

PR #2825 完整报告

radixark/miles

fix: chunk true-on-policy logprob computation

合并时间: 2026-09-10 09:41

原文链接: <http://prhub.com.cn/radixark/miles/pull/2825>

执行摘要

- 一句话: 修复 true-on-policy logprob 计算路径未按 chunk 处理响应行导致 OOM 的问题。
- 推荐动作: 该 PR 是一个高质量的 bugfix, 值得精读。其核心实现(响应行分块)和配套的全面测试(正确性、数值稳定性、内存)都展示了严谨的工程实践。建议关注:
 1. `_calculate_log_probs_and_entropy_true_on_policy` 中分块循环的实现, 特别是熵计算部分对 `entropy_requires_grad` 的处理和原地操作优化。
 2. GPU 测试如何设置 TP2 环境、构建参考实现以及验证内存峰值, 这对编写类似测试有借鉴意义。
 3. PR 虽然修改文件少, 但变更涉及核心计算路径, 且测试覆盖深入, 是 bugfix 的典范。

功能与动机

动机源于一个实际的内存溢出问题: 当使用 true-on-policy 模式且配置了正数 `log_probs_chunk_size` 时, 8192 token 的响应会因全量计算熵而 OOM。PR body 明确指出: 'An 8192-token true-on-policy response can OOM while computing entropy because all response rows are processed together.' 根因是 `calculate_log_probs_and_entropy` 在调用 true-on-policy 路径时遗漏了 `chunk_size` 参数, 且原生函数内部按全量行处理。

实现拆解

1. 入口参数透传: 在 `miles/backends/training_utils/loss_hub/math_utils.py` 的 `calculate_log_probs_and_entropy` 函数中, 当 `true_on_policy=True` 时, 将调用者传入的 `chunk_size` 参数显式传递给 `_calculate_log_probs_and_entropy_true_on_policy`, 修复了参数遗漏。
2. 核心逻辑分块重构: 在 `_calculate_log_probs_and_entropy_true_on_policy` 内部, 新增 `chunk_size` 参数。在完成 TP 词汇量 gather 后, 不再对整个 `full_logits` 张量执行一次性的 `log_softmax` 和熵计算, 而是使用 `split(chunk_size, dim=0)` 将 logits、tokens 和熵所需的 logits (可能因 sampling mask 而不同) 按行分块。然后在一个循环中对每个块独立计算 `log_probs` 和 `entropy`, 最后使用 `torch.cat` 拼接结果。这确保了内存占用与 `chunk_size` 而非总行数成正比。
3. 熵计算的内存优化: 在熵计算循环中, 当 `entropy_requires_grad=False` 时, 使用原地操作 `probs.mul_(entropy_log_probs)` 替代 `probs * entropy_log_probs`, 以减少中间张量分配。

4. 全面测试覆盖：

- 快速单元测试(tests/fast/utils/test_true_on_policy_logprobs.py)：新增 test_true_on_policy_logprobs_honor_chunk_size 通过 monkeypatch 验证分块大小；新增 test_true_on_policy_chunked_entropy_preserves_gradients 验证分块后梯度与全量计算一致。
- GPU 集成测试(tests/fast-gpu/test_true_on_policy_logprob_chunking.py)：新增全面的 TP2 GPU 测试，包含：
 - 正确性测试(_assert_correctness_case)：在 TP2 设置下，验证带 / 不带 sampling mask 的分块计算与独立参考实现（全量 log_softmax）的结果一致，并验证梯度正确性。
 - 低精度数值稳定性测试(_assert_low_precision_reductions)：在 BF16/FP16 精度下，验证分块与非分块路径、以及与 FP64 参考实现的结果差异在容许范围内。
 - 内存峰值测试：模拟生产环境（8192 行，真实词表大小），测量分块（chunk_size=1024）与非分块（-1）的 CUDA 峰值内存差异，要求至少节省 1 GiB。

关键文件：

- miles/backends/training_utils/loss_hub/math_utils.py（模块 损失计算；类别 source；类型 core-logic；符号 calculate_log_probs_and_entropy, _calculate_log_probs_and_entropy_true_on_policy）：核心源码文件，包含被修改的 calculate_log_probs_and_entropy 和 _calculate_log_probs_and_entropy_true_on_policy 函数，是修复 OOM 问题的关键。
- tests/fast-gpu/test_true_on_policy_logprob_chunking.py（模块 GPU 测试；类别 test；类型 test-coverage；符号 _correctness_worker, _assert_low_precision_reductions, _run_memory_forward）：新增的全面 GPU 集成测试，是验证修复在真实 TP2 环境、低精度和内存峰值下有效的关键。
- tests/fast/utils/test_true_on_policy_logprobs.py（模块 快速单元测试；类别 test；类型 test-coverage；符号 test_true_on_policy_logprobs_honor_chunk_size, test_true_on_policy_chunked_entropy_preserves_gradients）：修改了现有测试文件，新增了快速单元测试以验证分块行为和梯度保持，是测试覆盖的重要补充。

关键符号：calculate_log_probs_and_entropy, _calculate_log_probs_and_entropy_true_on_policy

