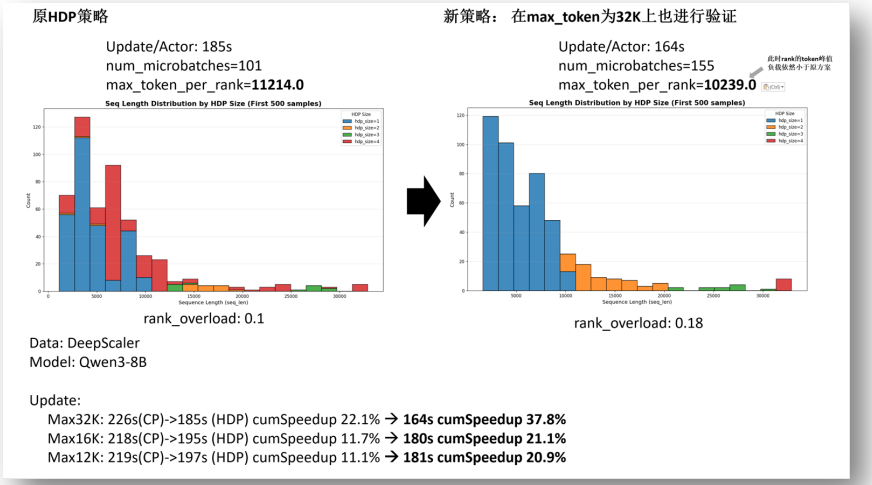
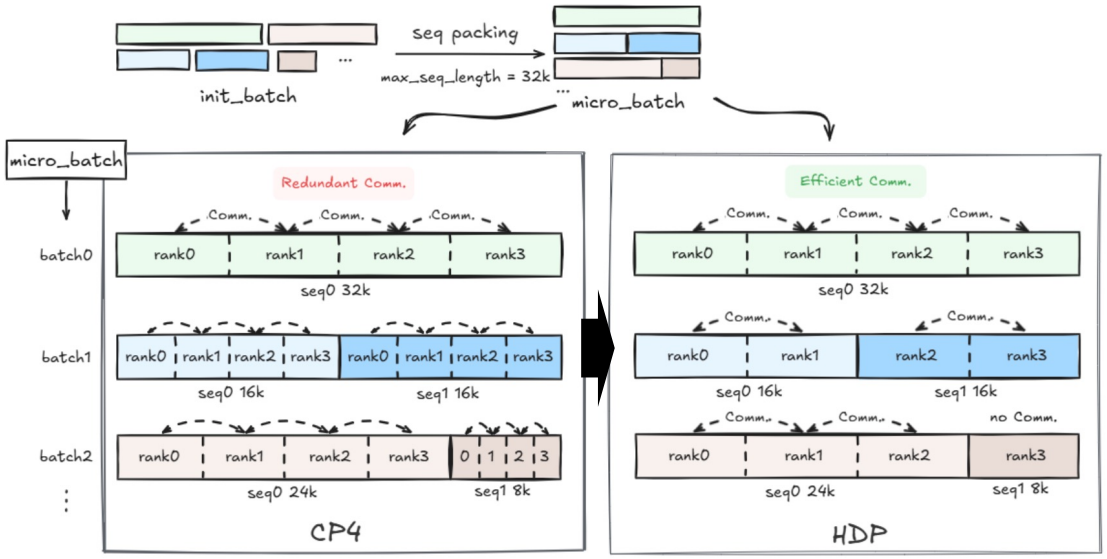
训练HDP优化 创新实践 -- 基于昇腾官方最佳实践[cann-recipes-train Link]

效果一瞥



背景一：Hybrid Data Parallel [link]



注：动态部署CP和DP, 减少小sample的CP通信冗余

1. HDP装箱分组算法设计

对于给定的micro_batch，本算法根据各序列长度以及总rank数对序列进行分组，以图示batch2：[seq0_length, seq1_length] = [24k, 8k]， cp_size = 4 的情况为例，最终的分组结果为：[[1, 2, 3], [4]]，seq0占用rank0~rank2执行CP3，seq1单独占用rank3。具体算法流程如下：

- 对于给定的micro_batch，首先根据序列长度及总rank数计算序列预计占用的rank数量。
- 根据每个序列占用rank数量来进行动态装箱，算法在初次遍历中会对各序列进行整体划分，并保留序列在装箱过程中溢出的部分及其对应的序列索引。
- 对于序列的溢出部分，算法将根据剩余rank数量进行轮转合并或均衡分组，得到最终的hdp_group。
- 另外，算法设定了上、下阈值来保证算法的鲁棒性。

背景一：Hybrid Data Parallel [link]

