

PR #6853 完整报告

verl-project/verl

[trainer, rollout] feat: log rollout moe load-balance metrics

合并时间: 2026-06-29 09:48

原文链接: <http://prhub.com.cn/verl-project/verl/pull/6853>

执行摘要

- 一句话: 新增 rollout MoE 负载均衡指标并移除旧 VeOmni 监控
- 推荐动作: 适合团队精读, 特别是 RolloutMoELoadBalanceMetricsAccumulator 的设计 (跨步聚合、配置驱动) 以及为兼容不同配置嵌套的专家数推断策略。建议关注其如何通过 accumulator 模式避免每次训练步都计算和传输大量数据, 而是批量处理。对需要实现类似指标聚合的模块有参考价值。

功能与动机

需要监控 MoE 模型在 rollout 阶段的路由负载均衡情况, 以识别专家利用率不均、路由策略偏差等问题。旧方案仅在 VeOmni engine 内实现, 无法与其他 rollout 后端 (如 vLLM) 共用, 且无法通过 trainer 统一指标路径上报。新设计从路由回放数据中提取专家分配信息, 由 trainer 聚合后报告。

实现拆解

1. 核心指标计算: 在 `verl/trainer/ppo/metric_utils.py` 中新增:
 - `infer_moe_num_experts`: 通过多层次候选配置结构 (`model_config`、`hf_config`、`text_config`、`override_config.model_config` 等) 推断全局路由专家数量, 兼容 `OmegaConf DictConfig` 和普通 `dict`。
 - `compute_rollout_moe_load_balance_metrics`: 对输入的 `routed_experts` 张量和 `response_mask`, 先截取 `response` 部分, 在 GPU 上按 `mask` 筛选有效 tokens, 再传输到 CPU 逐层统计专家调用次数, 最后计算每个路由层的 `max/min/avg violation` 以及全局聚合值。
 - `RolloutMoELoadBalanceMetricsAccumulator`: 一个跨训练步累积的容器, 内部维护各层专家累积计数。每次调用 `update` 传入 `routed_experts` 和 `response_mask`, 增量累加; 当累计步数达到 `moe_lb_metrics_interval` 时, `pop_metrics` 返回指标并重置计数。支持通过构造函数传入 `model_config` 自动推断专家数, 无需每步指定。
2. 集成到 trainer 流程: 在 `verl/trainer/ppo/v1/trainer_base.py` 中:
 - `PPOTrainer.__init__` 创建 `RolloutMoELoadBalanceMetricsAccumulator` 实例。
 - 在 `_compute_metrics` 中, 使用 `get_metric_data_with_optional_routed_experts` 包裹原有的 `kv_batch_get` 调用: 当配置 `interval > 0` 时, 自动将 "routed_experts" 加入获取字段列表; 若后端未提供该字段, 则日志记录并回退到基本字段。

- 随后调用 `compute_moe_lb_metrics`，传入 `batch` 数据、`interval`、`global_steps` 和 `accumulator`。该函数内部调用 `accumulator.update`，并在达到 `interval` 时调用 `pop_metrics` 生成指标，merge 到 `metrics` 字典中返回。

3. 配置变更：

- `verl/workers/config/rollout.py` 新增 `moe_load_balance_metrics_interval: int = 0`。
- 所有 `_generated_*.yaml` 配置文件中同步加入该字段（值为 0）。
- `verl/workers/config/engine.py` 中移除 `VeOmniEngineConfig` 的 `moe_load_balance_monitor_interval` 字段；`verl/trainer/config/engine/veomni.yaml` 同步删除该配置。

4. 清理旧代码：`verl/workers/engine/veomni/transformer_impl.py`：

- 删除 `_init_moe_monitor` 和 `_log_moe_metrics` 方法。
- 删除 `__init__` 中的 `_moe_monitor` 和 `_moe_monitor_step` 属性。
- 删除 `forward_backward_batch` 中对 `monitor` 的 `pause/resume` 调用。

5. 测试：新增 `tests/trainer/ppo/test_rollout_moe_lb_metrics_on_cpu.py`，包含：

- 验证 `compute_rollout_moe_load_balance_metrics` 对正常输入输出正确 `violation` 值。
- 验证无效输入（`None`、维度不足、`num_experts` 为 `None`）返回空字典。
- 验证 `accumulator` 跨更新累积并正确弹出指标，以及弹出后状态归零。
- 验证 `infer_moe_num_experts` 能从多种配置结构中正确提取值。
- 验证 `compute_moe_lb_metrics` 在未达到 `interval` 时不产生指标，达到 `interval` 后返回正确指标。
- 验证 `get_metric_data_with_optional_routed_experts` 在缺失 `routed_experts` 时的回退行为。
- 验证 `keyed warning` 去重（每种缺失只警告一次）。

