

PR #54901 完整报告

vllm-project/vllm

[Perf][Model Runner V2] Compact sampling masks on GPU instead of unpacking the full-vocab bitmask on CPU

合并时间: 2026-09-04 10:03

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/54901>

执行摘要

- 一句话: 将采样掩码紧凑操作移至 GPU, 消除 CPU 端的 $O(\text{num_reqs} \times \text{vocab})$ 解包开销, 恢复高并发下的解码吞吐量。
- 推荐动作: 这个 PR 值得精读, 因为它解决了一个严重的生产性能问题, 且设计决策 (如 GPU 紧凑、宽度上限、回退机制) 展示了在 GPU-CPU 协同优化中的良好实践。核心 Triton 内核实现和测试策略 (覆盖各种支持大小) 也值得学习。

功能与动机

PR body 明确指出: `--return-sampling-mask` (#49577) 导致解码吞吐量在高负载下崩溃, PrimeIntellect 报告了约 2 倍的 RL 步时间和更高的引擎不平衡 (PrimeIntellect-ai/prime-rl#3431)。根本原因是 `SamplingMaskTensors.tolist()` 在 worker 的异步输出线程中, 对每个步骤的 `[num_reqs, vocab]` 位掩码运行 `np.unpackbits + np.nonzero`, 这是一个与实际保留的 token 数量无关的 $O(\text{num_reqs} \times \text{vocab})$ CPU 开销。

实现拆解

1. 核心 GPU 内核替换: 在 `vllm/v1/worker/gpu/sample/output.py` 中, 用新的 `_compact_sampling_mask_kernel` 替换旧的 `_pack_sampling_mask_kernel`。新内核在一次遍历中同时生成每个请求的有限 logit id 的紧凑 int32 数组 (按升序写入 `[num_reqs, max_num_kept]` 缓冲区) 和位掩码, 并计算总数。通过 `tl.cumsum` 实现流式紧凑。
2. `SamplingMaskTensors` 数据结构扩展: 在 `SamplingMaskTensors` 命名元组中增加 `token_ids` 字段 (`[num_reqs, max_num_kept]`), `tolist()` 方法改为优先使用紧凑数组, 仅对支持大小超过 `max_num_kept` 的行 (即 top-k 边界平局或巨大 top_k) 回退到位掩码。引入 `MAX_COMPACT_SUPPORT = 2048` 作为缓冲区宽度上限, 防止内存爆炸。
3. `SamplingMaskLists` 简化: 在 `vllm/v1/outputs.py` 中, 将 `SamplingMaskLists.offsets` 改为可选 (`np.ndarray | None`), `slice_request` 方法断言 `num_positions == 1` 并直接按请求索引切片, 移除已不使用的 `merge` 静态方法。这简化了 Model Runner V2 的单步单位位置语义。
4. 前端响应构建调整: 在 `vllm/v1/engine/output_processor.py` 中, `_new_completion_output` 方法直接从每步的 `sampling_mask_chunks` 构建响应列表, 而无需先合并成 CSR 再转换, 减少了冗余操作。

5. 配置验证与兼容性：在 `vllm/config/vllm.py` 的 `_validate_batch_sharded_sampling` 中新增检查，拒绝 `return_sampling_mask` 与 `enable_batch_sharded_sampling` 的组合，因为 `gather_sampler_output()` 不转发采样掩码。在 `vllm/v1/worker/gpu/sample/sampler.py` 中，从当前批次的 `top_k` 状态计算 `max_num_kept` 并传递给 `from_logits`。
6. 测试全面更新：在 `tests/v1/test_outputs.py` 中，用参数化测试 `test_sampling_mask_tensors_match_finite_support` 替换旧的布局测试，覆盖各种支持大小（包括空、1、接近紧凑宽度、超过上限、整个词表）和边界平局情况。新增 `test_sampling_mask_preserves_top_k_boundary_ties` 验证 top-k 平局处理。在 `tests/v1/core/test_scheduler.py` 中新增 `test_update_from_output_routes_sampling_masks_by_request`，验证三个请求各自获得正确的采样掩码行。在 `tests/test_config.py` 中新增 `test_batch_sharded_sampling_rejects_return_sampling_mask`。端到端测试 `test_generate_sampling_mask` 也更新以验证 `logprobs` 内容与支持集的一致性。

