

PR #7781 完整报告

verl-project/verl

[trainer] fix: count failed validation sessions in accuracy

合并时间: 2026-09-15 15:12

原文链接: <http://prhub.com.cn/verl-project/verl/pull/7781>

执行摘要

- 一句话: V1 验证指标把失败会话按 `acc=0` 计入
- 推荐动作: 值得精读。这是一个“小 diff、大语义”的指标修正: 104 行改动里的核心只有三处 (期望计数采集、源级 opt-in 补齐、None/has_pred 安全化), 但直接改变了 Trainer V1 验证主指标的语义, 阅读时建议重点看 `metric_utils.py` 的 `expected_acc_counts` 块与 `has_pred` 的判定条件, 以及 `trainer_base.py` 中 `(str(data_source), str(uid))` 键的构造是否与指标侧分组严格对齐。建议配套动作: 在监控中同时画 `acc` 与 `acc_success_only`, 把两者差值作为失败率诊断; 为 `_validate` $\rightarrow$ `_val_metrics_update` 补一个 CPU 级端到端单测 (覆盖 `__rollout_n__` 缺省回退与键一致性); 明确 `pred` 是否也应随 `acc` 补齐, 避免 `maj` 指标与 `mean` 基数不一致。

功能与动机

PR body 明确说明动机: *Fixes V1 validation accuracy when failed agent-loop sessions do not materialize in TransferQueue. Missing sessions now count as acc=0.* 也就是说, 在 V1 (异步、agent-loop) 链路中, 会话失败后不会在 `TransferQueue/ReplayBuffer` 中出现任何行, `sample_uids / data_sources` 里自然没有它的位置, `process_validation_metrics` 只能对“成功留下的样本”求均值, 于是 `val-core` 准确率被高估, 而这个指标正是用于模型选择、早停和 `recipe` 对比的关键信号。改动希望恢复“失败即错误”的严格口径, 同时不丢失旧口径以便对照排查。

实现拆解

1. 验证入口记录“期望分母” (`verl/trainer/ppo/v1/trainer_base.py`, `_validate`): 新增 `expected_acc_counts: dict[tuple[str, str], int]`, 在每个 `val batch` 里读取 `batch_dict["uid"]`、`data_source` (缺省 "unknown") 与 `__rollout_n__` (缺省 `self.config.actor_rollout_ref.rollout.val_kwargs.n`), 用 `zip(..., strict=True)` 逐样本写入 `(str(data_source), str(uid))  $\rightarrow$  int(rollout_n)`, 最后透传给 `_val_metrics_update`。原因: 指标侧唯一缺失的信息就是“这条 prompt 本该产出几条会话”, 只能在提交 `rollout` 之前记录下来; 影响: 该字典在整个验证集上累积, 供后续所有指标计算复用。
2. 指标侧按源补齐失败会话 (`verl/trainer/ppo/metric_utils.py`, `process_validation_metrics`): 签名新增 `expected_acc_counts` 参数 (默认 `None`, 非 V1 调用方行为完全不变)。先计算 `acc_data_sources`——只有存在非 `None` 的 `acc` 值的源才参与补齐, 避免 `reward-only` 评测源因为别的源有 `acc` 而被凭空造出准确率; 随后对每个期望条目取出实际

观测值 `observed_acc`，写入 `acc_success_only`，并用 `missing_count`

`=max(expected_count - len(observed_acc), 0)` 追加 0.0，形成新的 `acc`。原因：

`max(...,0)` 同时防住了成功会话数超出预期（例如重复上报）时的负补齐；影响：完全失败的 `uid` 会通过 `defaultdict` 被“创建”出来并只带 `acc/acc_success_only` 两个变量，参与后续跨 `uid` 平均，从而真正拉低 `acc`。