关键源码片段

miles/backends/training_utils/loss_hub/math_utils.py

核心源码文件，包含被修改的 calculate_log_probs_and_entropy 和 _calculate_log_probs_and_entropy_true_on_policy 函数，是修复 OOM 问题的关键。

```
# miles/backends/training_utils/loss_hub/math_utils.py (修改后)
def _calculate_log_probs_and_entropy_true_on_policy(
    logits: torch.Tensor,
    tokens: torch.Tensor,
    tp_group: dist.ProcessGroup | None,
    with_entropy: bool = False,
```

```

entropy_requires_grad: bool = True,
chunk_size: int = -1, # 新增参数, 控制每次处理的行数
vocab_size: int | None = None,
sampling_mask: torch.Tensor | None = None,
) -> tuple[torch.Tensor, torch.Tensor | None]:
    # ... ( 文档字符串和边界条件检查省略 )

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)
_maybe_dump_top_logprob_backward("full_logits", full_logits)

# 准备熵计算所需的 logits ( 可能与 log_prob_logits 不同 )
entropy_logits = None
if with_entropy:
    entropy_logits = (
        full_logits
        if sampling_mask is None
        else _gather_true_on_policy_full_logits(logits, tp_group, vocab_size=vocab_size)
    )

# 根据 chunk_size 决定是否分块
if chunk_size > 0:
    # 按响应行 (dim=0) 分块
    full_logits_chunks = full_logits.split(chunk_size, dim=0)
    tokens_chunks = tokens.split(chunk_size, dim=0)
    # 熵 logits 可能共享同一组块或需要独立分块
    entropy_logits_chunks = (
        full_logits_chunks
        if entropy_logits is full_logits # 身份比较, 避免重复 gather
        else entropy_logits.split(chunk_size, dim=0) if entropy_logits is not None else ()
    )
else:
    # 不分块, 使用单块元组包裹
    full_logits_chunks = (full_logits,)
    tokens_chunks = (tokens,)
    entropy_logits_chunks = (entropy_logits,) if entropy_logits is not None else ()

log_prob_chunks = []
entropy_chunks = []
# 逐块计算 log_softmax 和熵
for chunk_index, (full_logits_chunk, tokens_chunk) in enumerate(
    zip(full_logits_chunks, tokens_chunks, strict=True)
):
    log_probs_full = torch.log_softmax(full_logits_chunk, dim=-1)
    _maybe_dump_top_logprob_backward("log_probs_full", log_probs_full)
    log_prob_chunks.append(torch.gather(log_probs_full, dim=-1, index=tokens_chunk.
        unsqueeze(-1)).squeeze(-1))

```

```

if with_entropy:
    entropy_logits_chunk = entropy_logits_chunks[chunk_index]
    entropy_log_probs = (
        log_probs_full # 复用已计算的 log_probs (当 entropy_logits 是 full_logits 时)
        if entropy_logits_chunk is full_logits_chunk
        else torch.log_softmax(entropy_logits_chunk, dim=-1)
    )
    if not entropy_requires_grad:
        entropy_log_probs = entropy_log_probs.detach()
    probs = entropy_log_probs.exp()
    # 当不需要梯度时, 使用原地操作以节省内存
    if entropy_requires_grad:
        entropy_chunks.append(-(probs * entropy_log_probs).sum(dim=-1))
    else:
        entropy_chunks.append(-probs.mul_(entropy_log_probs).sum(dim=-1))

# 拼接各块结果
log_prob = log_prob_chunks[0] if len(log_prob_chunks) == 1 else torch.cat(log_prob_chunks,
dim=0)
_maybe_dump_top_logprob_backward("log_prob", log_prob)
entropy = None
if with_entropy:
    entropy = entropy_chunks[0] if len(entropy_chunks) == 1 else torch.cat(entropy_chunks,
dim=0)

return log_prob, entropy

```

评论区精华

Review 讨论较少, 仅由 [claude\[bot\]](#) 提示因 fork 无法自动审查。主要审核者 [adrenaline21](#), [guapisolo](#), [yueming-yuan](#) 均直接 APPROVED。[guapisolo](#) 在审批时提到 'checked and add ci', 表明其亲自检查了代码并补充了 CI 测试。PR 的测试提交由 [guapisolo](#) 完成, 专注于验证分块在 TP2 GPU 上的正确性、数值稳定性和内存收益。

- 暂无高价值评论线程

风险与影响

- 风险:

1. 正确性风险 (低): 核心变更在 `_calculate_log_probs_and_entropy_true_on_policy` 中, 逻辑清晰。测试覆盖全面, 包括了 TP2、低精度、采样掩码、梯度正确性和内存峰值, 降低了回归风险。需确保 `entropy_logits is full_logits` 的身份比较逻辑 (用于避免重复 gather) 在所有路径下正确。
2. 性能风险 (低): 引入分块循环会增加少量 Python 开销, 但 PR 的目的是避免 OOM, 这是关键的内存 - 性能权衡。内存测试已验证节省效果。
3. 兼容性风险 (低): 变更限于内部函数签名和实现, 不改变公共 API 行为。`chunk_size` 默认值 -1 保持了向后兼容性 (非正数时全量处理)。- 影响: 用户影响: 修复了

true-on-policy 模式下处理长序列（如 8192 token）时可能出现的 OOM 崩溃，提升了训练稳定性。配置了 `log_probs_chunk_size` 的用户将直接受益。系统影响：增强了loss计算路径的内存可预测性和鲁棒性，特别是对于长上下文训练场景。测试基础设施得到加强，新增了针对 TP 和低精度分块计算的专用测试。团队影响：代码质量高，测试详尽，可作为处理类似内存敏感计算分块的参考。

- 风险标记：核心路径变更，内存优化，数值精度

关联脉络

- 暂无明显关联 PR