关键文件：

- `verl/trainer/ppo/metric_utils.py`（模块 训练器；类别 `source`；类型 `dependency-wiring`；符号 `_get_nested_attr`, `get_hf_config_override_kwargs`, `_get_config_value`, `infer_moe_num_experts`）：核心实现文件，包含 MoE 指标计算函数和累积器类
- `tests/trainer/ppo/test_rollout_moe_lb_metrics_on_cpu.py`（模块 测试；类别 `test`；类型 `test-coverage`；符号 `test_rollout_moe_load_balance_metrics_response_tokens_only`, `test_rollout_moe_load_balance_metrics_skips_invalid_inputs`, `test_rollout_moe_load_balance_accumulator_spans_updates`, `test_rollout_moe_load_balance_accumulator_infers_num_experts`）：全面覆盖新功能的单元测试
- `verl/workers/engine/veomni/transformer_impl.py`（模块 推理引擎；类别 `source`；类型 `dependency-wiring`；符号 `_init_moe_monitor`, `_log_moe_metrics`）：移除旧的 VeOmni 端 MoE 监控代码
- `verl/trainer/ppo/v1/trainer_base.py`（模块 训练器；类别 `source`；类型 `core-logic`；符号 `_compute_metrics`）：集成 MoE 指标进 v1 trainer 训练循环

- verl/workers/config/engine.py (模块 配置; 类别 source; 类型 core-logic) : 移除旧配置项
- verl/workers/config/rollout.py (模块 配置; 类别 source; 类型 core-logic) : 新配置项定义

关键符号: infer_moe_num_experts, infer_rollout_moe_num_experts, compute_rollout_moe_load_balance_metrics, RolloutMoELoadBalanceMetricsAccumulator, compute_moe_lb_metrics, get_metric_data_with_optional_routed_experts, _get_nested_attr, get_hf_config_override_kwargs

关键源码片段

tests/trainer/ppo/test_rollout_moe_lb_metrics_on_cpu.py

全面覆盖新功能的单元测试

```
# test_rollout_moe_lb_metrics_on_cpu.py: 测试指标计算

def test_rollout_moe_load_balance_metrics_response_tokens_only():
    # routed_experts shape: (batch=1, layers=6, seq_len=2, top_k=1)
    routed_experts = torch.tensor(
        [
            [
                [[0, 1], [0, 0]], # layer 0: token0->0,1; token1->0,0
                [[0, 1], [0, 0]], # layer 1
                [[0, 1], [2, 2]],
                [[2, 3], [2, 2]],
                [[0, 1], [2, 2]],
                [[2, 3], [2, 2]],
            ]
        ],
        dtype=torch.long,
    )
    # response_mask: batch=1, response_len=4 (全部有效)
    response_mask = torch.tensor([[True, True, True, True]], dtype=torch.bool)

    metrics = compute_rollout_moe_load_balance_metrics(
        routed_experts=routed_experts,
        response_mask=response_mask,
        num_experts=4,
    )

    # layer 0: 专家分配均匀, vio 为 0
    assert math.isclose(metrics["rollout/moe/max_vio/layer_0"], 0.0)
    assert math.isclose(metrics["rollout/moe/min_vio/layer_0"], 0.0)
    assert math.isclose(metrics["rollout/moe/avg_vio/layer_0"], 0.0)
    # layer 1: 分配不均, 期望特定值
    assert math.isclose(metrics["rollout/moe/max_vio/layer_1"], 3.0)
```

```

assert math.isclose(metrics["rollout/moe/min_vio/layer_1"], -1.0)
assert math.isclose(metrics["rollout/moe/avg_vio/layer_1"], 1.5)
# 全局聚合
assert math.isclose(metrics["rollout/moe/max_vio/max"], 3.0)
assert math.isclose(metrics["rollout/moe/avg_vio/avg"], 0.75)

