PR #6593 完整报告

verl-project/verl

[fsdp] feat: chunked gather-logsumexp for top-K loss to avoid OOM at long context

合并时间：2026-06-10 11:24

原文链接：http://prhub.com.cn/verl-project/verl/pull/6593

执行摘要

- 一句话：分块 gather-logsumexp 避免长上下文 OOM
- 推荐动作：对于从事长上下文蒸馏或大型词表模型训练的开发者，本 PR 值得精读，它展示了在不牺牲数值精度前提下，通过数学恒等变形和分块计算优雅解决 OOM 问题的模式。对于 verl 核心维护者，建议关注后续 PR _forward_skip_lm_head，以完全消除 [B,T,V] 张量。其他人可快速了解设计权衡。

功能与动机

在 OPD 蒸馏中，学生端损失计算需要同时通过 gather 取 top-K 教师 ID 和策略梯度 ID，导致 student_log_probs 必须保留在内存中。对于 Qwen 等词表大小 V=152064 的模型，使用 remove-padding 后 N（有效 token 数）可达 60K 至 96K，单张 [N, V] 的 bf16 张量占用 18~28 GB 显存。原始实现中的 F.log_softmax(student_logits) 再次实例化相同大小的张量，使得总激活内存翻倍，在单卡 H20/H200 上频繁 OOM。PR body 指出："Without this fix, the only workarounds are: reduce context length (defeats the purpose), add nodes for SPREAD topology (expensive, not always available), or disable colocated rollout (reduces throughput)."

实现拆解

1. 新增 _chunked_topk_log_probs 函数：在 verl/trainer/distillation/fsdp/losses.py 中实现该核心函数。它将 logits 和 topk_ids 展平为 [N, V] 和 [N, K]，然后按 chunk_size（默认 4096）沿 N 维循环。每个 chunk 内先转 fp32 计算 logsumexp，再通过 gather 获取对应位置的 logit，减去 log_z 后转回原始 dtype。预分配输出张量并通过 in-place 切片赋值（out[s:e] = ...）写入，PyTorch autograd 支持此操作（CopySlices），梯度可正确回传。
2. 修改 compute_forward_kl_topk：在原有的 F.log_softmax + gather 路径旁新增条件分支。当 loss_config.use_chunked_topk 为 True 时，调用上述分块函数计算 student_topk_log_probs，同时 student_topk_ids 改为直接从 student_logits 做 torch.topk（因 log_softmax 单调）。默认路径保持不变。
3. 配置项扩展：在 verl/workers/config/distillation.py 的 DistillationLossConfig 中新增 use_chunked_topk: bool = False 和 chunked_topk_chunk_size: int = 4096，并附详细注释说明性能权衡。
4. 测试配套：新增 tests/workers/test_chunked_topk_log_probs_on_cpu.py，包含 7 个测试用例：数值等价性（fp32 多种参数）、低精度 GPU 等价性（bf16/fp16）、chunk_size

不变量、梯度正确性、小 chunk_size、空输入、slice 赋值梯度流验证。所有测试均通过。

关键文件：

- verl/trainer/distillation/fsdp/losses.py（模块 蒸馏损失；类别 source；类型 core-logic；符号 _chunked_topk_log_probs, compute_forward_kl_topk）：核心变更文件，新增 _chunked_topk_log_probs 函数并修改 compute_forward_kl_topk 以有条件使用分块路径。
- verl/workers/config/distillation.py（模块 配置；类别 source；类型 configuration；符号 DistillationLossConfig）：新增配置项 use_chunked_topk 和 chunked_topk_chunk_size，提供 opt-in 控制。
- tests/workers/test_chunked_topk_log_probs_on_cpu.py（模块 测试；类别 test；类型 test-coverage；符号 _reference_topk_log_probs, test_numerical_equivalence, test_numerical_equivalence_low_precision_on_gpu, test_chunk_size_invariance）：新增的测试文件，完整覆盖分块逻辑的数值正确性和梯度正确性。

