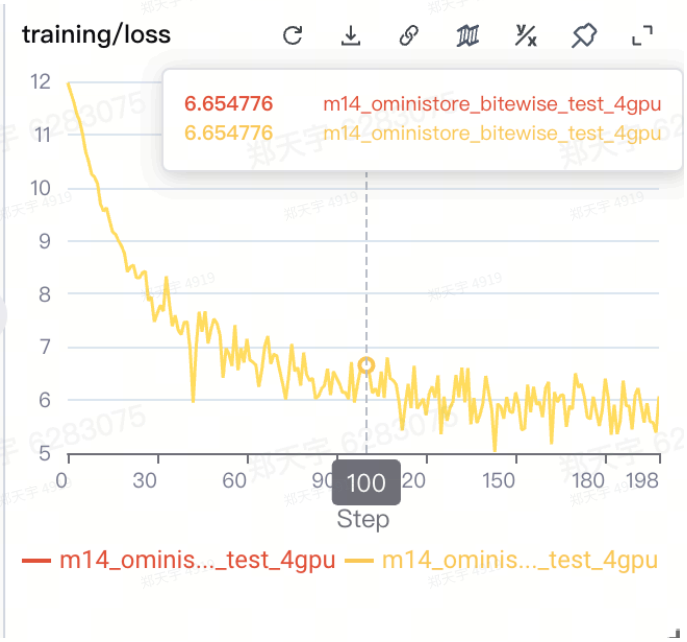
精度对齐小结

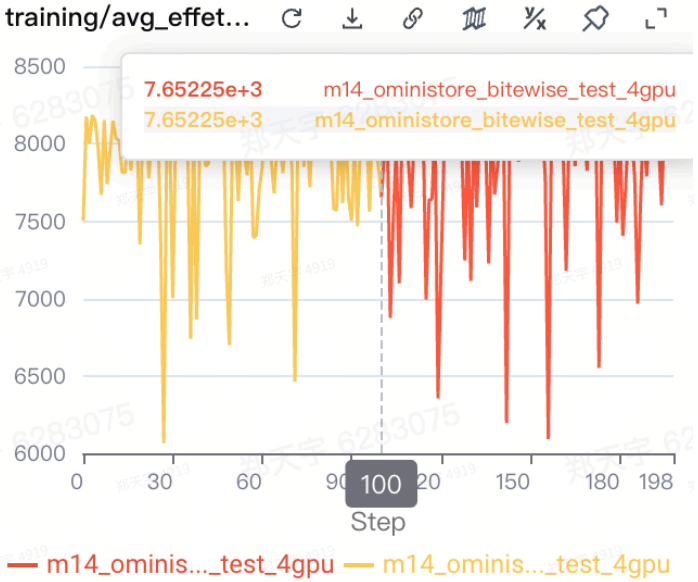
1. 同框架save resume bitewise对齐

Megatron纯文本对齐

单机4卡 gb200测试(pp2dsp2)：

Tracking: https://seed.byteintl.net/experiment/tracking/detail?Id=project_20260317_6645be31&selectedTrial=&tab=CHART&viewId=view_20260513_4eb57b44

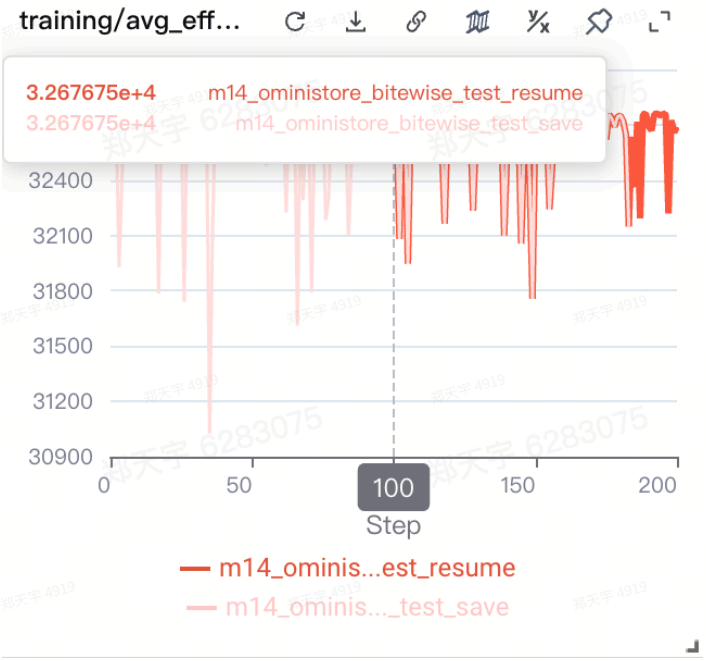


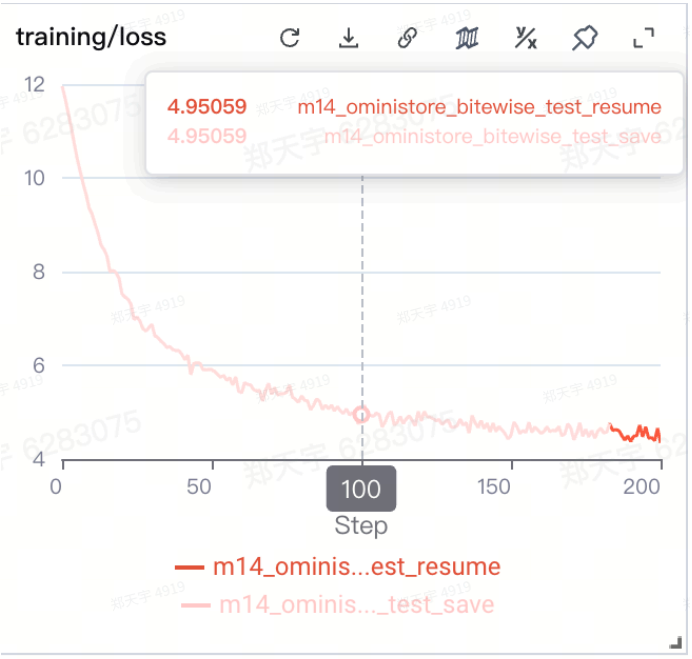


bitewise对齐

64 gb200测试：

https://seed.byteintl.net/experiment/tracking/detail?Id=project_20260317_6645be31&selectedTrial=&tab=CHART&viewId=view_20260513_5deec4c6





bitewise对齐。

Megatron omnipt save resume 对齐

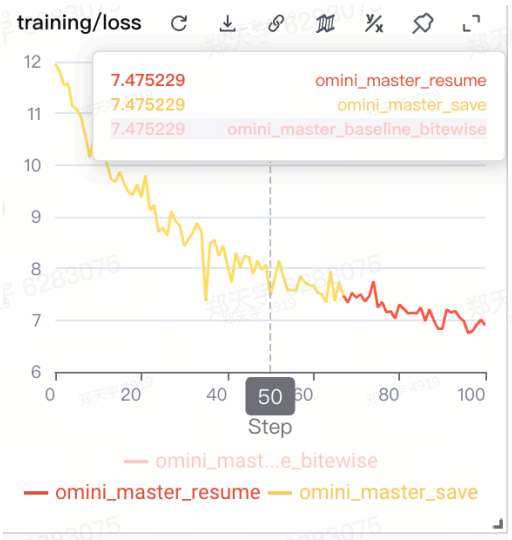
单机4卡 gb200测试(pp2dsp2):

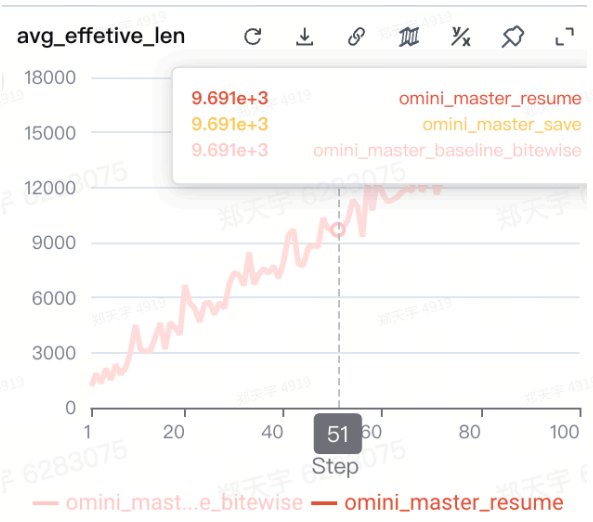
Tracking: https://seed.byteintl.net/experiment/tracking/detail?Id=project_20260406_8d7554da&selectedTrial=&tab=CHART&viewId=view_20260513_d77356a0

bitewise对齐

64卡 gb200测试:

Tracking: https://seed.byteintl.net/experiment/tracking/detail?Id=project_20260512_2fe9faed&selectedTrial=&tab=CHART&viewId=view_20260514_c0abf8a4





bitewise对齐

2. 首次处理fsdp与megatron omnipt 精度对齐(save resume)

1.key mapping对齐（确保ckpt正确加载）

	fsdp	megatron
loader文件命名	ran_i	ran_0i
Dataloder key	MultimodalDateModule.train_dataloader.xxx	train_dataloader.xxx
llm key	language_model.transfomer.xxx	transfomer.xxx
Vision key	vision_encoder.xxxx	visual_encoder.xx , ln_vision.xx, seed_proj.xx
Audio key	audio_encoeder.xxx	adapter.xx , backbone.xx, frontend.xxx
多的key	无	transfomer.xxx.xxx.rotary_embedding.inv_freq transfomer.norm.weight
Oe embedding	over_encoderd_embeddings(有padding)	over_encoderd_embedding(无padding)
optimizer的参数	1.无noop偏移 2.动量表示： 一阶：.m 二阶：.v	1.有noop偏移 2.动量表示： 一阶：.muon_buffer 二阶：.adamw_exp_avg_sq

	3.无main_param_data	3.有 `optimizer.shard_main_param_data.*` 额外主参数数据

mr:
https://code.byted.org/seed/mariana/commit/39e47d055af4cbe312cc766f926865beb922c2fd

2.config对齐

config对齐记录

- 自己整理对齐两边同时存在且同名同功效的config。
- 针对非同名参数，结构功能等不同的参数：拉通相关负责人，经过1h会议，高效完成 FSDP 与 Megatron 两侧 config 对齐、训练指标对齐及加载/观测数据对齐。

3.tracking 指标统一对齐管理：

tracking 指标统一对齐管理

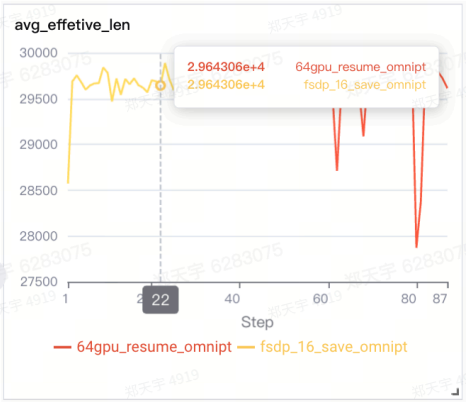
4.集群间迁移跑通实验总结

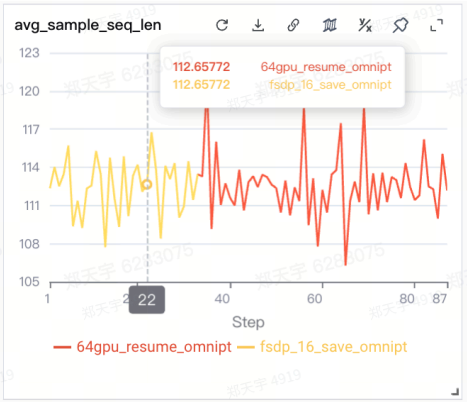
dallas和悉尼切换注意事项

5.实验对齐细节

1.三模态omnipt 数据与初始参数对齐

- 为了对齐fsdp和megatron的训练指标，首先得确保数据的消费与加载的ckpt完全对齐：





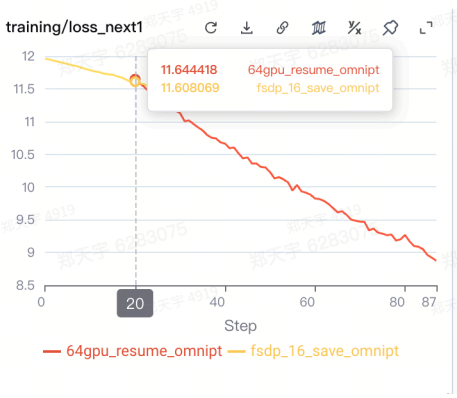
主要观测avg_effetive_len和avg_samle_seq_len两个指标：

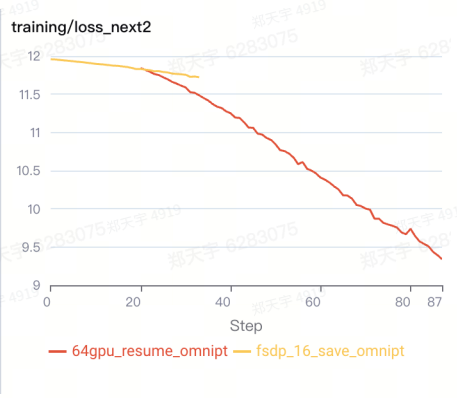
avg_effective_len bitewise对齐：表示dp间所有数据的token总数，除以batch数，即平均每个batch的有效token数

avg_sample_seq_len bitewise对齐：表示dp间所有数据的token总数，除以有效的sample数，即平均每个sample的有效token数

实验链接：https://seed.byteintl.net/experiment/tracking/detail?Id=project_20260512_2fe9faed&selectedTrial=&tab=CHART&viewId=view_20260519_7c269725

- 然而观察到，此时一些重要的观测指标，loss_next1/loss_next2,grad_norm还有较大差异，不符合预期：





因此决定先在纯文本上做精度对齐，从而排除文本部分的影响，因为megatron omnipt的结构是vision和audio分别对应一个encoder部分，最终经过adapter，将维度转化与文本拼接一起送入llm，因此先测文本数据，可以确保llm部分的各项性能优化没有逻辑上的问题。

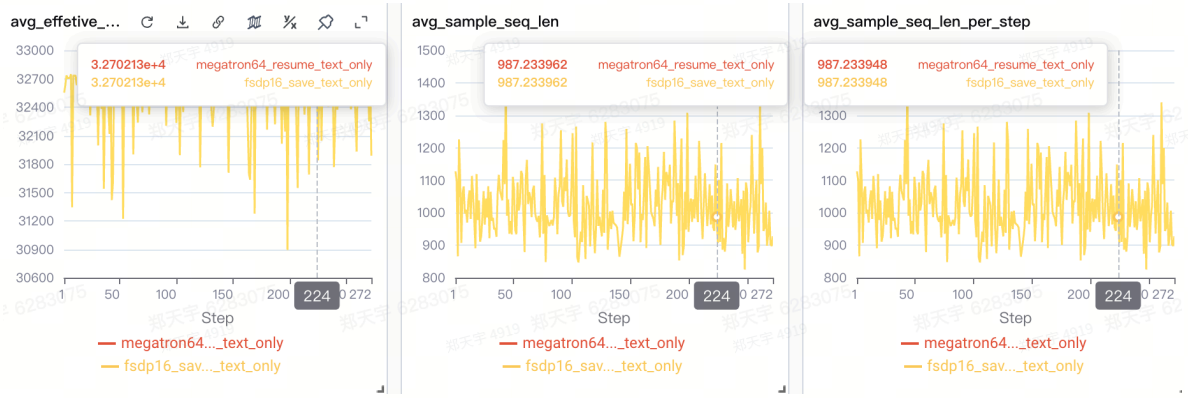
2. 文本数据 megatron omnipt 与 fsdp 对齐

主要对齐验证：

1. optimizer参数两边用的不同的两套优化器结构，因此存在未对齐的参数，进行了二次对齐
2. Dropout 对齐

实验结果: https://seed.byteintl.net/experiment/tracking/detail?Id=project_20260512_2fe9faed&selectedTrial=&tab=CHART&viewId=view_20260520_59262fb9