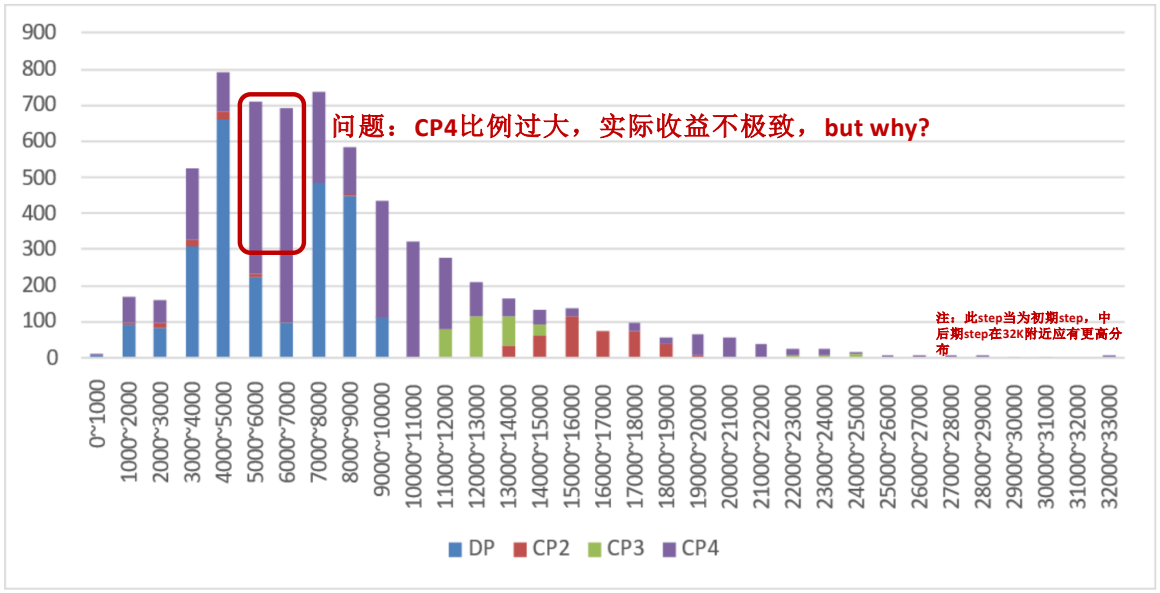
实验结果

目前在Deepscaler数据集、GBS512、TP4 PP4 CP4/HDP 128die的设置下进行实验，结果显示：前3个step的训练耗时平均收益仅5%左右，RL端到端性能收益更小。

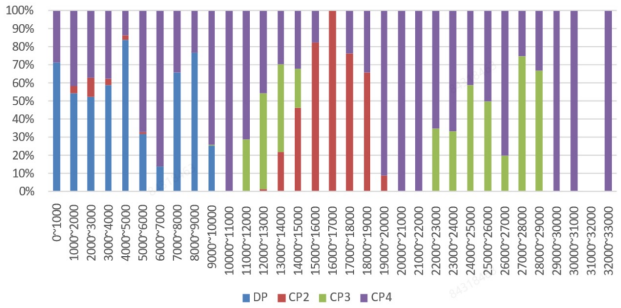
通过分析，性能收益不显著主要有以下两点原因：

- Deepscaler数据集生成序列普遍较长，在该数据长度分布情况下，HDP分组难以做到负载完全均衡，蚕食了通信耗时减少带来的性能收益。
- HDP主要是为了节约CP切分后RingAttention内的通信开销，而在CP大小仅为4的设置下，HDP分组调整空间非常有限。

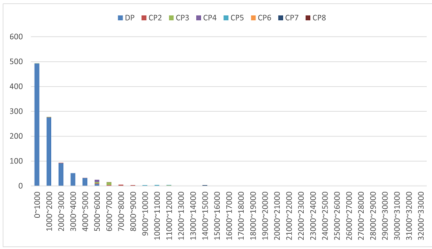
实验随机选取某一step，统计该步内所有序列的长度分布，以及各长度区间中序列经过HDP算法调度后进行的CP/DP的详细情况，如下图所示。



由图中结果可以发现，大多数序列的长度集中在3000以上，这也导致众多序列无法进行DP，转而寻求次优解（CP2、CP3）甚至是CP4。



我们还与HDP获得较高收益的场景进行了实验对比，设置如下：openr1-220k数据集、GBS256、TP4 EP32 PP8 CP8/HDP 256die，模型采用DeepSeekv3。实验同样统计step内所有序列长度以及在不同区间内的各个序列通过HDP算法后进行的CP/DP的详细情况，如下图所示。



图中可以看到，在该数据集生成的序列普遍集中在3000以下时，大多数的序列都由原来的纯CP8转为DP，这极大程度减少了通信过程的开销，并且基本不会有负载不均的问题。该场景的性能数据如下，最终训练耗时收益在25%以上。

注：不符合实际场景，但可作收益上限参考

复现HDP [cann-recipes-train Link]