关键符号：_chunked_topk_log_probs, compute_forward_kl_topk, _reference_topk_log_probs

关键源码片段

verl/trainer/distillation/fsdp/losses.py

核心变更文件，新增 _chunked_topk_log_probs 函数并修改 compute_forward_kl_topk 以有条件使用分块路径。

```
def _chunked_topk_log_probs(
    logits: torch.Tensor,
    topk_ids: torch.Tensor,
    chunk_size: int = 4096,
) -> torch.Tensor:
    """Compute log_softmax(logits).gather(topk_ids) without materializing [B, T, V].

    Uses the identity:
        log_softmax(x).gather(idx) == x.gather(idx) - logsumexp(x, keepdim=True)
    Streams the reduction in chunks of `chunk_size` tokens along (B*T) with fp32
    logsumexp for numerical stability.
    """
    B, T, V = logits.shape
    K = topk_ids.shape[-1]
    flat_logits = logits.reshape(-1, V) # [N, V], N = B*T
    flat_topk = topk_ids.reshape(-1, K) # [N, K]
    N = flat_logits.shape[0]

    # 边界情况：空输入（例如全 pad 的 micro-batch）
    if N == 0:
        return torch.empty((B, T, K), dtype=logits.dtype, device=logits.device)

    # 预分配输出，后续通过 `CopySlices` 支持 autograd
    out = torch.empty((N, K), dtype=logits.dtype, device=logits.device)
```

```
    for s in range(0, N, chunk_size):
        e = min(s + chunk_size, N)
        # chunk 内用 fp32 计算，保证数值稳定性
        chunk_logits_fp32 = flat_logits[s:e].float()
        log_z = torch.logsumexp(chunk_logits_fp32, dim=-1, keepdim=True) # [c, 1]
        chunk_topk_logits = torch.gather(chunk_logits_fp32, dim=-1, index=flat_topk[s:e])
        out[s:e] = (chunk_topk_logits - log_z).to(logits.dtype)
    return out.reshape(B, T, K)


def compute_forward_kl_topk(...):
    ...
    # 选择路径：默认使用 F.log_softmax；opt-in 使用分块路径
    loss_config: DistillationLossConfig = config.distillation_loss
    if loss_config.use_chunked_topk:
        # log_softmax 单调，topk(logits) == topk(log_softmax(logits))
        student_topk_ids = torch.topk(student_logits, k=teacher_topk_ids.shape[-1], dim=-1).
        indices
        student_topk_log_probs = _chunked_topk_log_probs(
            student_logits,
            teacher_topk_ids,
            chunk_size=loss_config.chunked_topk_chunk_size,
        )
    else:
        student_log_probs = F.log_softmax(student_logits, dim=-1)
        student_topk_ids = torch.topk(student_log_probs, k=..., dim=-1).indices
        student_topk_log_probs = torch.gather(student_log_probs, dim=-1, index=teacher_topk_ids)
    ...
```

tests/workers/test_chunked_topk_log_probs_on_cpu.py

新增的测试文件，完整覆盖分块逻辑的数值正确性和梯度正确性。

```
def _reference_topk_log_probs(logits, topk_ids):
    # 参考实现：标准 log_softmax + gather（即 patch 前的代码路径）
    log_probs = F.log_softmax(logits, dim=-1)
    return torch.gather(log_probs, dim=-1, index=topk_ids)


@pytest.mark.parametrize("B,T,V,K,dtype,atol", [
    # 小规模：紧公差验证正确性
    (2, 16, 128, 4, torch.float32, 5e-6),
    # 真实词表大小 Qwen V=152064，中等序列
    (2, 32, 152064, 8, torch.float32, 5e-6),
    # 更大的 top-K（OPD 常用 K=64）
    (2, 32, 152064, 64, torch.float32, 5e-6),
])
def test_numerical_equivalence(B, T, V, K, dtype, atol):
    """验证分块输出与参考实现一致（fp32，CPU）"""
    torch.manual_seed(42)
    logits = torch.randn(B, T, V, dtype=dtype)
```

```
    topk_ids = torch.randint(0, V, (B, T, K))
    ref = _reference_topk_log_probs(logits, topk_ids)
    out = _chunked_topk_log_probs(logits, topk_ids, chunk_size=4096)
    max_diff = (ref - out).abs().max().item()
    assert max_diff <= atol, f"max |ref - out| = {max_diff:.2e}"

@pytest.mark.skipif(not torch.cuda.is_available(), reason="GPU-only")
@pytest.mark.parametrize("dtype,atol", [
    (torch.bfloat16, 5e-3),
    (torch.float16, 1e-3),
])
def test_numerical_equivalence_low_precision_on_gpu(dtype, atol):
    """验证低精度 GPU 上分块与参考一致（GPU fused log_softmax 内部也使用 fp32）"""
    ...

@pytest.mark.parametrize("chunk_size", [1, 64, 256, 1024, 4096, 16384, 99999999])
def test_chunk_size_invariance(chunk_size):
    """验证结果不依赖 chunk_size（仅影响内存/速度，不影响数值）"""
    ...

def test_gradient_correctness():
    """验证分块路径的梯度与参考路径一致（fp32）"""
    ...
```

