PR #1866 完整报告

THUDM/slime

Rename critic config to megatron config

合并时间：2026-04-27 14:08

原文链接：http://prhub.com.cn/THUDM/slime/pull/1866

执行摘要

- 一句话：将 critic 配置重命名为通用 Megatron 角色配置
- 推荐动作：建议阅读该 PR，了解新的角色化配置系统。重点注意向后兼容性问题和新的 YAML 格式。如果团队中有使用旧配置的用户，需要提前沟通迁移计划。设计上角色缺失自动继承 CLI 参数的策略值得学习。

功能与动机

统一 actor 和 critic 的配置管理，避免维护两套独立的配置系统。通过角色化的 YAML 配置，用户可以在单一文件中指定 actor 和 critic 的不同参数（如学习率、并行策略），降低配置复杂度。

实现拆解

1. 修改 slime/utils/arguments.py：将 --critic-config-path 重命名为 --megatron-config-path；新增 _apply_megatron_role_overrides 函数，负责将 YAML 中的角色覆盖应用到参数副本；新增 parse_megatron_role_args 函数，解析顶层 megatron 键并根据角色返回覆盖后的参数。旧的 parse_critic_args 被移除。
2. 修改 slime/ray/placement_group.py 中的 create_training_models：在创建 actor 和 critic 模型之前，根据 megatron_config_path 加载对应角色的参数覆盖。actor 和 critic 的初始化参数改为使用覆盖后的副本。
3. 更新示例和测试脚本：将所有 --critic-config-path 替换为 --megatron-config-path，并将 YAML 配置格式更新为新的角色化结构。
4. 新增 tests/utils/test_megatron_role_config.py：覆盖了角色覆盖、缺失角色继承、顶层键验证以及 actor-only 场景的集成测试。
5. 新增 docs/en/advanced/megatron-config.md 和中文版文档，详细说明配置格式和使用方法，并更新了入门指南中的相关引用。

关键文件：

- slime/utils/arguments.py（模块 参数解析；类别 source；类型 core-logic；符号 parse_critic_args, _apply_megatron_role_overrides, parse_megatron_role_args）：核心文件，实现了配置重命名和通用角色解析逻辑。
- tests/utils/test_megatron_role_config.py（模块 角色配置测试；类别 test；类型 test-coverage；符号 _write_yaml, _base_args, TestMegatronRoleConfig,

test_parse_actor_and_critic_role_overrides）：新增的单元测试，覆盖了角色配置解析的核心路径。

- slime/ray/placement_group.py（模块 资源组；类别 source；类型 dependency-wiring）：展示如何使用新配置系统创建 actor 和 critic 模型。
- docs/en/advanced/megatron-config.md（模块 英文文档；类别 docs；类型 documentation）：新增的用户文档，详细说明新配置系统的设计和使用方法。
- tests/test_qwen3_4B_ppo.py（模块 集成测试；类别 test；类型 test-coverage）：集成测试示例，展示了新配置在训练脚本中的应用。

关键符号：parse_megatron_role_args, _apply_megatron_role_overrides, create_training_models

关键源码片段

tests/utils/test_megatron_role_config.py

新增的单元测试，覆盖了角色配置解析的核心路径。

```
import tempfile
from argparse import Namespace
import yaml

def _write_yaml(data: dict) -> str:
    """将字典写入临时 YAML 文件并返回路径。"""
    handle = tempfile.NamedTemporaryFile(mode="w", suffix=".yaml", delete=False)
    yaml.dump(data, handle)
    handle.flush()
    return handle.name

def _base_args(**overrides):
    """构造带有默认值的基底参数 Namespace。"""
    args = dict(
        lr=2e-6,
        tensor_model_parallel_size=1,
        kl_coef=0.1,
        use_kl_loss=False,
        use_opd=True,
        opd_type="megatron",
        custom_advantage_function_path="slime.test.adv",
        untie_embeddings_and_output_weights=False,
        actor_num_nodes=1,
        actor_num_gpus_per_node=1,
        critic_num_nodes=1,
        critic_num_gpus_per_node=1,
        use_critic=False,
        megatron_config_path=None,
        start_rollout_id=None,
        rollout_global_dataset=False,
    )
```

```
    args.update(overrides)
    return Namespace(**args)

class TestMegatronRoleConfig:
    def test_parse_actor_and_critic_role_overrides(self):
        """验证从 YAML 配置中分别解析 actor 和 critic 的角色覆盖。"""
        from slime.utils.arguments import parse_megatron_role_args

        path = _write_yaml({
            "megatron": [
                {"name": "default", "role": "critic", "overrides": {"lr": "1e-5", "tensor_model_parallel_
                size": 2}},
                {"name": "default", "role": "actor", "overrides": {"lr": "1e-6", "tensor_model_parallel_
                size": 4}},
            ]
        })
        args = _base_args()

        actor_args = parse_megatron_role_args(args, path, role="actor")
        critic_args = parse_megatron_role_args(args, path, role="critic")

        # 验证 actor 覆盖：lr 应为 1e-6，TP 为 4
        assert actor_args.lr == 1e-6
        assert actor_args.tensor_model_parallel_size == 4
        # actor 中未覆盖的继承自 base_args
        assert actor_args.kl_coef == args.kl_coef
        assert actor_args.use_opd is args.use_opd

        # 验证 critic 覆盖：lr 应为 1e-5，TP 为 2
        assert critic_args.lr == 1e-5
        assert critic_args.tensor_model_parallel_size == 2
        # critic 的 kl_coef 和 use_opd 等应有合理的默认值
        assert critic_args.kl_coef == 0
        assert critic_args.use_opd is False
        assert critic_args.custom_advantage_function_path is None
        assert critic_args.untie_embeddings_and_output_weights is True
```