1. dataloader对齐



2. grad_norm,loss_nex1,loss_next2趋势基本吻合（数据换成纯文本，optimizer相关系数对齐，biattention对齐，droupout对齐）：

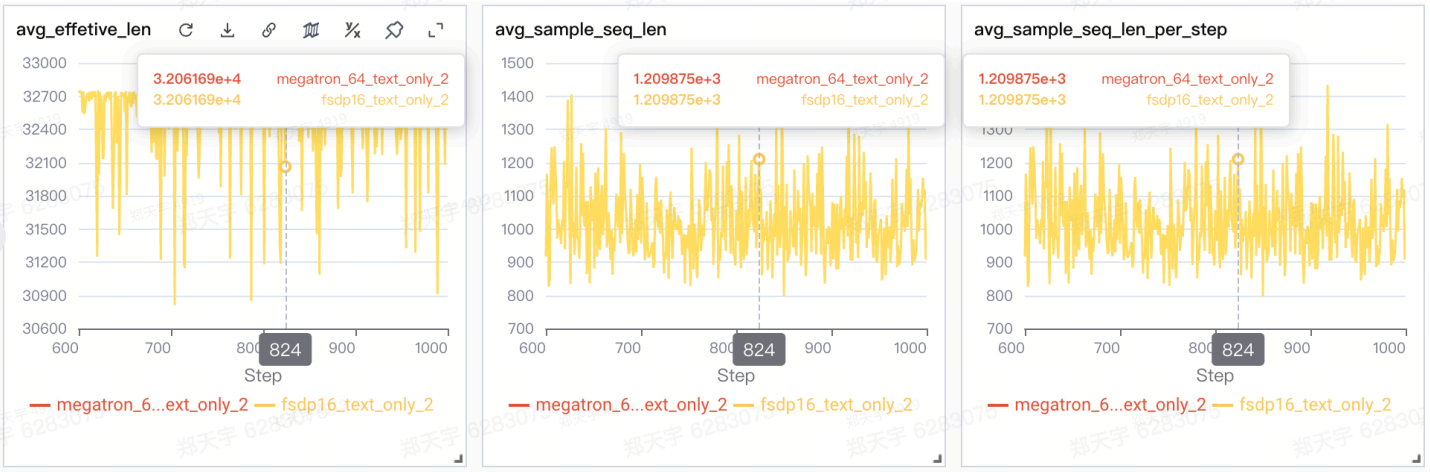
这里grad_norm和loss的趋势基本吻合，但loss和grad_norm都在百分位上的差距，因此还不符合预期，存在影响纯文本训练的config仍没对齐。

3. 文本数据 megatron omnipt 与 fsdp 对齐_2

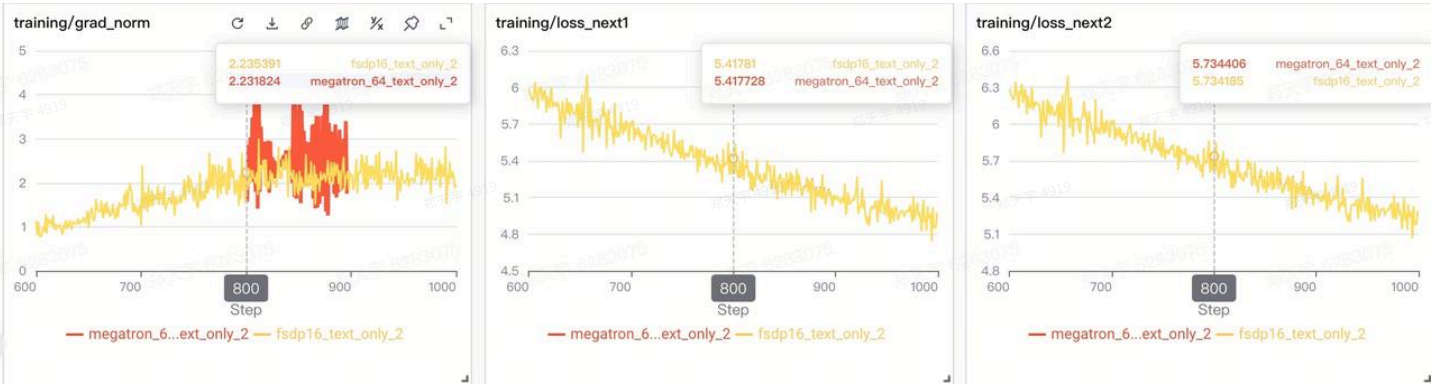
目前看大部分config肯定是对齐了的，因此直接将两边的cruise_cli.json，也就是实际运行时的config dump下来，直接对比文本相关的config，做了进一步对齐。

实验链接：https://seed.byteintl.net/experiment/tracking/detail?Id=project_20260512_2fe9faed&selectedTrial=&tab=CHART&viewId=view_20260521_9171a6aa

dataloader对齐(bitewise)：



grad_nrom、loss_next1、loss_next2对齐(e-3):



可以看到，grad_norm在千分位，loss在万分位对齐了，基本符合预期，但是grad_norm，明显能看出，megatron的变化幅度更大，因此需要分析细项来分析原因。

4. 文本对齐的细项分析

这里做了一些参数影响的验证，包括dropout开关，num_heads异同等等参数是否有影响，这里放一些主要的对齐验证：

1. z_loss差距较大验证

两边分别用的fuse_ce_zloss_logits_kernel和seed_fused_linear_cross_entropy_and_zloss，但是看了底层实现原理计算是一致的。

发现是统计口径不同：

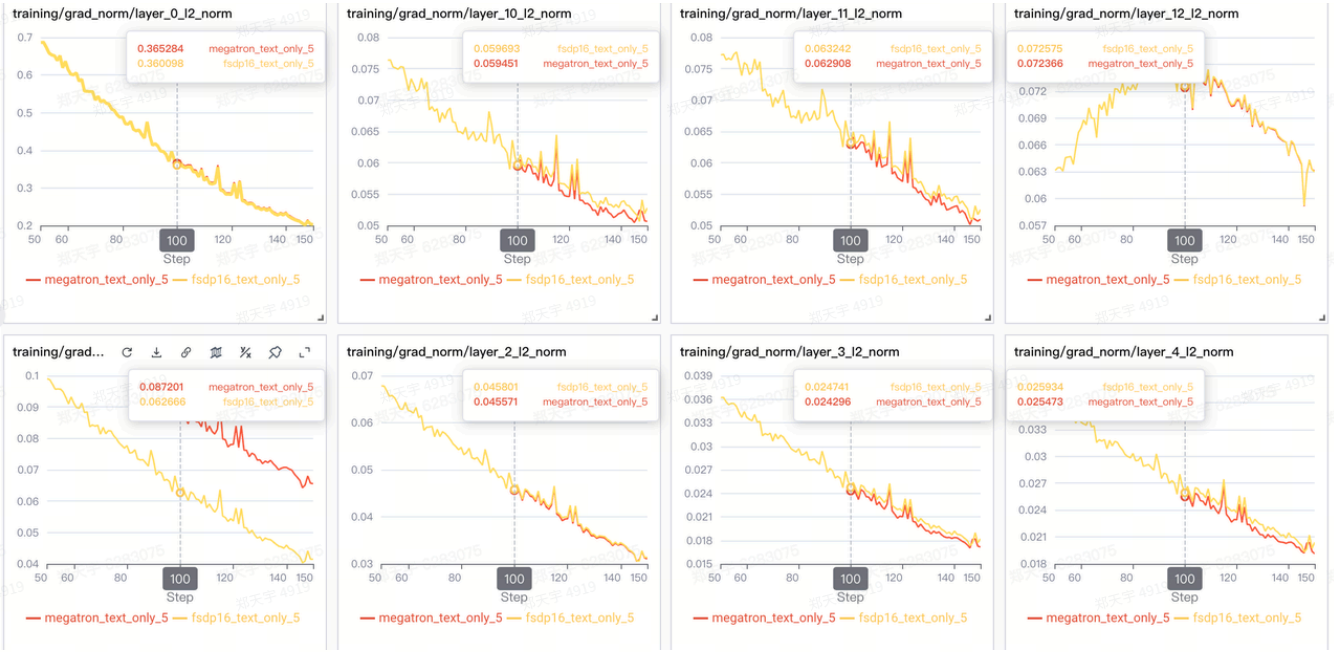
fsdp：每个dp rank 计算一个z_loss标量，即所有token 平均，然后再在dp间做平均。