评论区精华

- Autograd 正确性：gemini-code-assist 初审指出 torch.empty + 切片赋值会破坏梯度流。但作者验证并指出 Tensor.__setitem__ 可微分（CopySlices），并通过测试 test_gradient_flows_through_slice_assignment 和 test_gradient_correctness 证明。最终该担忧被澄清。（讨论对象：gemini-code-assist, kekelllllll）
- 注释简化：Luosuu 建议精简过长注释以提高可读性，作者在后续提交中删除了 autograd 论证段落，将 dispatch 注释从 22 行压缩到 5 行。（讨论对象：Luosuu, kekelllllll）
- 优化范围澄清：Luosuu 质疑 logits 本身仍然很大。作者解释本 PR 解决了两个 [B,T,V] 峰值中的第二个（log_softmax 缓冲区），第一个（student_logits）需要后续 LM-head skip 重构，作为未来工作。（讨论对象：Luosuu, kekelllllll）
- Autograd 正确性：in-place 切片赋值是否破坏梯度流 (correctness): 确认 in-place 赋值在 autograd 中正确，测试已验证，代码保持原样。
- 注释精简：inline 注释过于冗长 (style): 已按照 review 精简注释，提交 483a3072。
- 优化范围：student_logits 本身仍是大张量 (design): 明确 PR 边界，后续工作追跟踪。

风险与影响

- 风险：
  - 核心路径变更：修改了 compute_forward_kl_topk——这是 OPD 蒸馏损失的核心路径。虽然新增代码通过配置开关隔离，但分支逻辑存在引入新 bug 的可能。数值等价性测试和梯度正确性测试覆盖了多种参数，风险可控。

  - 性能退化风险：分块路径在短上下文时速度慢约 6 倍（如 14K tokens）。通过默认关闭和详尽的 benchmark 文档，用户可做出知情选择。但若用户在长上下文错误地开启（实际仍会 OOM），可能反而浪费调试时间。
  - 配置兼容性：新字段默认值保证向后兼容，旧配置文件无需改动。若用户在未来更新配置文件并错误设置，也不会影响运行（仅日志提示）。
- 影响：
  - 用户影响：长上下文蒸馏用户（>=64K tokens）可通过设置 use_chunked_topk: true 避免 OOM，但需接受额外的前向时间开销（约 3-6 倍）；短上下文用户无任何感知。
  - 系统影响：消除了一个显存瓶颈，使单卡能支持更长序列的蒸馏，减少了对多节点扩展的依赖。
  - 团队影响：新增了测试文件和配置项，维护成本较低。后续可能的 LM-head skip 重构将建立在此 PR 的基础上。
  - 风险标记：核心路径变更（损失计算），性能退化风险（opt-in 缓解），向后兼容性保障

关联脉络

- PR #6638 [fsdp] feat: add Qwen3.5-4B on-policy distillation FSDP script: 同一蒸馏功能线（on-policy distillation），该 PR 添加了蒸馏示例脚本，本 PR 优化了蒸馏损失函数的内存效率。