3. 聚合前做 `None` 安全化并收紧 `has_pred` 判定：`flatten` 循环内先执行 `var_vals = [value for value in var_vals if value is not None]`，再判断空值 / 字符串；`has_pred` 由原来的单次 `pred_vals is not None` 改为“`pred_vals` 非空 + 与过滤后的 `var_vals` 等长 + 所有 `pred` 非 `None`”，并下沉到变量循环内逐变量计算。原因：补齐路径会写入空列表（`acc_success_only`）与 0.0，原始 `acc` 列也可能带 `None`（见新单测里的 `{"acc": [1.0, None]}`），不先过滤 `None` 会在 `np.mean/np.std` 处直接抛错；同时长度不一致时继续做 `majority vote` 会导致下标错位。
4. 指标分区与命名（`trainer_base.py` 的 `_val_metrics_update`）：`core_var = "acc" if "acc" in var2metric2val else "reward"` 的分支不变，但补齐后 `acc` 必然存在，因此 `val-core` 仍锚定严格口径的 `acc`；`acc_success_only` 只会进入 `val-aux/<source>/acc_success_only/mean@N`，与 `commit message` 描述一致。
5. 测试配套（`tests/trainer/ppo/test_metric_utils_on_cpu.py`）：新增 `test_process_validation_metrics_counts_missing_sessions_as_incorrect`（2 条成功、期望 4 条 → `acc/mean@4 = 0.25`，`acc_success_only/mean@2 = 1.0`，`reward` 不受影响）与 `test_process_validation_metrics_does_not_add_acc_to_reward_only_source`（`reward_source` 不得出现 `acc/acc_success_only`）。需要注意：PR body 给出的验证命令还包含 `tests/trainer/ppo/v1/test_replay_buffer_on_cpu.py`，但该文件并未在本 PR 中改动，`trainer` 侧的接线（字段读取、键构造）没有端到端测试覆盖。

关键文件：

- `verl/trainer/ppo/metric_utils.py`（模块 指标计算；类别 `source`；类型 `core-logic`；符号 `process_validation_metrics`）：指标计算核心：`process_validation_metrics` 新增 `expected_acc_counts` 并按源补齐失败会话，同时把 `var_vals` 的 `None` 过滤与 `has_pred` 收紧作为前置条件，决定了新口径的正确性与边界行为。
- `verl/trainer/ppo/v1/trainer_base.py`（模块 训练主循环；类别 `source`；类型 `core-logic`；符号 `_validate`，`_val_metrics_update`）：验证入口：`_validate` 负责在提交 `rollout` 前记录每条 `prompt` 的期望会话数，`_val_metrics_update` 负责把参数透传并决定 `val-core` 锚定 `acc`，是补齐口径的数据来源。
- `tests/trainer/ppo/test_metric_utils_on_cpu.py`（模块 指标单测；类别 `test`；类型 `test-coverage`；符号 `test_process_validation_metrics_counts_missing_sessions_as_incorrect`，`test_process_validation_metrics_does_not_add_acc_to_reward_only_source`）：新增两个 CPU 单测锁定新语义：缺失会话计入 `acc`、`reward-only` 源不得出现 `acc`，是本次口径变更的唯一回归保护。

关键符号：`_validate`，`_val_metrics_update`，`process_validation_metrics`，`test_process_validation_metrics_counts_missing_sessions_as_incorrect`，`test_process_validation_metrics_does_not_add_acc_to_reward_only_source`