出乎意料的是，其MicrobatchRearrange的方案已经是Greedy， 依旧会产生如右下图的HDP分布

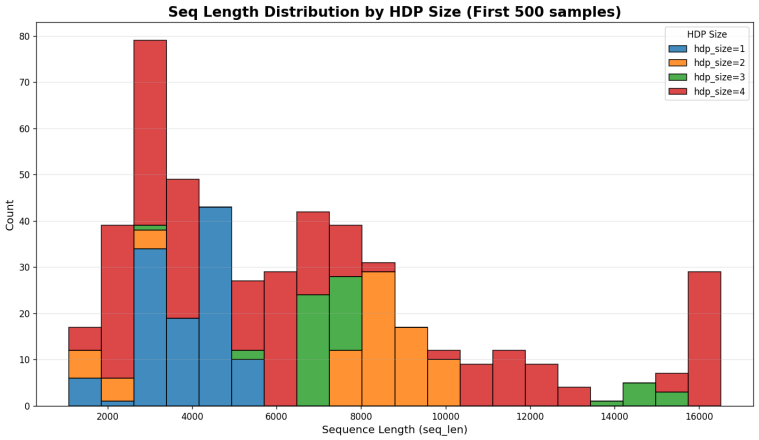
阶段一 **阶段二**

pack_sequences_into_buckets(决定每个microbatch的填充) → **generate_hdp_group_from_batch**(对每个microbatch进行HDP装箱)

阶段一 伪代码

1. 先对trajectory降序排序
2. 遍历每个trajectory
3. 对每个trajectory, 遍历MicroBatches, 从第一个MicroBatch尝试能否填入(根据MB的token上限)

```python
def pack_sequences_into_buckets(seqlen_list: list[int], max_bucket_length: int, num_buckets: int) -> list[list[int]]:
    """
    Pack sequences into buckets using greedy algorithm for subsequent HDP grouping.

    Args:
        seqlen_list: List of sequence lengths to be packed
        max_bucket_length: Maximum allowed total sequence length per bucket
        num_buckets: Number of available buckets for distribution

    Returns:
        List of buckets, where each bucket contains the original indices of sequences assigned to it

    """
    # Sort by length in descending order, preserving original indices
    indexed_seqlens = sorted(enumerate(seqlen_list), key=lambda x: -x[1])

    buckets = [[] for _ in range(num_buckets)]
    bucket_sums = [0] * num_buckets

    # Assign each sequence to the first bucket that can accommodate it
    for idx, length in indexed_seqlens:
        placed = False
        for b in range(num_buckets):
            if bucket_sums[b] + length <= max_bucket_length:
                buckets[b].append(idx)
                bucket_sums[b] += length
                placed = True
                break
        if not placed:
            # If no bucket can accommodate, place in the bucket with the smallest current total length
            min_bucket = min(range(num_buckets), key=lambda b: bucket_sums[b])
            buckets[min_bucket].append(idx)
            bucket_sums[min_bucket] += length

    return buckets
```



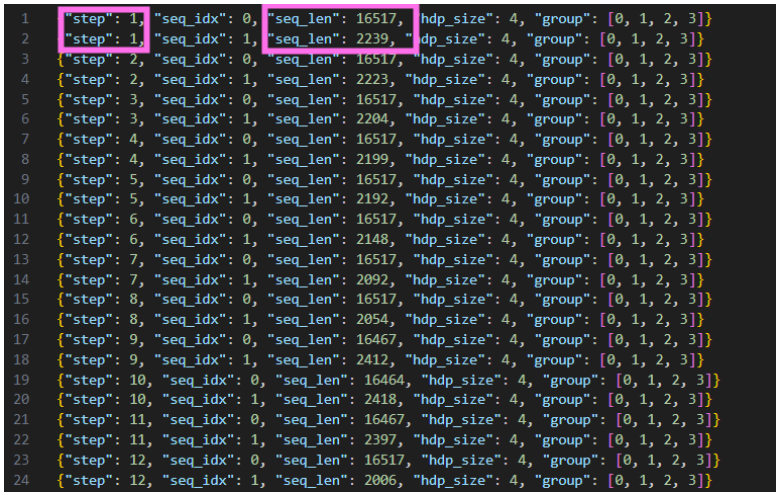
Data: DeepScaler
Model: Qwen3-8B
Update:

218s->195s (max16K) 原HDP策略Speedup 11.7%
219s->197s (max12K) 原HDP策略Speedup 11.1%

观察一：“四带一，要不起”

max_token_len := max_prompt_len(2048) + max_response_len(16384)

