PR #2200 完整报告

radixark/miles

[RL] Add sampling-support log-prob primitives

合并时间：2026-08-24 10:27

原文链接：http://prhub.com.cn/radixark/miles/pull/2200

执行摘要

- 一句话：新增采样支持 log-prob 原语，为 on-policy 归一化铺路
- 推荐动作：值得精读，重点看三处设计决策：（1）RolloutSamplingMask 为什么用 CSR 而非嵌套结构——这是「逻辑模型 vs 传输模型」权衡的典型范本；（2）_apply_sampling_mask 的 inplace 与非 inplace 分流，以及 true-on-policy 下 log-prob 用 mask 而 entropy 保持全 vocab 的语义对齐；（3）_iter_response_chunks 如何在不破坏旧契约的前提下为 CP 各模式补充全局响应索引。建议与 #2595、#2596 一起阅读以还原完整功能闭环。

功能与动机

PR body 明确指出：「The trainer needs a correct way to normalize actor log-probabilities over the same sampling support used during rollout」——即 rollout 端若启用 top-p 等采样 mask，actor 打分端必须使用相同采样支持做 log-prob 归一化，否则 on-policy 与 off-policy 的 log-prob 分布不一致会污染 PPO 优势估计。因此该原语必须在 SGLang capture 激活之前合入，且刻意「No production caller supplies a sampling mask in this PR, so rollout and training behavior remain unchanged」，保证合入时零行为影响。实现参考了 THUDM/slime#2102 的 top_p masking 方案并致谢作者 zhuzilin。

实现拆解

本 PR 的变更入口是 loss 计算链路的两个核心文件 miles/backends/training_utils/loss_hub/logit_processors.py 与 miles/backends/training_utils/loss_hub/math_utils.py，整体拆解为 4 步：

1. 新增 CSR 值对象 RolloutSamplingMask（miles/utils/sampling_mask.py，新增 129 行）以 dataclass(frozen=True) 承载两条扁平整数数组：_ids（int32，[total_support_size]）与 _offsets（int64，[num_response_tokens + 1]），第 t 个 token 的 support 为 ids[offsets[t] : offsets[t + 1]]。 - from_mask_list 从 SGLang output_token_sampling_mask 的嵌套结构（[num_response_tokens][mask_size_t]）构建 CSR；__post_init__ 强制校验 offsets 从 0 开始、终止于 ids 总数、严格递增，即每个 response token 必须有非空 support。 - _select_masks 针对 CP 场景优化：range 连续索引走纯切片快速路径；非连续索引先经 torch.nonzero 找连续 run 边界，再按 run 分段切片后 torch.cat，避免逐 token gather。 - _to_owned_cpu_integer_tensor 在输入为 list 且 dtype 为 int32 时使用 array("i", values) + torch.frombuffer 零拷贝私有化，并整

体复制 Tensor 输入，防止外部修改破坏 CSR 不变式。

2. 响应行全局索引追踪（logit_processors.py，+86/-10） - 原 get_responses 主体更名为 _iter_response_chunks，新增 include_response_indices 开关与第三个出参 response_indices（每个本地行对应的全局 response 位置），并在三种 CP 模式（无 CP、allgather_cp、zigzag）下分别推导：无 CP 直接 range(response_length)；allgather_cp 用全局 logit 区间与本地 chunk 区间求交；zigzag 用 tokens_offset 减去 prompt 长度换算。 - 对外保留 get_responses 二元组包装器，以 include_response_indices=False 转发，保证既有调用方契约不变。 - get_log_probs_and_entropy 新增 rollout_sampling_mask: Sequence[RolloutSamplingMask] | None，调用前用 zip(strict=True) 逐样本校验 mask 长度与 response 长度一致，仅在 mask 非 None 时启用索引追踪。
3. 本地 mask 构建与 log-prob 接线（sampling_mask.py 新增 + math_utils.py 修改） - 新增 miles/backends/training_utils/sampling_mask.py::build_local_sampling_mask：输入为本 rank 的 [local_rows, local_vocab_size] logits 与该样本 CSR mask，先校验 response_indices 为一维整数且与 logits 行数对齐；logits.size(0) == 0 时直接返回空 bool mask 跳过 _select_masks；否则将选出的 token id 搬上 GPU，通过 repeat_interleave 展开 row 索引，过滤落在 [tp_rank * local_vocab_size, (tp_rank + 1) * local_vocab_size) 内的 vocab 分片，最终以 flat_local_indices 填充稠密 bool mask。 - math_utils.py 新增 _apply_sampling_mask(logits, sampling_mask, inplace=...)：mask 为 None 时零开销直通；shape 不匹配抛 ValueError；用 masked_fill(~mask, float("-inf"))屏蔽支持外logit。compute_log_probs、calculate_log_probs_and_entropy、_calculate_log_probs_and_entropy_true_on_policy 三处透传 sampling_mask，分 chunk 时同步 sampling_mask.chunk。 - 关键语义决策：true-on-policy 路径先对本地 logits _apply_sampling_mask（非 inplace，因为后续还要 gather），再全 vocab gather 后 log_softmax；entropy 则 保持全 vocab 归一化——当 mask 存在时重新 gather 一次原始 logits 计算 entropy，与 SGLang scoring contract 对齐。fused CE 路径用 inplace=True（作用于 to(copy=True) 的副本，安全且省一次拷贝）。
4. 测试配套（两个新增测试文件，共 304 行） - tests/fast/utils/test_sampling_mask.py（83 行）：覆盖 CSR 构建、offsets 校验、输入存储所有权（外部篡改不影响）、_select_masks 的连续切片 / 非连续 run 拼接 / 空索引 / 越界拒绝。 - tests/fast/backends/training_utils/test_sampling_mask.py（221 行）：覆盖 build_local_sampling_mask 的 TP 分片选择、行对齐校验、空本地行跳过；_calculate_log_probs_and_entropy_true_on_policy 的「mask 后 log-prob + 全 vocab entropy」语义；get_log_probs_and_entropy 端到端逐 token support 归一化；_iter_response_chunks 在 allgather_cp 与 zigzag 两种 CP 布局下的全局 response 索引推导。全部为 CPU 快速测试。

