影响：
  - 对用户：修复了 --num-rollout 0 --eval-interval 1 纯评测场景的启动崩溃，用户无需再通过 hack train_iters 绕过断言。

- 对系统：eval-only 启动路径不再构造训练栈，内存占用更低，启动更快。
- 对团队：明确了 train_iters 的语义只属于真实训练；同时暴露了创建训练模型与评估模式组合时缺少自动化回归的问题。
- 风险标记：缺少测试覆盖，返回 Optional 契约变更，eval-only 启动路径

关联脉络

- PR #2109 （旧方案）train_iters=1 绕过：PR body 明确说明本 PR 是 #2109 的替代方案，弃用其 train_iters=1 做法。
- PR #2246 fix: cast gpu_id to int in sort_key to prevent lexicographic ordering: 同改 slime/ray/placement_group.py，训练编排路径的同类修复。
- PR #2170 Fix placement group crash for external engines under debug_rollout_only: 同改 slime/ray/placement_group.py，涉及训练模型创建流程的 placement group 修复。