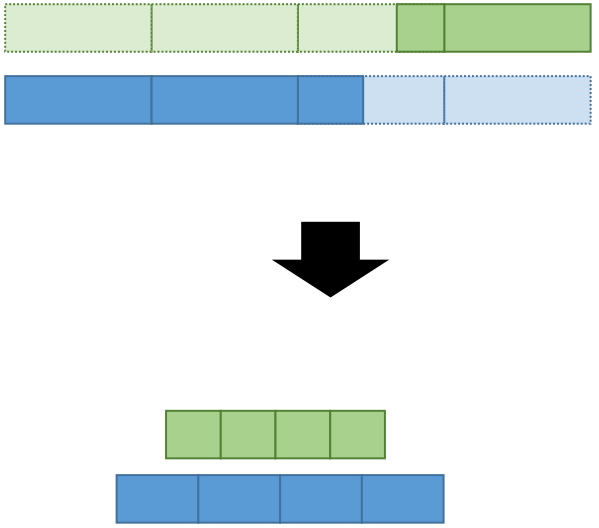
但是prompt_length/mean:78.15625普遍很短，
所以长序列填完HDP Group(CP4)后还可以再填装一个短序列。



```
{"step": 1, "seq_idx": 0, "seq_len": 16517, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 1, "seq_idx": 1, "seq_len": 2239, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 2, "seq_idx": 0, "seq_len": 16517, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 2, "seq_idx": 1, "seq_len": 2223, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 3, "seq_idx": 0, "seq_len": 16517, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 3, "seq_idx": 1, "seq_len": 2204, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 4, "seq_idx": 0, "seq_len": 16517, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 4, "seq_idx": 1, "seq_len": 2199, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 5, "seq_idx": 0, "seq_len": 16517, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 5, "seq_idx": 1, "seq_len": 2192, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 6, "seq_idx": 0, "seq_len": 16517, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 6, "seq_idx": 1, "seq_len": 2148, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 7, "seq_idx": 0, "seq_len": 16517, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 7, "seq_idx": 1, "seq_len": 2092, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 8, "seq_idx": 0, "seq_len": 16517, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 8, "seq_idx": 1, "seq_len": 2054, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 9, "seq_idx": 0, "seq_len": 16467, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 9, "seq_idx": 1, "seq_len": 2412, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 10, "seq_idx": 0, "seq_len": 16464, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 10, "seq_idx": 1, "seq_len": 2418, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 11, "seq_idx": 0, "seq_len": 16467, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 11, "seq_idx": 1, "seq_len": 2397, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 12, "seq_idx": 0, "seq_len": 16517, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 12, "seq_idx": 1, "seq_len": 2006, "hdp_size": 4, "group": [0, 1, 2, 3]}
```

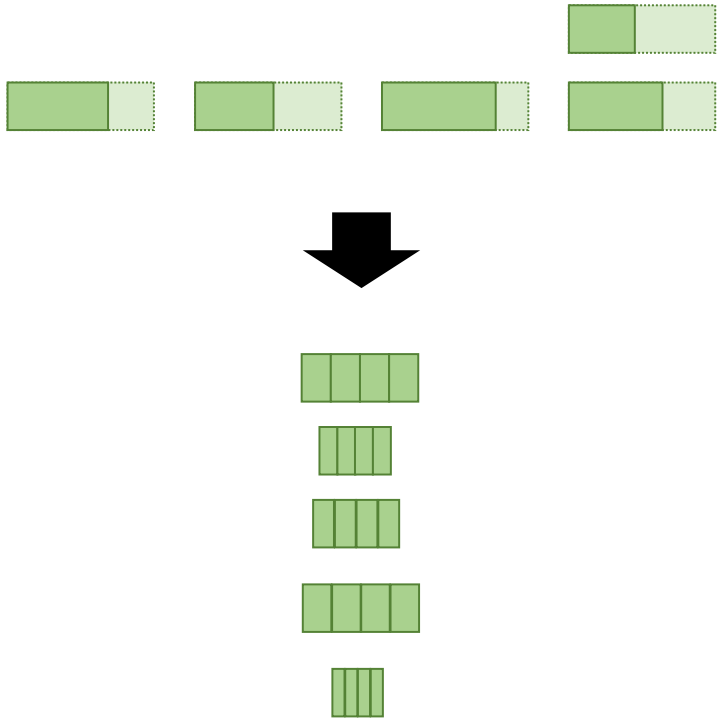
观察二：“叠手手”


```
{"step": 34, "seq_idx": 0, "seq_len": 15169, "hdp_size": 3, "group": [0, 1, 2]}
{"step": 34, "seq_idx": 1, "seq_len": 3258, "hdp_size": 1, "group": [3]}
{"step": 35, "seq_idx": 0, "seq_len": 15152, "hdp_size": 3, "group": [0, 1, 2]}
{"step": 35, "seq_idx": 1, "seq_len": 3273, "hdp_size": 1, "group": [3]}
{"step": 36, "seq_idx": 0, "seq_len": 15138, "hdp_size": 3, "group": [0, 1, 2]}
{"step": 36, "seq_idx": 1, "seq_len": 3286, "hdp_size": 1, "group": [3]}
{"step": 37, "seq_idx": 0, "seq_len": 14964, "hdp_size": 3, "group": [0, 1, 2]}
{"step": 37, "seq_idx": 1, "seq_len": 3463, "hdp_size": 1, "group": [3]}
{"step": 38, "seq_idx": 0, "seq_len": 14936, "hdp_size": 3, "group": [0, 1, 2]}
{"step": 38, "seq_idx": 1, "seq_len": 3491, "hdp_size": 1, "group": [3]}
{"step": 39, "seq_idx": 0, "seq_len": 14920, "hdp_size": 3, "group": [0, 1, 2]}
{"step": 39, "seq_idx": 1, "seq_len": 3498, "hdp_size": 1, "group": [3]}
{"step": 40, "seq_idx": 0, "seq_len": 14797, "hdp_size": 3, "group": [0, 1, 2]}
{"step": 40, "seq_idx": 1, "seq_len": 3624, "hdp_size": 1, "group": [3]}
{"step": 41, "seq_idx": 0, "seq_len": 14403, "hdp_size": 3, "group": [0, 1, 2]}
{"step": 41, "seq_idx": 1, "seq_len": 4011, "hdp_size": 1, "group": [3]}
{"step": 42, "seq_idx": 0, "seq_len": 13750, "hdp_size": 3, "group": [0, 1, 2]}
{"step": 42, "seq_idx": 1, "seq_len": 4633, "hdp_size": 1, "group": [3]}
{"step": 43, "seq_idx": 0, "seq_len": 13128, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 43, "seq_idx": 1, "seq_len": 5273, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 44, "seq_idx": 0, "seq_len": 12994, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 44, "seq_idx": 1, "seq_len": 5425, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 45, "seq_idx": 0, "seq_len": 12816, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 45, "seq_idx": 1, "seq_len": 5588, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 46, "seq_idx": 0, "seq_len": 12718, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 46, "seq_idx": 1, "seq_len": 5558, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 47, "seq_idx": 0, "seq_len": 12606, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 47, "seq_idx": 1, "seq_len": 5799, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 48, "seq_idx": 0, "seq_len": 12449, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 48, "seq_idx": 1, "seq_len": 5915, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 49, "seq_idx": 0, "seq_len": 12066, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 49, "seq_idx": 1, "seq_len": 6354, "hdp_size": 4, "group": [0, 1, 2, 3]}
```