slime/ray/placement_group.py

展示如何使用新配置系统创建 actor 和 critic 模型。

```
def create_training_models(args, pgs, rollout_manager):
    """根据 args 和 pgs 创建 actor 和 critic 的 RayTrainGroup。

    如果指定了 megatron_config_path，则分别加载 actor 和 critic 的角色覆盖，
    否则直接使用 CLI 参数。
    """
    # 准备 actor 参数：支持角色覆盖
    actor_args = args
    if args.megatron_config_path is not None:
```

```
    from slime.utils.arguments import parse_megatron_role_args
    actor_args = parse_megatron_role_args(args, args.megatron_config_path, role="actor")

actor_model = allocate_train_group(
    args=actor_args,
    num_nodes=args.actor_num_nodes,
    num_gpus_per_node=args.actor_num_gpus_per_node,
    pg=pgs["actor"],
)

critic_model = None
if args.use_critic:
    # 准备 critic 参数，同样支持角色覆盖
    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 args
    )
    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",
    )
    # 异步初始化 critic，并获取 rollout 起始 ID
    critic_start_rollout_ids = ray.get(critic_model.async_init(critic_model.args, role="critic",
    with_ref=False))

# 异步初始化 actor，使用可能被覆盖后的 actor_args
actor_start_rollout_ids = ray.get(
    actor_model.async_init(
        actor_args,
        role="actor",
        with_ref=actor_args.kl_coef != 0 or actor_args.use_kl_loss,
        with_opd_teacher=actor_args.use_opd and actor_args.opd_type == "megatron",
    )
)

# 根据是否有 critic 决定全局 rollout 起始 ID
if args.use_critic:
    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]
```

```
    # 设置 rollout manager
    actor_model.set_rollout_manager(rollout_manager)
    if args.use_critic:
        critic_model.set_rollout_manager(rollout_manager)

    # 如果需要加载全局数据集
    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

该 PR 没有公开的 Review 讨论，由作者直接合并。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 向后不兼容：新代码不再解析旧格式（顶层 critic 键），尽管帮助字符串声称兼容，但 parse_megatron_role_args 会严格检查顶层 megatron 键，导致旧配置直接报错。所有现有使用 --critic-config-path 的用户必须迁移到新格式。
  2. actor 参数行为变化：在 create_training_models 中，actor 的初始化参数由原始 args 变为通过 parse_megatron_role_args 返回的 actor_args。如果 YAML 中未定义 actor 角色，actor_args 是 args 的深拷贝，与以前相同。但若 YAML 中定义了 actor 覆盖，行为会改变，这符合预期。
  3. 缺乏对旧配置的迁移指南：虽然文档介绍了新格式，但没有提供从旧格式自动迁移的工具或详细步骤，用户需要手动修改 YAML 文件。
  4. 测试覆盖：新测试覆盖了正常场景，但未测试旧配置格式的报错信息是否符合预期，以及极端情况如空文件或格式错误。 - 影响：影响所有使用 Megatron 后端的 PPO 训练用户，尤其是那些依赖 --critic-config-path 自定义 critic 参数的用户。他们需要将 YAML 配置从 critic 键改为 megatron 键下的角色条目。由于该改动是突破性的，需要进行用户通知和文档更新。对系统本身，配置架构更统一，为未来支持更多角色（如 ref、OPD teacher）打下基础。 - 风险标记：向后不兼容，配置格式变更，核心路径变更

关联脉络

- PR #1856 refactor/ppo: 同一功能线的前置重构，该 PR 在 1856 的基础上进一步统一 actor-critic 配置。