megatron：所有dp rank的所有token 的z_loss做一个均值计算，即按token加权平均。

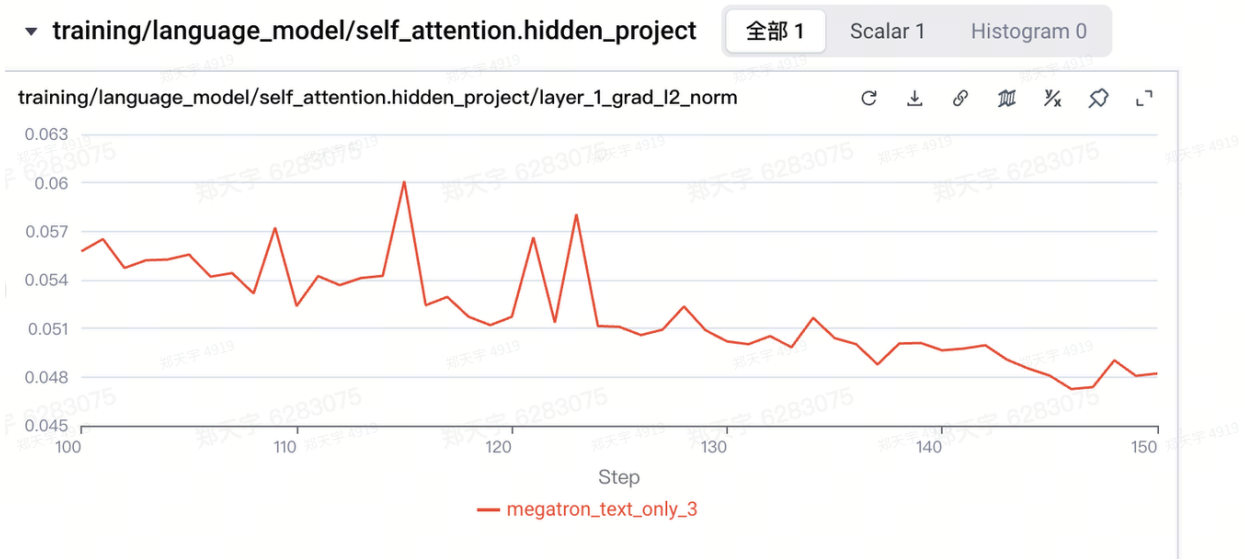
对齐两边的统计口径后（e-9差值）：

tracking:https://seed.byteintl.net/experiment/tracking/detail?Id=project_20260512_2fe9faed&selectedTrial=&tab=CHART&viewId=view_20260521_c999e814

2. grad_norm layer1差距较大验证



观察到grad_norm layer1差距和其他层表现完全不一样，因此怀疑是第一层有什么参数没对齐，查验后发现：



megatron在第一层多一个hidden_project，而fsdp没有？

进一步查验发现，两者挂载的位置不同，megatron挂载在shortcut_from层，而fsdp挂载在顶层，并且做了如下计算验证：

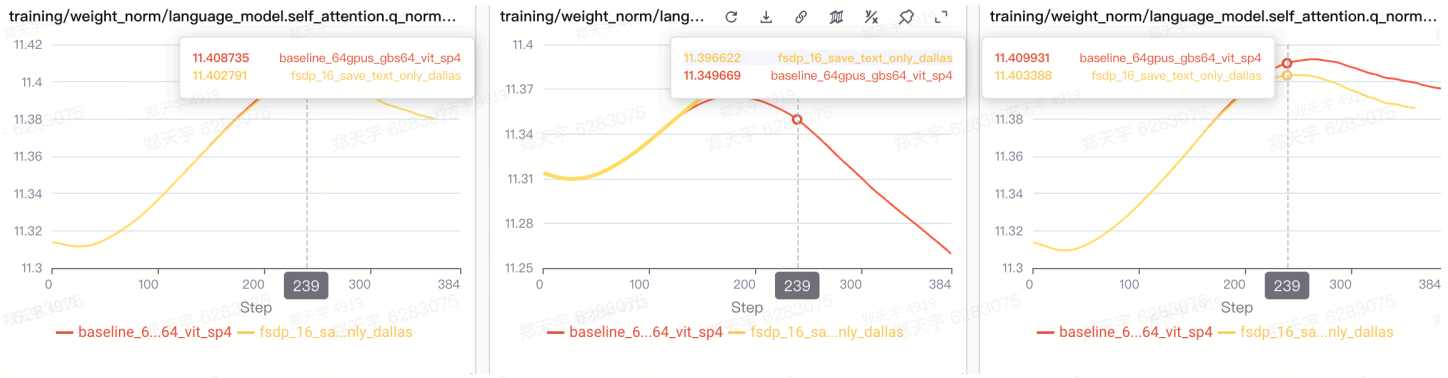
1️⃣fsdp_hidden_project_grad_norm = 0.054187