关键源码片段

verl/trainer/ppo/metric_utils.py

指标计算核心: `process_validation_metrics` 新增 `expected_acc_counts` 并按源补齐失败会话, 同时把 `var_vals` 的 `None` 过滤与 `has_pred` 收紧作为前置条件, 决定了新口径的正确性与边界行为。

```
# verl/trainer/ppo/metric_utils.py
# 新增入参: 按 (data_source, uid) 记录该分组“本应产出”的会话数; 默认 None
# 旧调用方行为完全不变
def process_validation_metrics(
    data_sources: list[str],
    sample_uids: list[str],
    infos_dict: dict[str, list[Any]],
    seed: int = 42,
    expected_acc_counts: dict[tuple[str, str], int] | None = None,
) -> dict[str, dict[str, dict[str, float]]]:
    # 1. 先按 data_source -> uid -> var_name 三层聚合; uid 即一条 prompt (GRPO group)
    data_src2uid2var2vals = defaultdict(lambda: defaultdict(lambda: defaultdict(list)))
    for sample_idx, data_source in enumerate(data_sources):
        uid = sample_uids[sample_idx]
        var2vals = data_src2uid2var2vals[data_source][uid]
        for var_name, var_vals in infos_dict.items():
            var2vals[var_name].append(var_vals[sample_idx])

    if expected_acc_counts:
        # 只让“真正产出 acc”的评测源参与补齐: 纯 reward 的源不能因为邻居有 acc 就凭空多出一个 acc
        acc_data_sources = {
            data_source
            for data_source, uid2var2vals in data_src2uid2var2vals.items()
            if any(value is not None for var2vals in uid2var2vals.values() for value in var2vals.get("acc", []))
        }
        for (data_source, uid), expected_count in expected_acc_counts.items():
            if data_source not in acc_data_sources:
                continue

            # defaultdict 访问会为“全部会话都失败的 uid”创建空条目, 其 acc 最终全为 0
            var2vals = data_src2uid2var2vals[data_source][uid]
            observed_acc = [value for value in var2vals.get("acc", []) if value is not None]
            # 旧口径 (仅统计成功会话) 保留为 acc_success_only, 最终落到 val-aux 供对照
            var2vals["acc_success_only"] = observed_acc
            # 新口径: 缺失会话按 acc=0 计入主指标; max(..., 0) 防止成功数超出预期时产生负补齐
            missing_count = max(expected_count - len(observed_acc), 0)
            var2vals["acc"] = observed_acc + [0.0] * missing_count

    # ... 中间为 ns 缓存、bootstrap 采样等未改动逻辑 ...
```

```

# 4. flatten loop (关键差异: None 过滤 + has_pred 收紧)
for data_source, uid2var2vals in data_src2uid2var2vals.items():
    uid_dict = data_src2uid2var2metric.setdefault(data_source, {})

    for uid, var2vals in uid2var2vals.items():
        pred_vals = var2vals.get("pred")
        var_dict = uid_dict.setdefault(uid, {})

        for var_name, var_vals in var2vals.items():
            # 先剔除 None: 补齐路径写入的 acc_success_only 可能是空列表, 原始 acc
            # 列也可能带 None,
            # 不过滤会让后面的 np.mean / np.std 直接抛错
            var_vals = [value for value in var_vals if value is not None]
            # skip empty or string values
            if not var_vals or isinstance(var_vals[0], str):
                continue
            # has_pred 由“pred 存在”改为“与过滤后的 var_vals 等长且无 None”:
            # 补齐后的 acc 比 pred 长, 此时该 uid 主动放弃 maj 指标, 避免投票下标错位
            has_pred = (
                pred_vals is not None
                and len(pred_vals) == len(var_vals)
                and all(pred is not None for pred in pred_vals)
            )

            # compute mean and std
            n_resps = len(var_vals)
            metric = {f"mean@{n_resps}": float(np.mean(var_vals))}

```

verl/trainer/ppo/v1/trainer_base.py

验证入口: `_validate` 负责在提交 rollout 前记录每条 prompt 的期望会话数,
`_val_metrics_update` 负责把参数透传并决定 `val-core` 锚定 `acc`, 是补齐口径的数据来源。

```

# verl/trainer/ppo/v1/trainer_base.py
def _validate(self) -> dict[str, float]:
    # ... 采样收集列表略 ...
    session_to_sample_idx: dict[str, int] = {}
    # 新增: 记录每条 prompt 期望产出的会话数, 作为验证指标的“分母”
    expected_acc_counts: dict[tuple[str, str], int] = {}

    for batch_dict in self.val_data_loader:
        batch_dict["uid"] = np.array(
            [str(uuid.uuid4()) for _ in range(len(batch_dict["raw_prompt"]))], dtype=object
        )
        batch_size = len(batch_dict["uid"])
        # data_source 缺省为 "unknown", 必须与指标侧分组键保持同一口径
        batch_data_sources = batch_dict.get("data_source", ["unknown"] * batch_size)
        # 单样本 rollout n: 优先取数据集提供的 __rollout_n__, 否则回退到 val_kwargs.n
        batch_rollout_ns = batch_dict.get(
            "__rollout_n__", [self.config.actor_rollout_ref.rollout.val_kwargs.n] * batch_size
        )