观察三：“卷起千堆雪”


```
{"step": 158, "seq_idx": 0, "seq_len": 4222, "hdp_size": 1, "group": [0]}
{"step": 158, "seq_idx": 1, "seq_len": 4205, "hdp_size": 1, "group": [1]}
{"step": 158, "seq_idx": 2, "seq_len": 4194, "hdp_size": 1, "group": [2]}
{"step": 158, "seq_idx": 3, "seq_len": 4190, "hdp_size": 1, "group": [3]}
{"step": 158, "seq_idx": 4, "seq_len": 1584, "hdp_size": 1, "group": [3]}
{"step": 159, "seq_idx": 0, "seq_len": 4179, "hdp_size": 1, "group": [0]}
{"step": 159, "seq_idx": 1, "seq_len": 4155, "hdp_size": 1, "group": [1]}
{"step": 159, "seq_idx": 2, "seq_len": 4153, "hdp_size": 1, "group": [2]}
{"step": 159, "seq_idx": 3, "seq_len": 4147, "hdp_size": 1, "group": [3]}
{"step": 159, "seq_idx": 4, "seq_len": 1763, "hdp_size": 1, "group": [3]}
{"step": 160, "seq_idx": 0, "seq_len": 4079, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 160, "seq_idx": 1, "seq_len": 3998, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 160, "seq_idx": 2, "seq_len": 3993, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 160, "seq_idx": 3, "seq_len": 3992, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 160, "seq_idx": 4, "seq_len": 2547, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 161, "seq_idx": 0, "seq_len": 3984, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 161, "seq_idx": 1, "seq_len": 3935, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 161, "seq_idx": 2, "seq_len": 3915, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 161, "seq_idx": 3, "seq_len": 3901, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 161, "seq_idx": 4, "seq_len": 2675, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 162, "seq_idx": 0, "seq_len": 3826, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 162, "seq_idx": 1, "seq_len": 3793, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 162, "seq_idx": 2, "seq_len": 3772, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 162, "seq_idx": 3, "seq_len": 3738, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 162, "seq_idx": 4, "seq_len": 3292, "hdp_size": 4, "group": [0, 1, 2, 3]}
{"step": 163, "seq_idx": 0, "seq_len": 3714, "hdp_size": 4, "group": [0, 1, 2, 3]}
```

解决方案

选项一：二段式 → 一段式 （分/装一体）
选项二：依旧二段式，但保持逻辑一致 （分batch时考虑装箱策略）

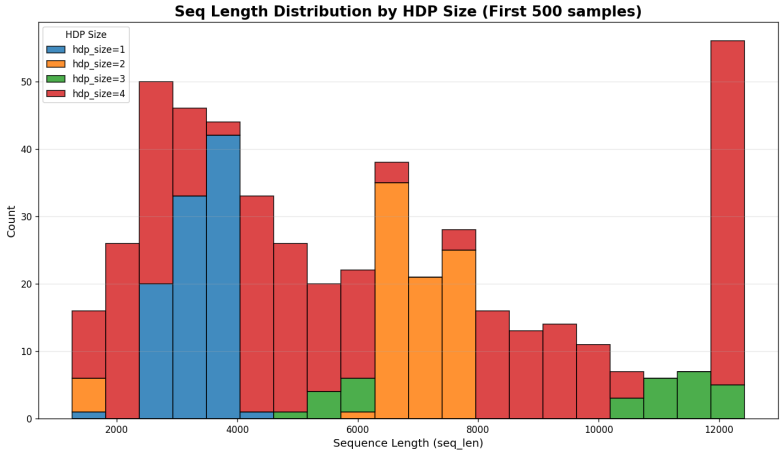
工程上，综合考虑代码结构，选择选项二

主旨：分batch时, 对每个microbatch按照装箱逻辑进行分配，消除后续装箱策略的序列堆叠。(即，按照当前CPsize的装箱规则一条一条填充，排除舍入等复杂逻辑，如此，在装箱时也不用考虑舍入，可以无脑装箱)