关键文件：

- miles/utils/sampling_mask.py（模块 采样掩码；类别 source；类型 core-logic；符号 RolloutSamplingMask, post_init, from_mask_list, len）：新增 CSR 值对象 RolloutSamplingMask，是本 PR 的核心数据结构与后续传输协议的基础
- miles/backends/training_utils/sampling_mask.py（模块 掩码构建；类别 source；类型 core-logic；符号 build_local_sampling_mask）：新增 build_local_sampling_mask，把

CSR 采样支持转换为 TP 本地稠密 bool mask，是两条计算路径共用的桥接层

- miles/backends/training_utils/loss_hub/math_utils.py（模块 损失计算；类别 source；类型 core-logic；符号 compute_log_probs, _apply_sampling_mask, calculate_log_probs_and_entropy, _calculate_log_probs_and_entropy_true_on_policy）：修改 log-prob 与 entropy 核心计算：新增 _apply_sampling_mask，并在 true-on-policy 与 fused CE 两条路径接入采样支持
- miles/backends/training_utils/loss_hub/logit_processors.py（模块 响应提取；类别 source；类型 refactor；符号 get_responses, _iter_response_chunks, get_log_probs_and_entropy）：get_responses 重构为 _iter_response_chunks，新增全局响应索引追踪，get_log_probs_and_entropy 接入 rollout_sampling_mask
- tests/fast/backends/training_utils/test_sampling_mask.py（模块 测试覆盖；类别 test；类型 test-coverage；符号 test_build_local_sampling_mask_selects_original_response_rows_and_tp_shard, test_build_local_sampling_mask_rejects_out_of_range_response_index, test_build_local_sampling_mask_rejects_row_misalignment, test_build_local_sampling_mask_skips_selection_for_empty_local_rows）：覆盖 mask 构建、TP 分片、true-on-policy 语义与 CP 索引推导，是核心语义的验证主力
- tests/fast/utils/test_sampling_mask.py（模块 测试覆盖；类别 test；类型 test-coverage；符号 test_rollout_sampling_mask_builds_private_int32_storage, test_rollout_sampling_mask_requires_non_empty_mask, test_rollout_sampling_mask_owns_input_storage, test_rollout_sampling_mask_validates_csr_offsets）：覆盖 RolloutSamplingMask 的 CSR 不变式、输入所有权与 run 拼接逻辑

关键符号：RolloutSamplingMask.from_mask_list, RolloutSamplingMask._select_masks, RolloutSamplingMask.post_init, build_local_sampling_mask, _iter_response_chunks, get_responses, get_log_probs_and_entropy, compute_log_probs, _apply_sampling_mask, calculate_log_probs_and_entropy, _calculate_log_probs_and_entropy_true_on_policy