关键文件：

- `vllm/v1/worker/gpu/sample/output.py`（模块 采样输出；类别 `source`；类型 `core-logic`；符号 `_compact_sampling_mask_kernel`, `tolists`, `support`, `from_logits`）：核心变更文件，包含新的 Triton 内核和 `SamplingMaskTensors` 结构，是性能优化的关键。
- `tests/v1/test_outputs.py`（模块 输出测试；类别 `test`；类型 `test-coverage`；符号 `test_sampling_mask_tensors_match_finite_support`, `test_sampling_mask_preserves_top_k_boundary_ties`）：测试更新文件，用参数化测试覆盖各种支持大小，验证采样掩码的正确性。
- `vllm/v1/outputs.py`（模块 输出定义；类别 `source`；类型 `data-contract`；符号 `slice_request`, `to_nested_list`）：定义 `SamplingMaskLists` 的文件，接口简化以适应单步单位置语义。
- `tests/v1/core/test_scheduler.py`（模块 调度器测试；类别 `test`；类型 `test-coverage`；符号 `test_update_from_output_routes_sampling_masks_by_request`）：新增调度器测试，验证采样掩码按请求正确路由。
- `vllm/v1/engine/output_processor.py`（模块 输出处理；类别 `source`；类型 `core-logic`）：前端响应构建文件，调整以直接从步骤切片构建列表，避免冗余合并。
- `vllm/config/vllm.py`（模块 配置验证；类别 `source`；类型 `configuration`）：配置验证文件，添加拒绝 `return_sampling_mask` 与 `batch-sharded sampling` 组合的检查。

关键符号：`_compact_sampling_mask_kernel`, `SamplingMaskTensors.from_logits`, `SamplingMaskTensors.tolists`, `SamplingMaskLists.slice_request`, `SamplingMaskLists.to_nested_list`, `test_sampling_mask_tensors_match_finite_support`, `test_sampling_mask_preserves_top_k_boundary_ties`, `test_update_from_output_routes_sampling_masks_by_request`, `test_batch_sharded_sampling_rejects_return_sampling_mask`

关键源码片段

`vllm/v1/worker/gpu/sample/output.py`

核心变更文件，包含新的 Triton 内核和 `SamplingMaskTensors` 结构，是性能优化的关键。

```

# 新内核 _compact_sampling_mask_kernel 在一次遍历中同时生成紧凑 id 数组和位掩码
@triton.jit
def _compact_sampling_mask_kernel(
    logits_ptr,
    logits_row_stride,
    logits_col_stride,
    num_sampled_tokens_ptr,
    token_ids_ptr,
    token_ids_row_stride,
    packed_mask_ptr,
    packed_mask_row_stride,
    counts_ptr,
    vocab_size,
    max_num_kept,
    BLOCK_SIZE: tl.constexpr,
):
    """Per row: first ``max_num_kept`` finite-logit ids, the full count, the bitmask."""
    req_idx = tl.program_id(0)
    is_active = tl.load(num_sampled_tokens_ptr + req_idx) > 0
    count = tl.zeros(), dtype=tl.int32

    for start_idx in range(0, vocab_size, BLOCK_SIZE):
        offsets = start_idx + tl.arange(0, BLOCK_SIZE)
        logits = tl.load(
            logits_ptr + req_idx * logits_row_stride + offsets * logits_col_stride,
            mask=offsets < vocab_size,
            other=-float("inf"),
        )
        keep = (logits > -float("inf")) & (logits < float("inf")) & is_active
        keep_i32 = keep.to(tl.int32)
        # 使用前缀和计算每个 keep 位置在紧凑数组中的目标位置
        pos = count + tl.cumsum(keep_i32, axis=0) - keep_i32
        # 仅当 keep 且位置在 max_num_kept 内时写入紧凑数组
        tl.store(
            token_ids_ptr + req_idx * token_ids_row_stride + pos,
            offsets.to(tl.int32),
            mask=keep & (pos < max_num_kept),
        )
        count += tl.sum(keep_i32)

    # 同时生成位掩码
    bits = tl.reshape(keep_i32, (BLOCK_SIZE // 8, 8)) << tl.arange(0, 8)[None, :]
    byte_offsets = start_idx // 8 + tl.arange(0, BLOCK_SIZE // 8)
    tl.store(
        packed_mask_ptr + req_idx * packed_mask_row_stride + byte_offsets,
        tl.sum(bits, axis=1).to(tl.uint8),
        mask=byte_offsets < tl.cdiv(vocab_size, 8),
    )