```

```

    )
    for uid, data_source, rollout_n in zip(
        batch_dict["uid"], batch_data_sources, batch_rollout_ns, strict=True
    ):
        rollout_n = int(rollout_n)
        # 键必须是 (str(data_source), str(uid)), 与指标侧 data_sources / sample_uids
        # 的字符串值严格对齐
        expected_acc_counts[(str(data_source), str(uid))] = rollout_n

    # ... 提交 TransferQueue、采样 replay buffer、计算 reward、收集日志字段等逻辑略 ...

    return self._val_metrics_update(
        data_sources,
        sample_uids,
        reward_extra_infos_dict,
        sample_turns,
        expected_acc_counts=expected_acc_counts,
    )

def _val_metrics_update(
    self,
    data_sources,
    sample_uids,
    reward_extra_infos_dict,
    sample_turns,
    expected_acc_counts: dict[tuple[str, str], int] | None = None,
) -> dict[str, float]:
    data_src2var2metric2val = process_validation_metrics(
        data_sources,
        sample_uids,
        reward_extra_infos_dict,
        expected_acc_counts=expected_acc_counts,
    )
    metric_dict = {}
    for data_source, var2metric2val in data_src2var2metric2val.items():
        # 补齐后 acc 必然存在, val-core 继续锚定严格口径的 acc;
        # acc_success_only 只会走 else 分支进入 val-aux
        core_var = "acc" if "acc" in var2metric2val else "reward"
        # ... 指标命名与分区逻辑略 ...

    return metric_dict

```

tests/trainer/ppo/test_metric_utils_on_cpu.py

新增两个 CPU 单测锁定新语义：缺失会话计入 acc、reward-only 源不得出现 acc，是本次口径变更的唯一回归保护。

```

# tests/trainer/ppo/test_metric_utils_on_cpu.py
def test_process_validation_metrics_counts_missing_sessions_as_incorrect(self):
    # prompt1 实际只有 2 条成功会话，但按 expected_acc_counts 应有 4 条

```

```

data_sources = ["source1", "source1"]
sample_uids = ["prompt1", "prompt1"]
infos_dict = {
    "reward": [0.4, 0.6],
    "acc": [1.0, 1.0],
}

result = process_validation_metrics(
    data_sources,
    sample_uids,
    infos_dict,
    expected_acc_counts={
        ("source1", "prompt1"): 4,
        ("source1", "prompt2"): 4, # 该 prompt 完全失败，指标侧此前完全看不到它
    },
)

# prompt1 补齐为 [1, 1, 0, 0] → 0.5; 完全失败的 prompt2 记 0.0; 跨 uid 平均后为 0.25
self.assertAlmostEqual(result["source1"]["acc"]["mean@4"], 0.25)
# 旧口径只对已成功产出的 2 条会话取均值，仍是 1.0
self.assertAlmostEqual(result["source1"]["acc_success_only"]["mean@2"], 1.0)
# reward 不受 acc 补齐影响，仍按 2 条会话统计
self.assertAlmostEqual(result["source1"]["reward"]["mean@2"], 0.5)

def test_process_validation_metrics_does_not_add_acc_to_reward_only_source(self):
    # acc_source 有 acc 语义 → 参与补齐; reward_source 只有 reward → 不应出现任何 acc 指标
    result = process_validation_metrics(
        data_sources=["acc_source", "reward_source"],
        sample_uids=["acc_prompt", "reward_prompt"],
        infos_dict={"reward": [1.0, 0.5], "acc": [1.0, None]},
        expected_acc_counts={
            ("acc_source", "acc_prompt"): 2,
            ("reward_source", "reward_prompt"): 2,
        },
    )

    self.assertAlmostEqual(result["acc_source"]["acc"]["mean@2"], 0.5) # (1.0 + 0.0) / 2
    self.assertAlmostEqual(result["acc_source"]["acc_success_only"]["mean@1"], 1.0)
    self.assertNotIn("acc", result["reward_source"])
    self.assertNotIn("acc_success_only", result["reward_source"])

```

评论区精华

本 PR 在平台上没有任何 review 评论 (`comments_count=0`、`review_comments_count=0`)，仅由 wuxibin89 直接 APPROVED 并合并，因此没有可供提炼的技术交锋。唯一被记录下来的设计表述来自作者自己的 commit message: `fix: val-core now pads failure sessions with acc=0, acc_success_only is moved to val-aux`——它说明了两点决策：主指标接受“更

严格、数值更低”的口径，同时保留旧口径到辅助命名空间。由于缺少评讨论，以下几处设计取舍没有留下论证痕迹：为什么不把 `pred` 一起补齐、为什么不把“整个 data source 全部失败”的情况也纳入统计、以及新键 (`str(data_source)`, `str(uid)`) 与指标侧分组键的一致性如何保证。