注：由于此时num_microbatch是动态决定的，allreduce.MAX(num_batches, groups=dp_group)后，要把尾部的装箱拆开，对齐DP间的num_microbatch

原**HDP**策略

Update/Actor: 197s
num_microbatches=215
max_token_per_rank=**4481.0**

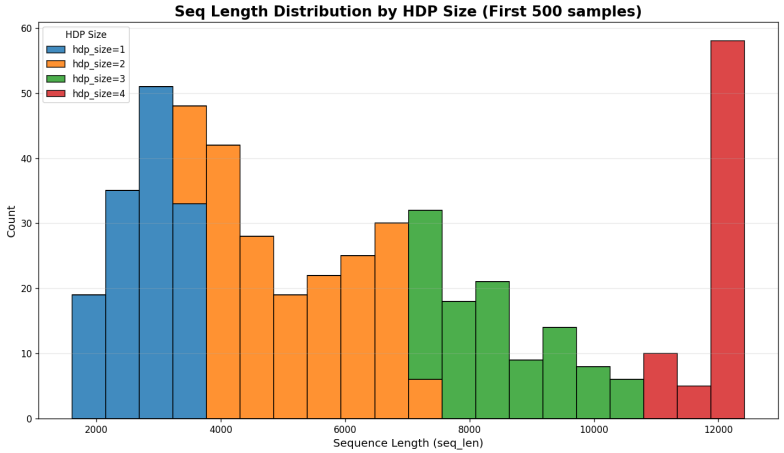


新策略：

新方案会让
microbatch变稀疏，
反而降低训练效率

Update/Actor: 211s
num_microbatches=281
max_token_per_rank=**3571.0**

但最大rank的token
负载小于原方案，
可以利用并优化



Data: DeepScaler
Model: Qwen3-8B

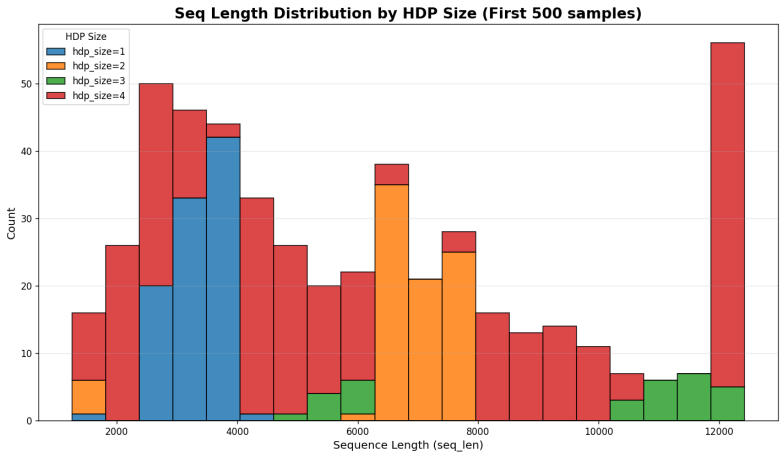
Update:

Max16K: 218s(CP)->195s (HDP) cumSpeedup 11.7%

Max12K: 219s(CP)->197s (HDP) cumSpeedup 11.1% → **211s cumSpeedup 3.0%**

原HDP策略

Update/Actor: 197s
num_microbatches=215
max_token_per_rank=**4481.0**

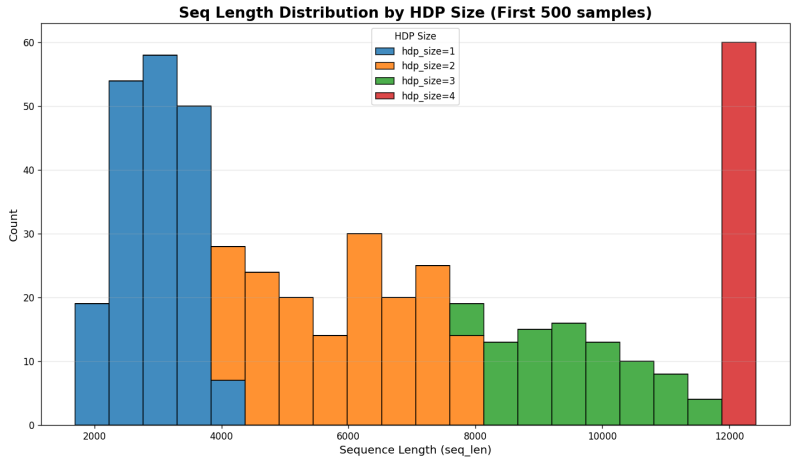


rank_overload: 0.1

新策略： 借鉴原HDP策略，也设置每个rank的token overload

Update/Actor: 192s
num_microbatches=252
max_token_per_rank=**3935.0**

此时rank的token峰值负载依然小于原方案



rank_overload: 0.1

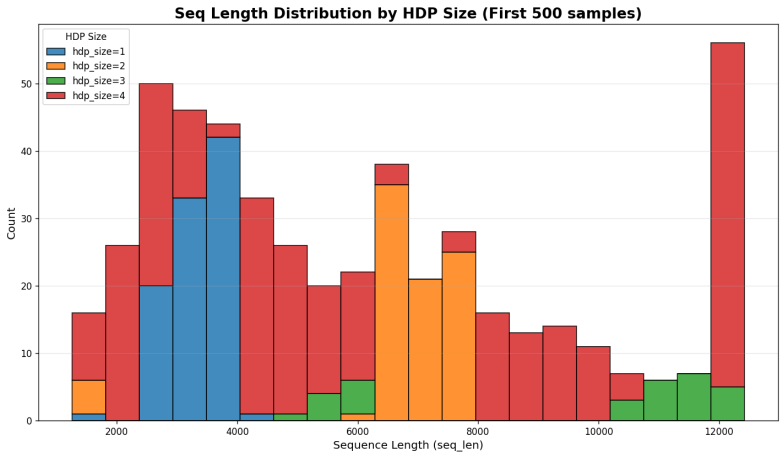
Data: DeepScaler
Model: Qwen3-8B

Update:

Max16K: 218s(CP)->195s (HDP) cumSpeedup 11.7%
Max12K: 219s(CP)->197s (HDP) cumSpeedup 11.1% → 211s → **192s cumSpeedup 14.0%**

原**HDP**策略

Update/Actor: 197s
num_microbatches=215
max_token_per_rank=**4481.0**

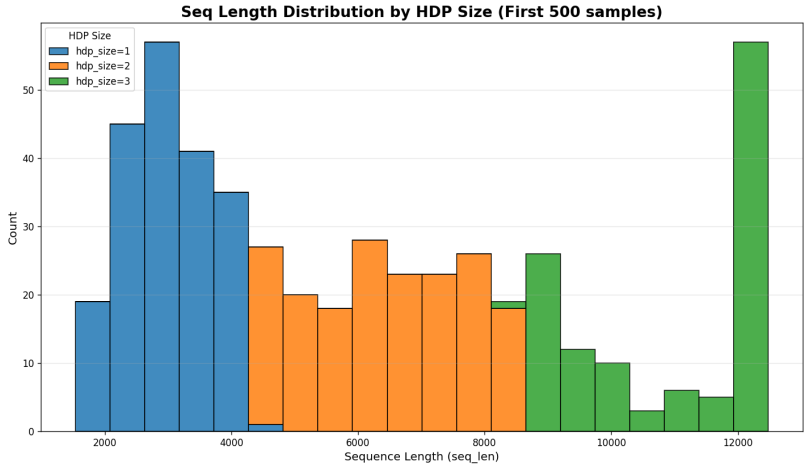


rank_overload: 0.1

新策略：观察到**rank**的**token**负载依旧低于原方案，调大**overload**

Update/Actor: 181s
num_microbatches=235
max_token_per_rank=**4222.0**

此时rank的token峰值
负载依然小于原方案



rank_overload: 0.18

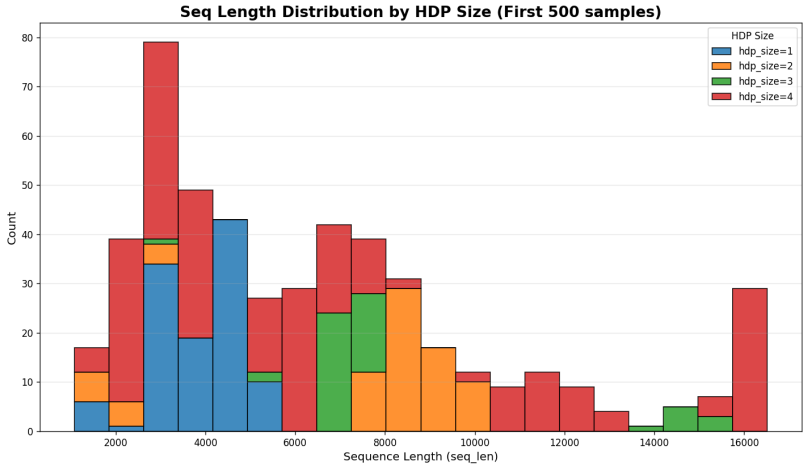
Data: DeepScaler
Model: Qwen3-8B

Update:

Max16K: 218s(CP)->195s (HDP) cumSpeedup 11.7%
Max12K: 219s(CP)->197s (HDP) cumSpeedup 11.1% → 211s → 192s → **181s cumSpeedup 20.9%**