关键源码片段

miles/utils/sampling_mask.py

新增 CSR 值对象 RolloutSamplingMask，是本 PR 的核心数据结构与后续传输协议的基础

```
@dataclass(frozen=True, eq=False)
class RolloutSamplingMask:
    """单条样本的采样支持：每个 response 位置可被 sampler 采样的 token id 集合。

    以 CSR 形式存储，始终是两段扁平整数数组：``ids`` 为 ``[total_support_size]``
    （所有位置的 support 顺序拼接），``offsets`` 为 ``[num_response_tokens + 1]``，
    第 t 个 token 的 support 为 ``ids[offsets[t] : offsets[t + 1]]``。
    这种形态正是对象存储传输（如 Mooncake）需要的，trainer 侧无需重建嵌套结构。
    """

    ids: InitVar[Sequence[int] | torch.Tensor]
```

```
offsets: InitVar[Sequence[int] | torch.Tensor]
_ids: torch.Tensor = field(init=False, repr=False)
_offsets: torch.Tensor = field(init=False, repr=False)

def __post_init__(self, ids, offsets):
    # 输入统一转为私有 CPU 张量：int32 的 ids + int64 的 offsets，
    # Tensor 输入整体复制，list 输入经 array("i") + frombuffer 零拷贝持有，
    # 避免外部修改破坏 CSR 不变式。
    owned_ids = _to_owned_cpu_integer_tensor(ids, dtype=torch.int32)
    owned_offsets = _to_owned_cpu_integer_tensor(offsets, dtype=torch.long)
    if owned_offsets.numel() == 0 or owned_offsets[0] != 0 or owned_offsets[-1] != owned_ids.
    numel():
        raise ValueError("sampling-mask offsets must start at zero and end at the flattened id
        count")
    if torch.any(owned_offsets[1:] <= owned_offsets[:-1]):
        # 每个 response token 都必须有非空 support，否则 log-prob 无法归一化
        raise ValueError(
            "sampling-mask offsets must be strictly increasing: "
            "every response token needs a non-empty sampling mask"
        )
    object.__setattr__(self, "_ids", owned_ids)
    object.__setattr__(self, "_offsets", owned_offsets)

@classmethod
def from_mask_list(cls, mask_list):
    """从 SGLang ``output_token_sampling_mask`` 的嵌套结构构建 CSR。"""
    ids = []
    offsets = [0]
    for mask in mask_list:
        ids.extend(mask)
        offsets.append(len(ids))
    return cls(ids=ids, offsets=offsets)

def _select_masks(self, token_indices):
    """按全局 response 位置选取 support，返回拼接后的 ``ids`` 与各位置长度。

    CP 各 rank 持有的 response 行通常构成少数连续 run，这里对 ``range``
    走切片快速路径；非连续索引则先找 run 边界再分段拼接，尽量少复制。
    """
    if isinstance(token_indices, range) and token_indices.step == 1:
        if len(token_indices) == 0:
            return self._ids.new_empty(0), self._offsets.new_empty(0)
        if token_indices.start < 0 or token_indices.stop > len(self):
            raise ValueError(f"response indices must be in [0, {len(self)})")
        start, stop = token_indices.start, token_indices.stop
        lengths = self._offsets[start + 1 : stop + 1] - self._offsets[start:stop]
        return self._ids[self._offsets[start] : self._offsets[stop]], lengths

    indices = _to_cpu_integer_tensor(token_indices).to(torch.long)
```

```
        if torch.any(indices < 0) or torch.any(indices >= len(self)):
            raise ValueError(f"response indices must be in [0, {len(self)})")
        lengths = self._offsets[indices + 1] - self._offsets[indices]
        if indices.numel() == 0:
            return self._ids.new_empty(0), lengths
        # 找出连续 run 的起点，每个 run 只做一次切片，避免逐行 gather
        run_starts = [0]
        run_starts.extend((torch.nonzero(indices[1:] != indices[:-1] + 1).flatten() + 1).tolist())
        run_starts.append(indices.numel())
        parts = [
            self._ids[self._offsets[indices[start]] : self._offsets[indices[end - 1] + 1]]
            for start, end in zip(run_starts[:-1], run_starts[1:], strict=True)
        ]
        return (parts[0] if len(parts) == 1 else torch.cat(parts)), lengths
```