2️⃣fsdp_layer1_grad_norm = 0.05533

算在layer1，则fsdp_layer1_grad_norm = sqrt(1️⃣^2+2️⃣^2)=0.077444

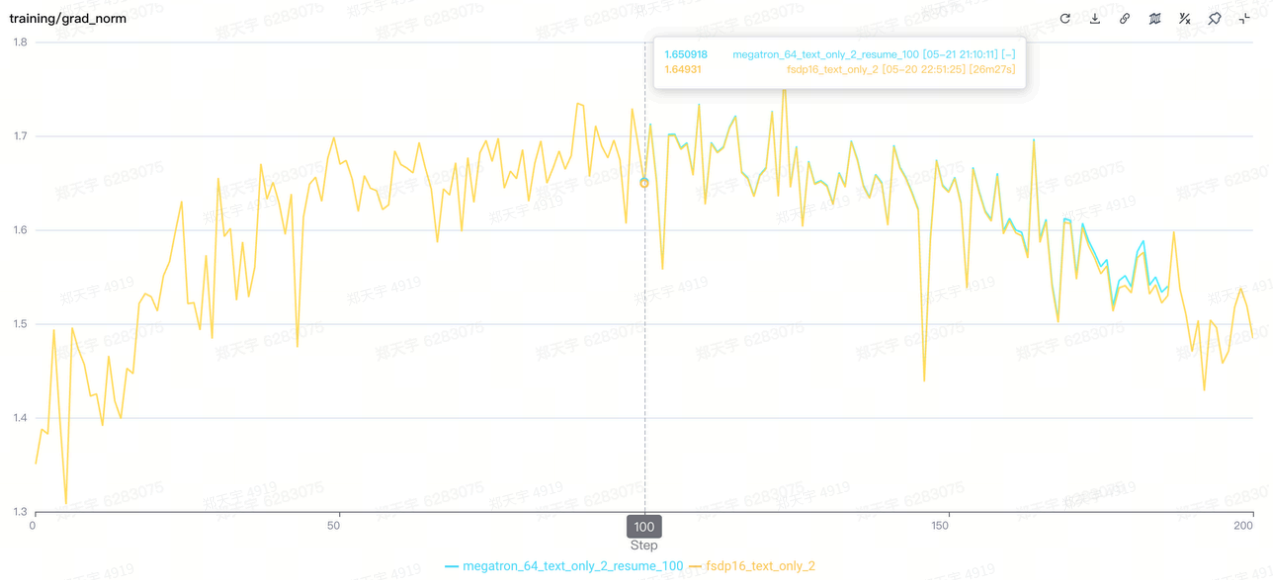
与0.078797在千分位差值，符合预期。

3. layer5的 self_attention.k/q_norm_swa （resume点bitewise对齐，后续参数差值逐渐增大，需要关注）

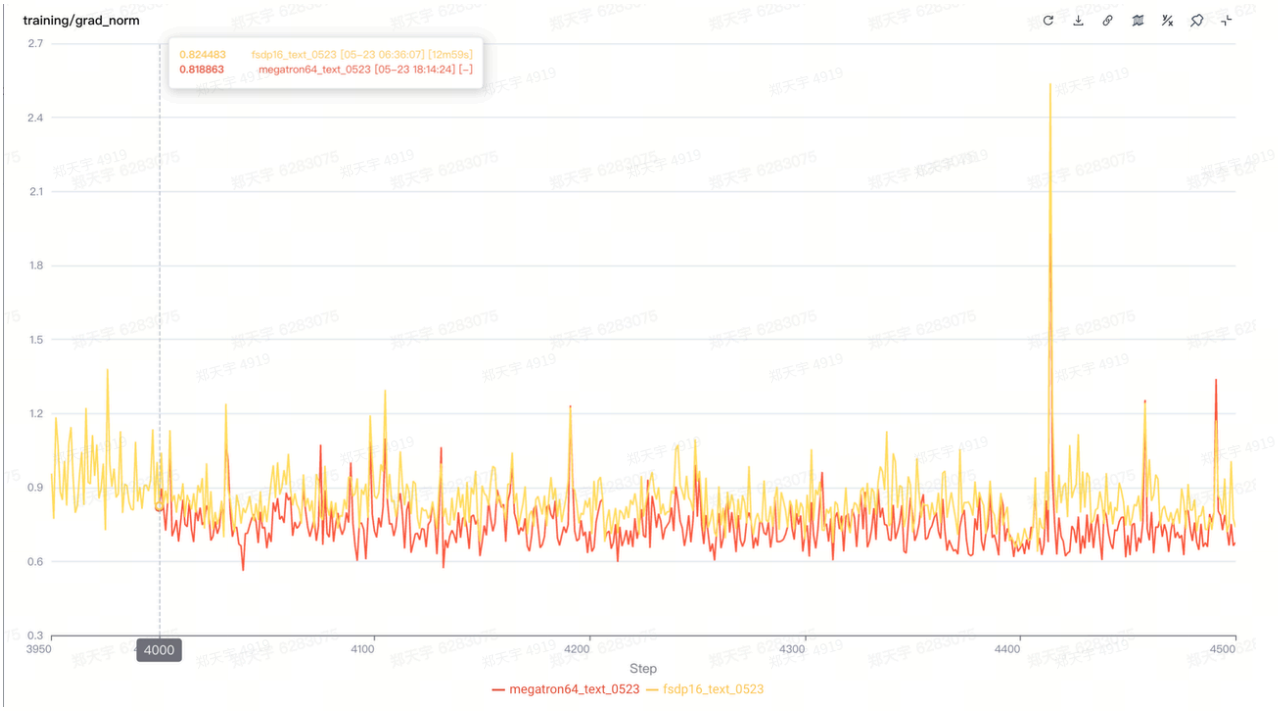


这个也为后续发现yoco部分实现有问题埋下伏笔，因为当时没深入了解算法那边期望的实现（后面会讲），只是验证了当前gb大算子中attention的实现，发现没有什么问题，因此暂时保留。

4. resume步数不同，观察到不同现象？



在100步左右resume，grad_norm对的比较齐，而在4000步对齐，如下：



grad_norm有较大波动，认为仍然存在一些不符合预期未对齐的配置。

5. 针对grad_norm进行更细项的分析
    - 首先分析了resume点 逐层grad_norm的差异，基本都在千分位，没有明显异常：

metric	FSDP	Megatron	M-F	相对差
total training/grad_norm	0.824483	0.818863	-0.005620	0.68%
layer_0	0.314511	0.309272	-0.005239	1.67%
layer_1	0.449927	0.444504	-0.005423	1.21%
layer_2	0.316729	0.313987	-0.002742	0.87%
layer_3	0.237212	0.238834	+0.001623	0.68%
layer_4	0.173472	0.186284	+0.012812	6.88%
layer_5	0.239373	0.240459	+0.001087	0.45%
layer_6	0.142806	0.146906	+0.004100	2.79%
layer_7	0.084439	0.086748	+0.002309	2.66%
layer_8	0.076746	0.077560	+0.000814	1.05%
layer_9	0.084349	0.084053	-0.000296	0.35%
layer_10	0.098138	0.097606	-0.000531	0.54%
layer_11	0.166265	0.172356	+0.006091	3.53%
layer_12	0.139393	0.142718	+0.003325	2.33%

    - 为了避免是两边每层grad_norm统计的参数范围不同，因此对每一层grad_norm个数和具体统计的param做了分析，和预期一样，只有第一层多一个：

层	结果
layer 0	对齐，都是 28 个
layer 1	不对齐，FSDP 28 个，Megatron 29 个
layer 2-12	全部对齐，都是 28 个

唯一逐层差异是 layer 1：

```
Megatron 多了:
self_attention.hidden_project.weight
```

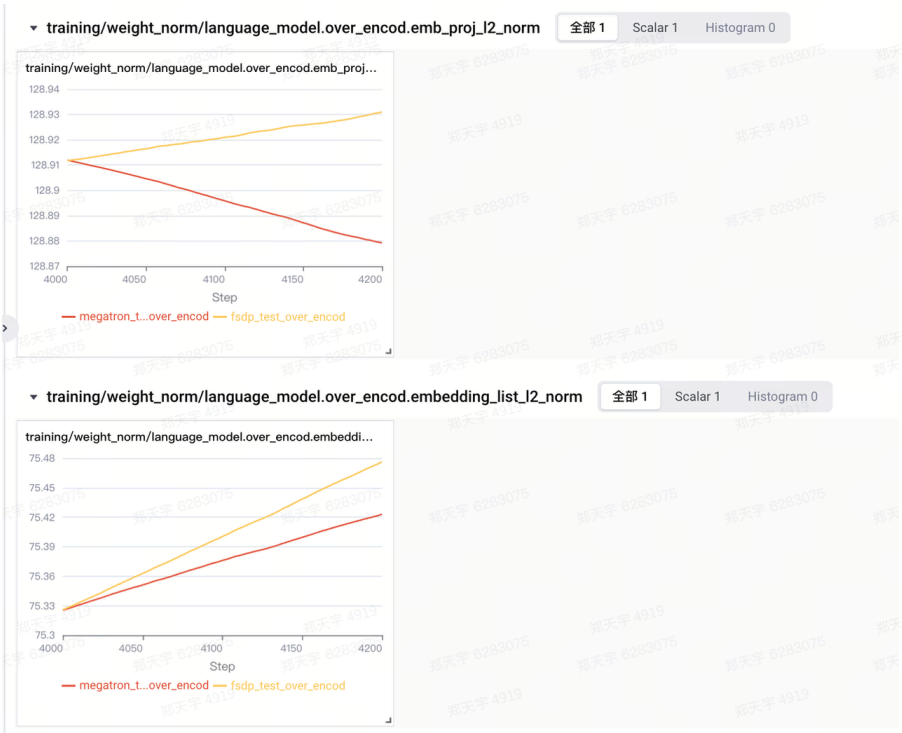
    - 针对更细项的grad_norm 分析：

`self_attntion_o_proj` 、 `swa_flash_qkv_proj` 到4001步，某些层如第5层，差别较大：

`h2.static_alpha`，某些层如第5层，差别较大：

因此这里担心是optimizer在加载ckpt的时候，有一些没对齐的动量，而当前omnistore也不支持resume后立即save，因此在功能上做了支持，将megatron resume后的ckpt和fsdp的ckpt 的optmizer相关的tensor比较它们的md5，除了megatron多出来的norm参数的动量，其它均bitewise对齐，符合预期。所以基本确定是optimizer的某些配置没对齐。

- 因为oe embedding在最上层，所以这里对齐了一下fsdp和megatron两边对oe embedding的统计，发现果然两边project更新完全是反方向：