原**HDP策略**

Update/Actor: 195s
num_microbatches=177
max_token_per_rank=**5910.0**

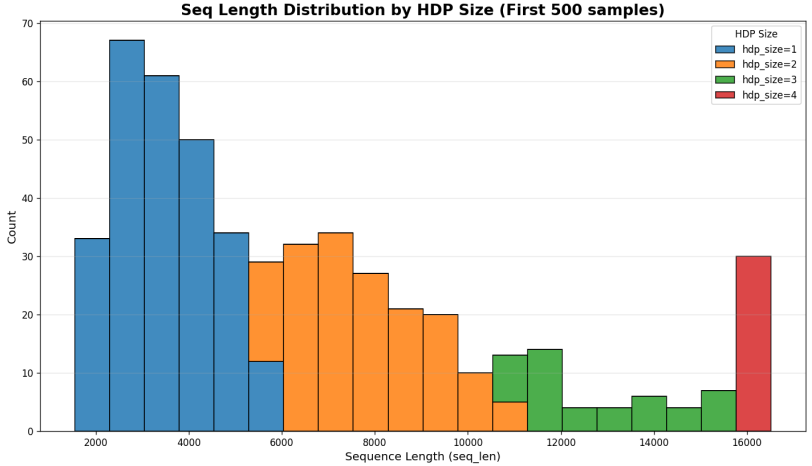


rank_overload: 0.1

新策略： 在**max_token为16K上也进行验证**

Update/Actor: 180s
num_microbatches=220
max_token_per_rank=**5415.0**

此时rank的token峰值
负载依然小于原方案



rank_overload: 0.18

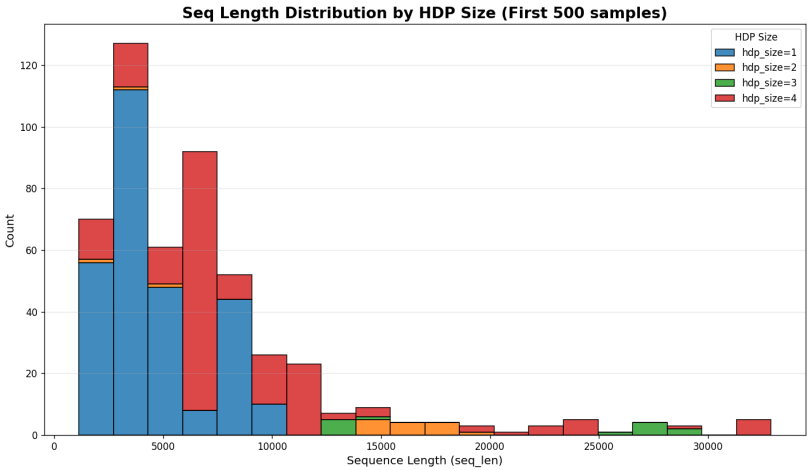
Data: DeepScaler
Model: Qwen3-8B

Update:

Max16K: 218s(CP)->195s (HDP) cumSpeedup 11.7% → **180s cumSpeedup 21.1%**
Max12K: 219s(CP)->197s (HDP) cumSpeedup 11.1% → 211s → 192s → **181s cumSpeedup 20.9%**

原**HDP**策略

Update/Actor: 185s
num_microbatches=101
max_token_per_rank=**11214.0**

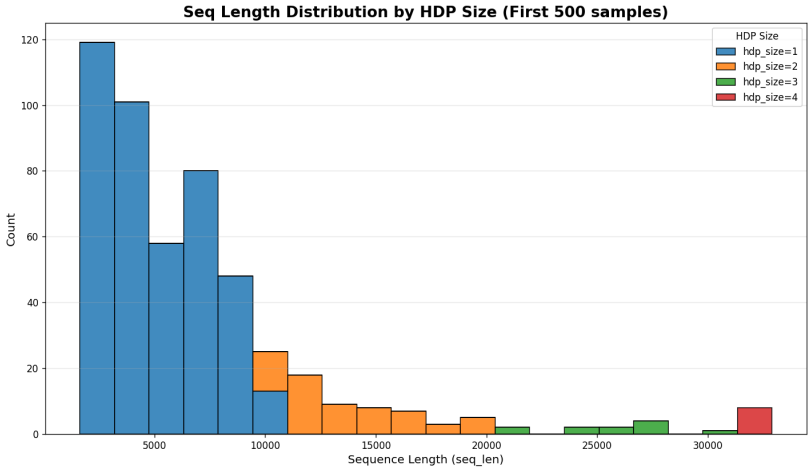


rank_overload: 0.1

新策略： 在**max_token**为**32K**上也进行验证

Update/Actor: 164s
num_microbatches=155
max_token_per_rank=**10239.0**

此时rank的token峰值
负载依然小于原方案



rank_overload: 0.18

Data: DeepScaler
Model: Qwen3-8B

Update:

Max32K: 226s(CP)->185s (HDP) cumSpeedup 22.1% → **164s cumSpeedup 37.8%**
Max16K: 218s(CP)->195s (HDP) cumSpeedup 11.7% → **180s cumSpeedup 21.1%**
Max12K: 219s(CP)->197s (HDP) cumSpeedup 11.1% → **181s cumSpeedup 20.9%**

总结：

原方案中先allreduce.max了num_microbatch，先确定了num_microbatch。

于是只能按照固定的microbatch数量来分配, 使得分配时首要顾及了num_microbatch，难以考虑消除序列堆叠和舍入。使得hdp装箱逻辑复杂。

优化方案，优先考虑microbatch分配和hdp装箱的逻辑一致性，后依靠拆分少量batch来，对齐dp间的num_microbatch。从而消除了hdp中的“bad case”

固定CP	原HDP策略	新方案
218s->195s (max16K) cumSpeedup 11.7%	→ **180s cumSpeedup 21.1%**	
219s->197s (max12K) cumSpeedup 11.1%	→ 211s → 192s	→ **181s cumSpeedup 20.9%**