miles/backends/training_utils/sampling_mask.py

新增 build_local_sampling_mask，把 CSR 采样支持转换为 TP 本地稠密 bool mask，是两条计算路径共用的桥接层

```
def build_local_sampling_mask(
    logits: torch.Tensor,
    sampling_mask: RolloutSamplingMask,
    response_indices: Sequence[int] | torch.Tensor,
    *,
    tp_rank: int,
) -> torch.Tensor:
    """构建 log-prob 原语消费的本地 vocab 稠密布尔 mask。

    Args:
        logits: ``[local_rows, local_vocab_size]`` 本 rank 持有的 response 行
            logits（TP vocab 分片 + CP 行子集）。
        sampling_mask: 该样本完整的采样支持。
        response_indices: ``[local_rows]`` 每行对应的全局 response 位置。
        tp_rank: 本 rank 在 TP 组内的序号。

    Returns:
        与 ``logits`` 同形状的布尔 mask，True 表示该位置在采样支持内。
    """
    # 张量形式的索引必须是一维整数，先做防御性校验
    if isinstance(response_indices, torch.Tensor) and (
        response_indices.ndim != 1
        or response_indices.dtype == torch.bool
        or torch.is_floating_point(response_indices)
        or torch.is_complex(response_indices)
    ):
        raise ValueError("sampling-mask ids, offsets, and response indices must be one-
        dimensional integers")
    if len(response_indices) != logits.size(0):
        raise ValueError(
```

```
            f"sampling-mask rows must align with logits: indices={len(response_indices)}, logits=
            {logits.size(0)}"
        )

    # 本 rank 没有 response 行时直接返回空 mask，避免无意义的 gather
    if logits.size(0) == 0:
        return torch.zeros(logits.numel(), dtype=torch.bool, device=logits.device).view_as(logits)

    # CP response 行通常是少数连续 run，CSR gather 在 CPU 上做少量切片，
    # 再一次性搬到 GPU 展开成稠密 mask。
    selected_ids, lengths = sampling_mask._select_masks(response_indices)
    selected_ids = selected_ids.to(logits.device)
    row_indices = torch.repeat_interleave(
        torch.arange(len(response_indices), dtype=torch.long, device=logits.device),
        lengths.to(device=logits.device, dtype=torch.long),
    )
    local_vocab_size = logits.size(-1)
    vocab_start = tp_rank * local_vocab_size
    # 只保留落在本 rank TP vocab 分片内的 token id
    is_local = (selected_ids >= vocab_start) & (selected_ids < vocab_start + local_vocab_size)
    flat_local_indices = row_indices[is_local] * local_vocab_size + selected_ids[is_local].to(torch.
    long) - vocab_start
    mask = torch.zeros(logits.numel(), dtype=torch.bool, device=logits.device)
    mask[flat_local_indices] = True
    return mask.view_as(logits)
```

miles/backends/training_utils/loss_hub/math_utils.py

修改 log-prob 与 entropy 核心计算：新增 _apply_sampling_mask，并在 true-on-policy 与 fused CE 两条路径接入采样支持

```
def _apply_sampling_mask(
    logits: torch.Tensor,
    sampling_mask: torch.Tensor | None,
    *,
    inplace: bool = False,
) -> torch.Tensor:
    # mask 为 None 时零开销直通，保证无采样支持路径的行为完全不变
    if sampling_mask is None:
        return logits
    if sampling_mask.shape != logits.shape:
        raise ValueError(f"sampling mask shape {sampling_mask.shape} != logits shape {logits.
        shape}")
    # 用 -inf 屏蔽支持外的 logit，随后 log_softmax 等价于仅对支持内归一化
    if inplace:
        return logits.masked_fill_(~sampling_mask, float("-inf"))
    return logits.masked_fill(~sampling_mask, float("-inf"))


# true-on-policy 分支内的关键语义（math_utils.py::_calculate_log_probs_and_entropy_true_on_
```

```
policy）：
# log-prob 使用被 mask 后的本地 logits 做全 vocab gather 再 log_softmax；
# entropy 保持全 vocab 分布（与 SGLang scoring contract 一致），
# 因此 mask 存在时需要基于原始 logits 重新 gather 一次。
log_prob_logits = _apply_sampling_mask(logits, sampling_mask)
full_logits = _gather_true_on_policy_full_logits(log_prob_logits, tp_group, vocab_size=vocab_size)
log_probs_full = torch.log_softmax(full_logits, dim=-1)
...
if with_entropy:
    if sampling_mask is None:
        entropy_log_probs = log_probs_full
    else:
        entropy_logits = _gather_true_on_policy_full_logits(logits, tp_group, vocab_size=vocab_
        size)
        entropy_log_probs = torch.log_softmax(entropy_logits, dim=-1)
    if not entropy_requires_grad:
        entropy_log_probs = entropy_log_probs.detach()
    probs = entropy_log_probs.exp()
    entropy = -(probs * entropy_log_probs).sum(dim=-1)
```