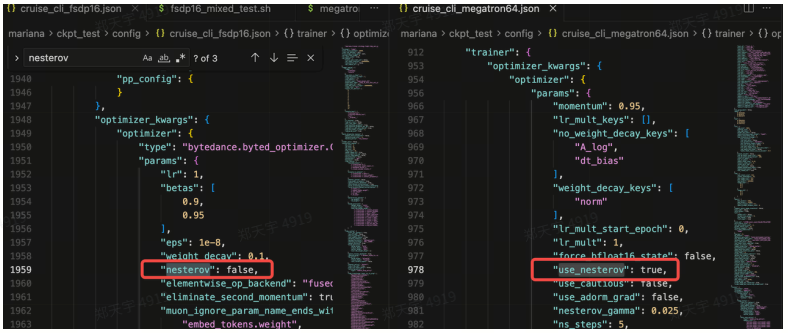


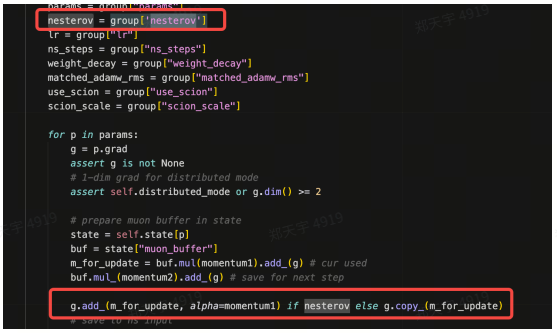
6. 详细对比optimizer的参数

发现两个参数没对齐：

a. Nesterov

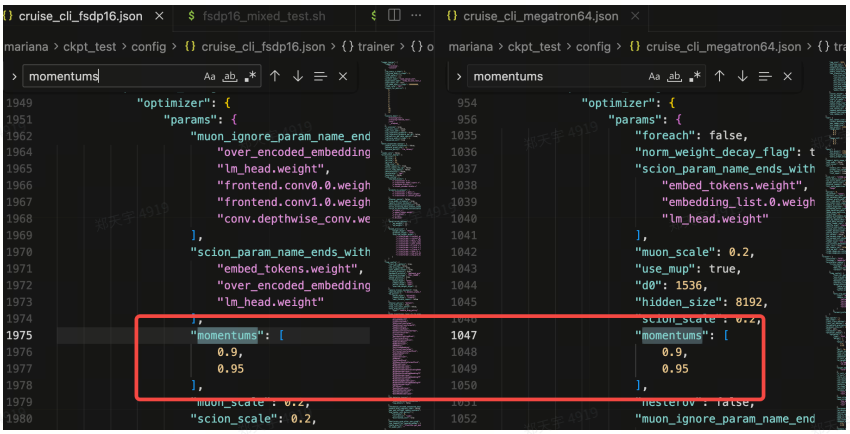
两边config名不一致，导致之前漏掉了

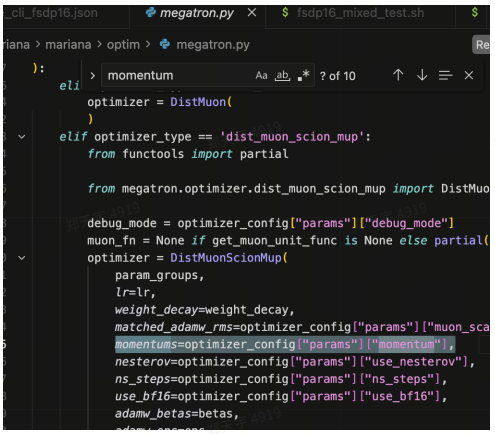


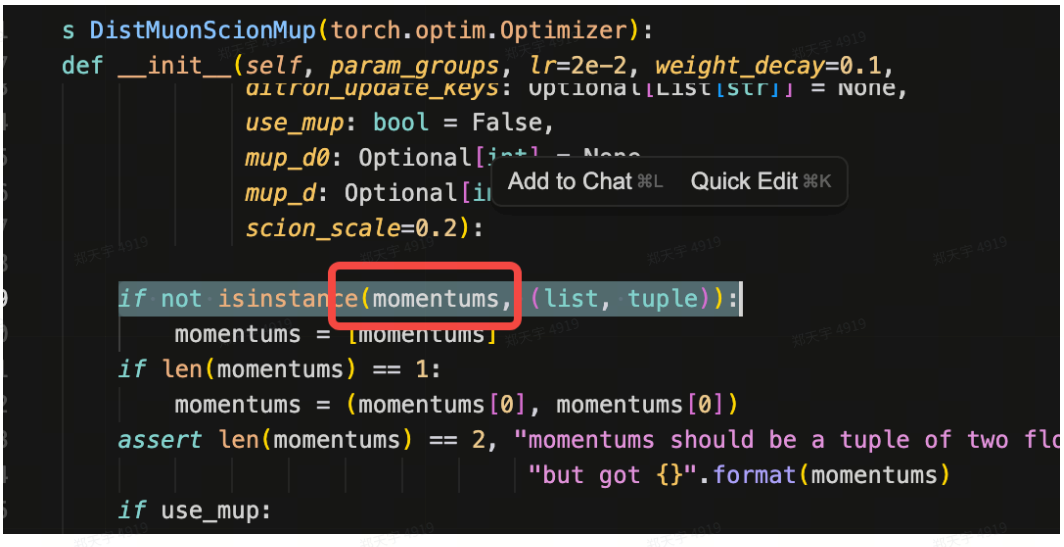


b. Momentums

megatron这边只能读取momentum字段，而不是momentums，跟fsdp不一致，所以传momentums一致是无效的，实际还是读的momentum的单值字段，发现实际DistMuonScionMup能接收momentums，但入口没传进来，所以在入口处做了适配，和fsdp对齐：







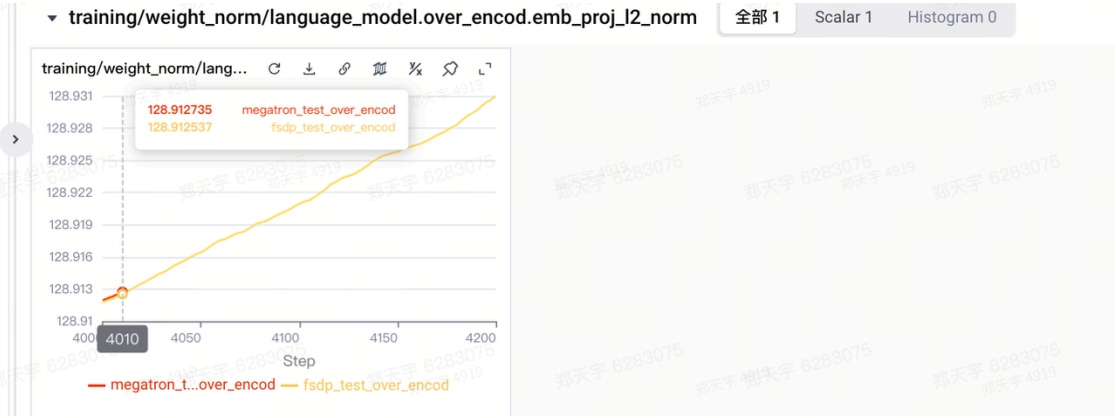
c. 同时记录了一个会影响后续非freeze的问题，关于optimizer的extra_param_groups：

```
"extra_param_groups": [
    {
        "param_regex": "visual_encoder",
        "use_mup": false,
        "muon_scale": 0.2,
        "nesterov": true,
        "momentums": [
            0.95,
            0.95
        ]
    },
    {
        "param_regex": "audio_encoder.(?!adapter.)",
        "use_mup": false,
        "muon_scale": 0.2,
        "nesterov": true,
        "momentums": [
            0.95,
            0.95
        ]
    }
```