- 审核无评论即获批：`acc` 口径变更缺少讨论记录 (design)：变更被快速放行，没有评审质疑；但“是否应把 `pred` 一并补齐以保留 `maj` 指标”“data source 全部失败时的残余盲区”“(str(data_source), str(uid)) 键的一致性保障”等取舍均未在评审中被讨论。

风险与影响

- 风险：

1. 指标口径变更带来的读数与基线断裂（高确定性）：任何存在失败会话的任务，`val-core/<source>/acc/mean@N` 都会低于历史值；失败率越高下降越明显。依赖该指标做模型选择 / 早停 / recipe 对比的团队必须重建基线，否则会把“口径修正”误判为模型退化。
2. 键一致性的静默降级风险 (`trainer_base.py` 用 (`str(data_source)`, `str(uid)`), `metric_utils.py` 用 `data_sources[sample_idx] / sample_uids[sample_idx]` 分组)：两侧只要有一处格式不一致（缺省 "unknown" 与其它缺省值不同、uid 被改写为 {uid}_{session}_{idx} 形式、bytes/np.str_ 字符串化差异），补齐就会写到“幽灵 uid”上，主指标被 0 稀释而不是修正。好在新代码同时输出 `acc` 与 `acc_success_only`，两者差值的异常放大可以作为对账信号；但这需要额外的监控动作，代码本身不会报错。
3. `acc_data_sources` 的源级过滤留下残余盲区：该集合来自“已聚合数据”里出现过非 `None` 的 `acc`，若某个 data source 的所有会话全部失败，它连分组都不会出现，补齐逻辑会跳过它，该源的失败仍然不可见（虽然不会污染其它源）。
4. `pred` 未同步补齐导致的指标降级：补齐只作用于 `acc`，`pred_vals` 长度仍是成功会话数，新 `has_pred` 判定（等长 + 无 `None`）会因此为 `False`，该 uid 不再产出 `maj@N/mean`、`maj@N/std`。跨 uid 聚合后，`maj` 的样本基数会少于 `mean`，`val-core` 中的 `maj` 序列可能在失败率变化时出现缺口或跳变。
5. 输入结构假设变严（兼容性）：`zip(batch_dict["uid"], batch_data_sources, batch_rollout_ns, strict=True)` 要求 `data_source` 与 `__rollout_n__` 都是逐样本序列；自定义数据集若把其中任一字段写成标量或长度不匹配，验证阶段会直接抛异常（fail-fast 虽好，但属于行为变化）。
6. 性能与内存：每次验证额外维护一个 (`data_source`, `uid`) 映射并做 $O(N)$ 补齐，相对 rollout 开销可忽略；仅在超大验证集上字典内存略有增长。- 影响：用户 / 研究者：`val-core` 准确率从“成功样本口径”变为“含失败样本口径”，更能反映真实端到端成功率，但数值会下降，历史曲线不可直接续接；`acc_success_only` 提供了过渡对照。系统：改动完全落在验证路径与指标计算，不触碰 `reward`、`advantage`、梯度与 rollout 调度，训练动态不受影响，风险面窄。团队 / 流程：所有引用 `val-core` 的仪表盘、recipe 基线、回归阈值都需要复核；同时因缺少 trainer 侧接线测试，未来若调整 `_validate` 中 `data_source/__rollout_n__` 的读取方式，容易悄悄破坏补齐口径。- 风险标记：验证指标口径变更，`val-core` 数值将下降，键一致性静默降级，缺少 trainer 接线测试，`maj` 指标可能被跳过

关联脉络

- PR #7722 [perf, tool] fix: Avoid overlapping rollout and training profilers in Trainer V1: 同样改动 verl/trainer/ppo/v1/trainer_base.py 的 _validate/ 验证路径，并补充 V1 验证相关 CPU 单测，与本 PR 属于同一条 V1 验证主循环的维护脉络。
- PR #7762 [trainer, ckpt] fix: skip V1 async warmup after inflight reissue: 同样围绕 verl/trainer/ppo/v1/trainer_base.py 及其 async 子类的 V1 训练 / 恢复逻辑，并配套 CPU 单测，为本 PR 提供同文件的上下文演进。
- PR #7792 [trainer, cfg] feat: filter V1 DAPO groups by canonical reward by default: 同样处理 V1 训练中“哪些样本 / 奖励口径参与统计”的语义问题（replay buffer + 配置 + 单测），与本 PR 对验证指标口径的修正属于同一类“指标与分组语义”演进。