评论区精华

核心讨论集中在数据表示形式的取舍上：

- CSR 扁平结构 vs 嵌套 list[list[int]]：guapisolo 连续两次追问「Why didn't directly use list[list[int]] as data format?」以及「why you use csr (flatten tensor + offset) instead of direct storage」。nanjiangwill 回应：「for logical model the nested structure is better but for data transport the current csr is better」，点出逻辑可读性与对象存储传输形态之间的权衡。最终由 guapisolo 自己提交一个小 PR（nanjiangwill/miles#1）把原始裸嵌套输入包装成 per-sample 值对象 RolloutSamplingMask，再经「remove redundant sampling-mask guards」提交收尾，删除冗余守卫并依赖 zip(strict=True) 做批次对齐。
- Mooncake 传输可行性：guapisolo 提到「Codex said mooncake can support this by some tiny modifications」，侧面印证 CSR 扁平形态正是为了对接后续 #2595 的 Mooncake/ 对象存储传输。
- CI 失败归属：yueming-yuan 要求排查 CI failure，guapisolo 判断「seems like introduced by rdt PR instead of this one」，即由 RDT PR（#1313）引入，计划另开 PR 修复。
  - 为什么用 CSR 扁平结构而非 list[list[int]] 嵌套结构 (design): guapisolo 亲自提交小 PR（nanjiangwill/miles#1）把裸嵌套输入包装成 per-sample 值对象 RolloutSamplingMask，随后 merged，并进一步删除了冗余守卫。
  - Mooncake 对象存储对采样支持传输的支持度 (question): 未在本 PR 展开，作为后续 #2595 持久化与传输落地的依据。
  - CI 失败是否由本 PR 引入 (other): 定位为外部 PR 引入，本 PR 继续合并流程。

风险与影响

- 风险：
  1. 核心响应提取路径重构：get_responses 是 rollout 与训练共用的 log-prob 提取入口，本次重构为 _iter_response_chunks 包装器形态虽有专门测试覆盖，但任何既有调用方若直接引用内部实现细节仍可能受影响；response_indices 在 zigzag CP 模式下的推导依赖 tokens_offset - prompt_length 换算，off-by-one 仅在启用 mask 时暴露，风险被无生产调用方的现状掩盖。
  2. _apply_sampling_mask 的 inplace 副作用：masked_fill_ 会原地修改 logits；当前 true-on-policy 走非 inplace、fused CE 路径操作的是 to(copy=True) 副本，均安全，但这一设计对后续新调用方是隐患——若直接传入共享 logits 且 inplace=True 会污染上游张量。
  3. entropy 重复 gather 的通信开销：true-on-policy + mask 存在时，entropy 需要第二次 _gather_true_on_policy_full_logits，多一次 TP all-gather；在超大 vocab 模型（如 40 万词表）上该开销不可忽略，需要后续 #2596 实测。
  4. 平台相关的 array("i") 宽度：_to_owned_cpu_integer_tensor 假设 C int 为 32 位并用 torch.frombuffer 以 int32 解释，在主流平台成立，但跨平台移植性未显式防护。
  5. 端到端覆盖缺口：当前全部为 CPU 快速测试，缺少真实多节点 CP+TP 组合下的 log-prob 一致性验证，该验证依赖系列后续 PR 的 e2e 场景。 - 影响：用户与行为：本 PR 无生产调用方传入采样 mask，rollout 与训练输出完全不变，属于零行为影响的基础设施合入。系统层面：为「rollout 采样支持（top-p 等）→ 持久化传输 → actor 同支持打分」这条 on-policy 正确性链路补齐了第一块拼图；RolloutSamplingMask 的 CSR 形态直接服务对象存储（Mooncake）传输需求。团队层面：PR body 明确标注这是三 PR 系列的第 1 步，后续 #2595 负责持久化与传输、#2596 负责原子启用 SGLang capture 与 actor replay；reviewer 需要结合系列整体理解设计意图。涉及 PPO loss 计算、响应提取、TP/CP 并行边界多个模块，影响面广但受控。 - 风险标记：核心 loss 路径重构，无生产调用方覆盖，inplace 修改副作用风险，entropy 二次 gather 通信开销，CP 索引推导易错，端到端验证依赖后续 PR

关联脉络

- PR #2595 persists and transports rollout sampling support（系列第 2 步）：PR body 明确的系列规划：本 PR 提供采样支持 log-prob 原语，2595 负责持久化与传输 rollout 采样支持。
- PR #2596 atomically enables bounded SGLang capture and actor replay（系列第 3 步）：PR body 明确的系列规划：2596 原子启用 SGLang capture 与 actor replay，是本 PR 原语的生产消费方。
- PR #1313 RDT weight sync: GPU->GPU zero copy transfer through SGLang Ray actor backend: 关联 issue 评论确认本 PR CI 失败由 RDT PR 引入，且 CSR 采样支持后续将经由相似对象存储 /Mooncake 通道传输。