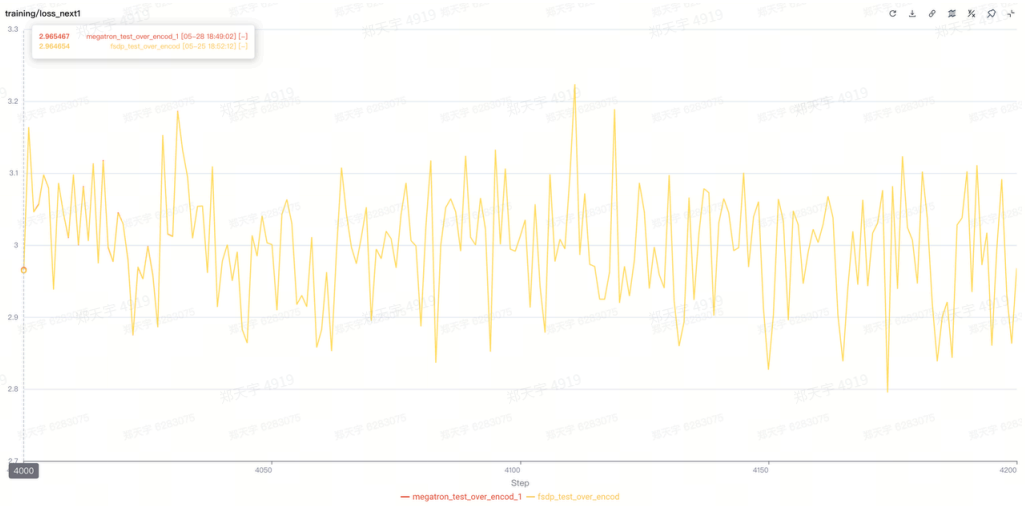

fsdp当前还有extra_param_groups来控制vit和audio与全局配置不同的optimizer更新的参数，而megatron是没有这些的，但是由于现在是freeze的，不会影响。这个也及时反馈给了 王曦 进行开发。

对齐后效果：

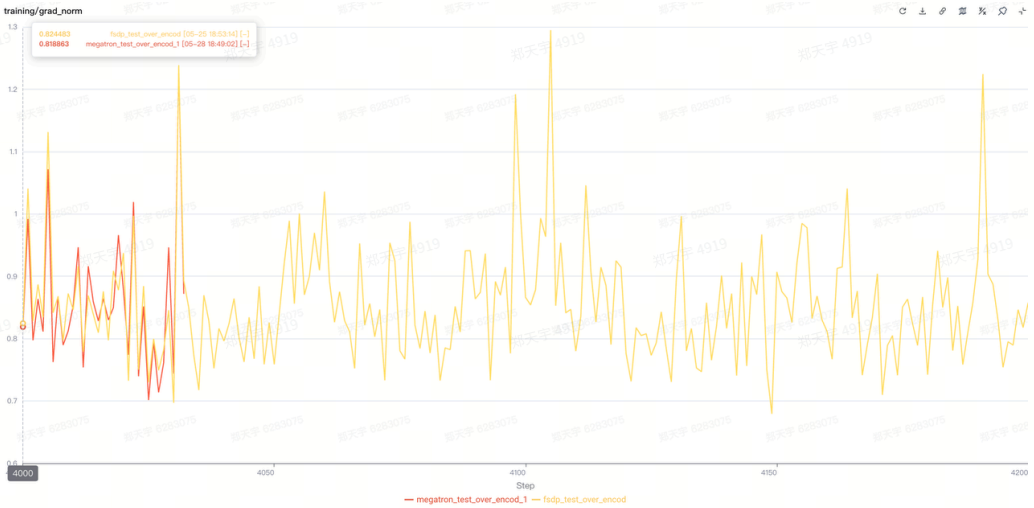
ov_emb_proj更新方向一致：



Loss:万分位误差



grad_norm 千分位误差，上下交错，基本符合预期。



5. 多模态对齐

1. Vrope源数据修复（历史代码原因）

在文本对齐的基础上，对齐了多模态部分的config，启动时发现问题，vrope部分报错：

这里存在一个表面原因和一个根本性原因：

a. **表面原因是**：表面看是skip_mm_dsp_in_data和refract_transform两个在用的时候没做好兼容。megatron开的skip_mm_dsp_in_data=True，原意是在数据侧按dsp切分时，不切多模态的数据，而当前又开了refract_transform，这个会把visual_position_ids这种元数据给切开，在做vrope的时候，先由vidpool选出了select的token，如果select的token是空列表，会fallback到不用vidpool，而fallback的路径认为，skip_mm_dsp_in_data开了True，则visual_position_ids就没切，不需要allgather，后续访问时就会越界报错。

b. **根本原因是**：这里开了vidpool且存在视觉token的时候，理论不应该fallback，因为一定会选出token，而fallback的原因有两个：

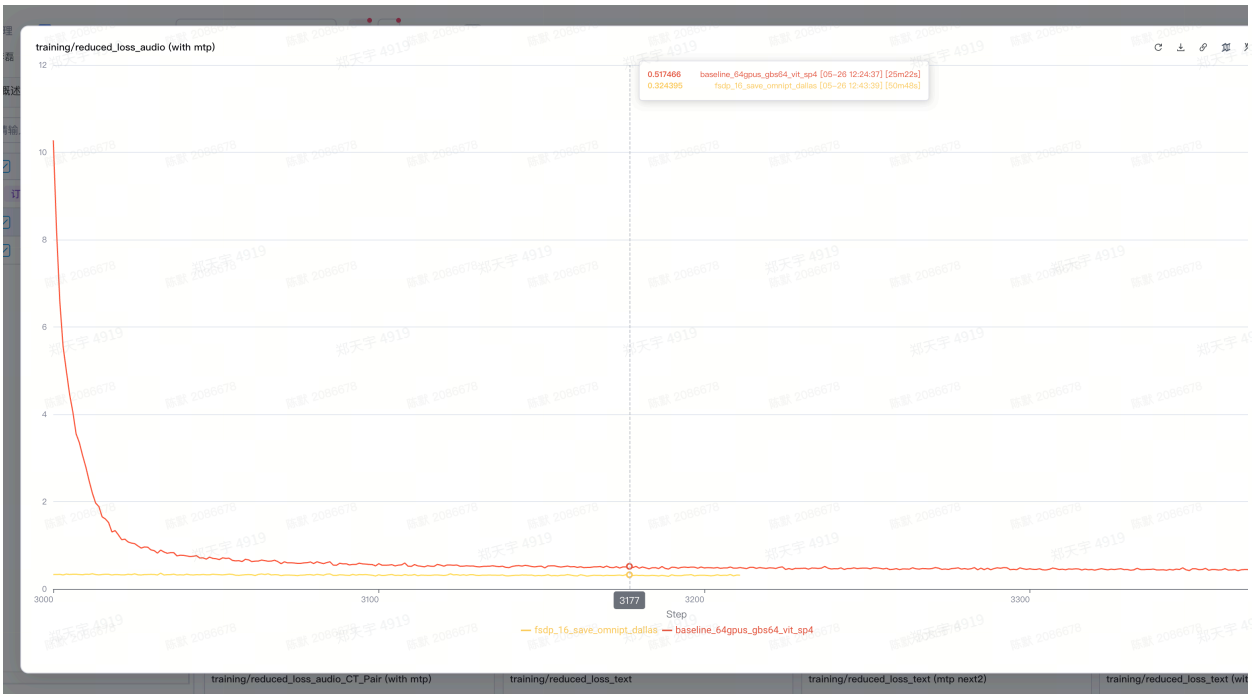
i. 旧代码的adaptor(audio/vision)会走同一个执行逻辑，即如果batch_vrope_select_indices存在，则将vrope_select_indices拆分给每个batch，如果不存在，就设置为None，而audio比vision的adaptor后执行，audio本身就没有batch_vrope_select_indices，就会设置成None，把前面vit的产出给覆盖掉。

ii. 即使修复了1，这里还存在一个问题，旧代码把vrope_select_indices和embeddings一样，从pp0传进去，在encoder这里有一个特殊的处理，即将pp rank用来做dp并行，所有每个pp_rank拿到了部分的vrope_select_indices，旧代码调用 ppn_to_n，把所有vrope_select_indices都丢给pp0，但却没有把它在pp_rank间传递，而每一层block都是要做vrope的，导致后续拿到的全是空列表，都走了fallback，也会导致fsdp和megatron行为无法对齐。