```

verl/trainer/ppo/v1/trainer_base.py

集成 MoE 指标进 v1 trainer 训练循环

```

# trainer_base.py 导入新增
from verl.trainer.ppo.metric_utils import (
    RolloutMoELoadBalanceMetricsAccumulator,
    compute_moe_lb_metrics,
    get_metric_data_with_optional_routed_experts,
    compute_data_metrics,
    compute_throughout_metrics,
    compute_timing_metrics,
    compute_variance_proxy_metrics,
    process_validation_metrics,
)

class PPOTrainer(ABC):
    def __init__(self, config: DictConfig):
        # ... 已有初始化
        self._rollout_moe_lb_metrics_accumulator = RolloutMoELoadBalanceMetricsAccumulator(
            model_config=self.config.actor_rollout_ref.model
        )

    def _compute_metrics(self, batch, metrics, timing_raw, global_steps):
        # ... 原有代码
        moe_lb_metrics_interval = self.config.actor_rollout_ref.rollout.get(
            "moe_load_balance_metrics_interval", 0
        )
        # 使用包装函数获取数据，自动附加 routed_experts 字段
        data = get_metric_data_with_optional_routed_experts(
            keys=batch.keys,
            partition_id=batch.partition_id,
            fields=fields,
            moe_lb_metrics_interval=moe_lb_metrics_interval,
            global_steps=global_steps,
            accumulator=self._rollout_moe_lb_metrics_accumulator,
            kv_batch_get=tq.kv_batch_get,
        )
        # ... 处理其他数据
        metrics_batch = self._build_metrics_batch(data)
        # 计算 MoE 指标（如果达到 interval 则返回非空字典）
        metrics.update(
            compute_moe_lb_metrics(
                metrics_batch=metrics_batch,

```

```

        moe_lb_metrics_interval=moe_lb_metrics_interval,
        global_steps=global_steps,
        accumulator=self._rollout_moe_lb_metrics_accumulator,
    )
)
# ... 其他指标计算和返回

```

评论区精华

- `_get_nested_attr` 兼容 OmegaConf: gemini-code-assist 指出使用 `isinstance(obj, dict)` 不够健壮, 因为 OmegaConf DictConfig 不继承 dict 但有 `.get()` 方法, 建议改用 `hasattr(obj, 'get')`。已采纳修改。
- GPU 端 mask 减少传输: gemini-code-assist 建议在 `compute_rollout_moe_load_counts` 中先对 `response_routed_experts` 在 GPU 上应用 mask, 再传输筛选后的结果到 CPU, 以减少传输量。已修改为 `response_routed_experts[response_mask.bool()].detach().to(device='cpu')`。
- 将 helper 移出 `trainer_base.py`: wuxibin89 要求将 MoE 相关的辅助函数移至 `metric_utils.py`, 保持 `trainer_base.py` 简洁。作者通过多次 commit 逐步迁移, 最终 `trainer_base.py` 仅保留必要集成点。
- 配置访问使用 `.get()`: wuxibin89 建议使用 `self.config.actor_rollout_ref.rollout.get('moe_load_balance_metrics_interval', 0)` 而非 `getattr`。已修改。
- 将 `routed_experts` 附加到 `fields`: wuxibin89 建议避免单独 `kv_batch_get`, 而是将 `routed_experts` 附加到 `fields` 中。作者实现了 `get_metric_data_with_optional_routed_experts` 封装该逻辑。
 - `_get_nested_attr` 兼容 OmegaConf DictConfig (design): 已接受并修改代码, 改用 `hasattr(obj, 'get')`。
 - GPU 端 mask 减少 host-device 传输 (performance): 已修改为 `response_routed_experts[response_mask.bool()].detach().to(device='cpu')`, 先 mask 再 transfer。
 - 将 helper 移出 `trainer_base.py` 保持简洁 (design): 作者通过多次 commit 逐步将函数迁移到 `metric_utils.py`, 最终 `trainer_base.py` 只保留必要的集成点。
 - 使用 `.get()` 访问配置而非 `getattr` (style): 已修改代码。

风险与影响

- 风险:
 - 默认关闭, 开启需显式配置, 因此对现有用户无影响。
 - 但修改了 `trainer_base.py` 的核心 `_compute_metrics` 方法, 可能引入 regression。需要确保旧路径不受影响。
 - 移除 VeOmni 端 MoE monitor 后, 如果遗留脚本仍使用旧配置项 `moe_load_balance_monitor_interval`, 会导致配置错误。已通过删除配置字段使错误显式化。

- 推断专家数的逻辑依赖于模型配置，若配置具有不常见的嵌套方式（如不在 standard 路径中），可能返回 None，导致指标缺失。代码设计了级联回退，但仍有未覆盖情况。
- 跨步聚合使用 TransferQueue 的 KVBatchMeta，需要确保 routed_experts 在 batch 中可用。代码提供了 fallback 和诊断日志。
- 影响：
 - 用户：需设置 rollout.moe_load_balance_metrics_interval > 0 才能启用指标，指标出现在 tracker 的 rollout/moe/* 路径下。对非 MoE 模型无意义。
 - 性能：仅在启用且达到 interval 时才有额外计算和传输开销，可忽略。
 - 团队：移除了 VeOmni 专用 monitor，简化 engine 代码，统一指标路径。
 - 风险标记：默认关闭，配置项移除，核心路径变更，回退机制

关联脉络

- PR #6823 [BREAKING][trainer, cfg] chore: enable V1 trainer by default: 同时修改了 trainer_base.py 的 _compute_metrics 方法，存在合并冲突风险。