```

```
tl.store(counts_ptr + req_idx, count)
```

SamplingMaskTensors.tolists() 优先使用紧凑数组，仅在回退时解包位掩码

```
def tolists(self) -> SamplingMaskLists:
    """CSR over all requests; rows without a sampled token are empty."""
    counts = self.counts.cpu().numpy()
    token_ids = self.token_ids.cpu().numpy()
    packed_mask = self.packed_mask.cpu().numpy()
    width = token_ids.shape[1]

    def support(row: int) -> np.ndarray:
        if counts[row] <= width:
            return token_ids[row, : counts[row]]
        # 支持超过紧凑宽度时回退到位掩码
        bits = np.unpackbits(
            packed_mask[row], count=self.vocab_size, bitorder="little"
        )
        return np.flatnonzero(bits).astype(np.int32, copy=False)

    supports = [support(row) for row in range(len(counts))]
    offsets = np.zeros(len(supports) + 1, dtype=np.int64)
    np.cumsum([len(s) for s in supports], out=offsets[1:])
    return SamplingMaskLists(np.concatenate(supports), offsets)
```

评论区精华

Review评论较少，只有njhill的审批和肯定：'CorechangesLGTM'，表明核心实现获得认可。未发现技术争议或未解决的疑虑。PR body 中详细说明了设计权衡，包括 compact 宽度上限 `MAX_COMPACT_SUPPORT` 的选择、回退到位掩码的条件，以及拒绝与 batch-sharded sampling 组合的原因。

- 暂无高价值评论线程

风险与影响

- 风险：

1. 正确性风险：核心逻辑变更涉及采样掩码生成路径，但通过全面的单元测试和端到端测试覆盖了各种边界情况，包括空支持、小支持、大支持（超过 compact 宽度）和 top-k 边界平局，风险较低。
2. 内存风险：引入 [num_reqs, max_num_kept] 的 int32 缓冲区，通过 `MAX_COMPACT_SUPPORT = 2048` 上限防止内存爆炸，且当支持超过上限时回退到位掩码，确保内存使用可控。
3. 兼容性风险：SamplingMaskLists 的接口变化（offsets 变为可选）可能影响下游代码，但 Model Runner V2 中每个请求每步只输出一个位置，因此兼容。拒绝 `return_sampling_mask` 与 `enable_batch_sharded_sampling` 的组合，防止了静默失败。
4. 性能风险：GPU 内核引入额外计算，但测试表明 GPU+D2H 时间几乎不变（约 0.4-1.8 ms），而 CPU 处理时间从毫秒级降至亚毫秒级，整体性能显著提升。

- 影响:

1. 用户影响: 对于使用 `--return-sampling_mask` 的用户 (如训练回放场景), 解码吞吐量从约 2.6k tok/s 恢复到约 18k tok/s (512 并发), 解决了生产环境中的性能瓶颈。
2. 系统影响: 消除了 CPU 成为瓶颈的情况, 使引擎在高并发下能更好地扩展, 降低了多引擎部署中的负载不均衡。
3. 团队影响: 需要确保依赖采样掩码的代码 (如训练回放) 与新的单步单位置语义兼容。配置验证的增强防止了误用。 - 风险标记: 核心路径变更, 性能关键路径, GPU-CPU 协同, 数据结构接口变更

关联脉络

- PR #49577 [V1][Sampling] Add `--return-sampling-mask` option: 引入了采样掩码功能, 本 PR 解决了其导致的性能问题。
- PR #53826 [V1][Batch-Sharded] Forward sampling masks in `gather_sampler_output`: 涉及相同的文件, 但未解决 CPU 解包成本。本 PR 拒绝了与 batch-sharded sampling 的组合, 因为 `gather_sampler_output` 不转发掩码。