陈默 做了修复

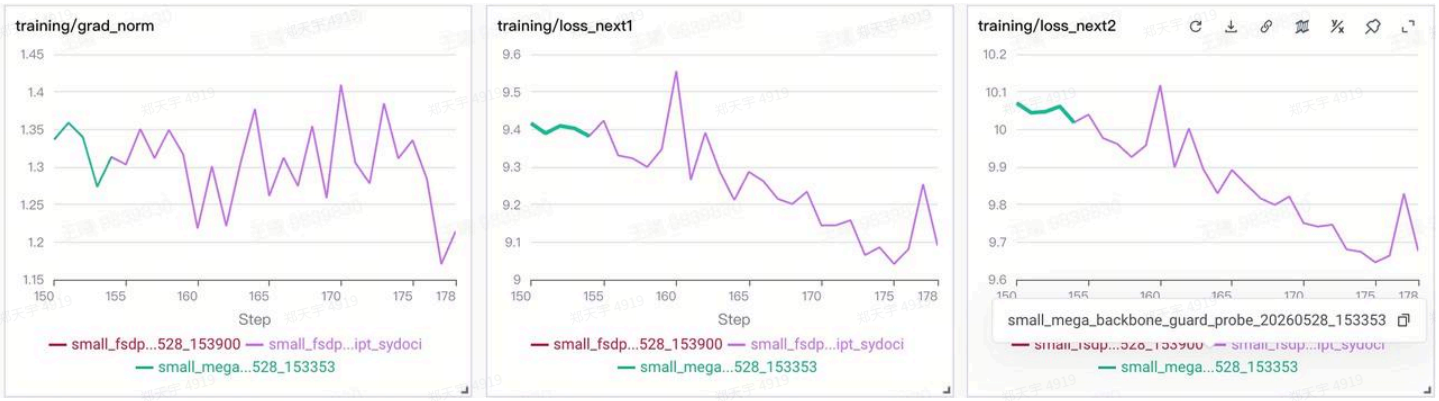
2.Nope 问题修复

上述问题修复后，发现此时多模态loss仍然存在差异，而分析具体细项发现，最大差异在reduced_loss_audio，这里megatron完全像是随机初始化的权重。



Dump weight后，发现有几层audio的参数对不上， 王曦 最终定位到，是noop映射的问题，因为megatron存在noop，而fsdp不存在，所以在做llm 权重加载的时候，omnistore需要做一个映射，但是这里存在一个bug，noop映射，误伤了audio的backbone。audio加载的权重出现了偏移，为什么只误伤了audio？因为这里有一个正则匹配，只会匹配Layers.N，刚好audio的backbone命名能匹配上，而vit不是这个格式。

修复后loss，grad_norm符合预期：

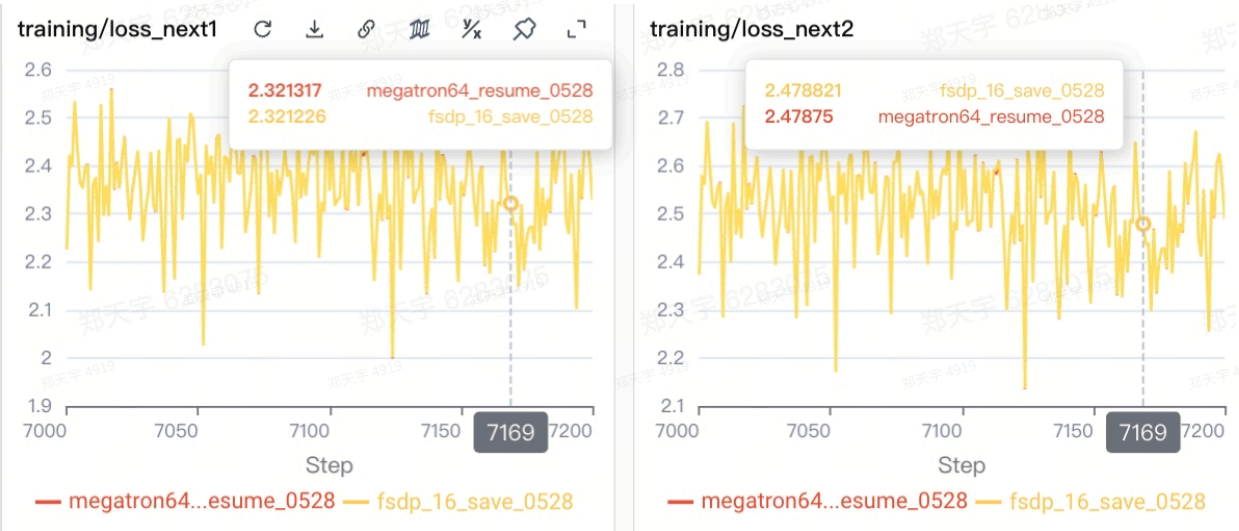


在fsdp64+megatron512卡测试，发现loss和gard_norm符合预期（这里从grad_norm和loss看，确实没看出异常，但实际还有问题）：

- 实验：https://seed.byteintl.net/experiment/tracking/detail?Id=project_20260512_2fe9faed&selectedTrial=&tab=CHART&viewId=view_20260528_10caa99e

grad_norm:

loss:



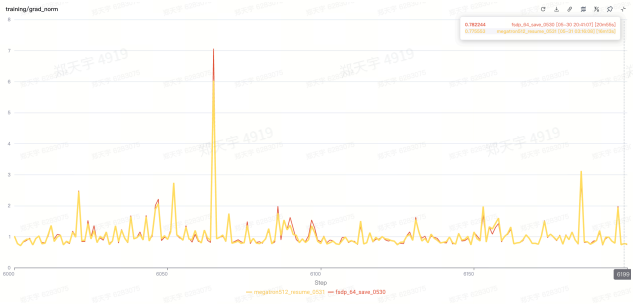
6. 二次对齐线上fsdp任务(5.29)

fsdp线上任务又有一些更新，做了二次对齐

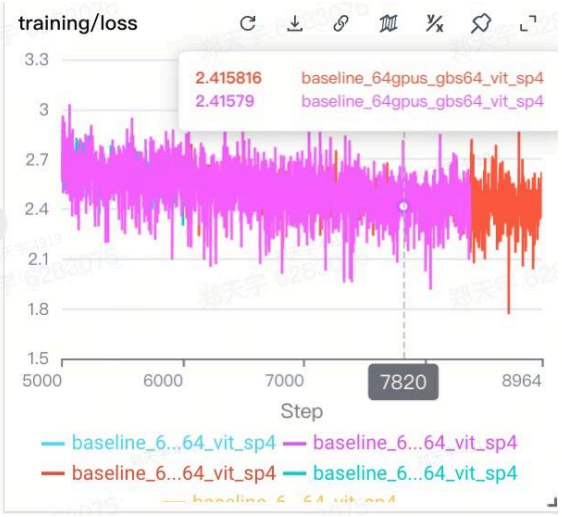
	更新前	更新后
镜像	base/seed.unify(1.0.0.6)	aliyun-va-hub.byted.org/reckon/gb_debian12.py312.cu131.torch291:1.0.0.15
FA3 安装方式不同（主要是适配镜像改变）	mkdir /tmp/install-fa3 pushd /tmp/install-fa3 rm -rf /tmp/install-fa3/* wget https://luban-source.byted.org/repository/scm/aarch64/seed.speech.flash_attention_v3_1.0.0.240.tar.gz tar zxvf *.tar.gz pip3 install *.whl --force-reinstall --no-deps popd	mkdir -p /tmp/install-fa3 pushd /tmp/install-fa3 rm -rf /tmp/install-fa3/* wget https://luban-source.byted.org/repository/scm/aarch64/**seed.speech.flash_attention_v3_1**.0.0.240.tar.gz tar zxvf **seed.speech.flash_attention_v3_1**.0.0.240.tar.gz # 注意: .240 里的 flash_attn-2.8.3-cp311 wheel 在 py3.12 上不兼容，跳过它。 # flash_attn_3 是 cp39-abi3 (跨 Python 兼容), flash_attn_4 是 py3-none-any (纯 Python), 这两个可装。
多了视频依赖路径	无	export LD_LIBRARY_PATH=/usr/local/decord_dependency/ffmpeg/lib:${LD_LIBRARY_PATH}
并行规模差异	sp_size=8	sp_size=4
dataloader参数配置	--data.train_num_workers=2 --data.prefetch_factor=4 --data.split_sp_inputs=false --data.vrope_image_pos_centering=true --data.vrope_image_pos_centering_intergerized=true	--data.train_num_workers=4 --data.prefetch_factor=2 --data.cpu_prefetch.pin_memory=false
datasets_meta_path		

		metainfo_32k_patch16_pool16_navidv2.yaml

扩卡做 fsdp64->megatron512的对齐验证：



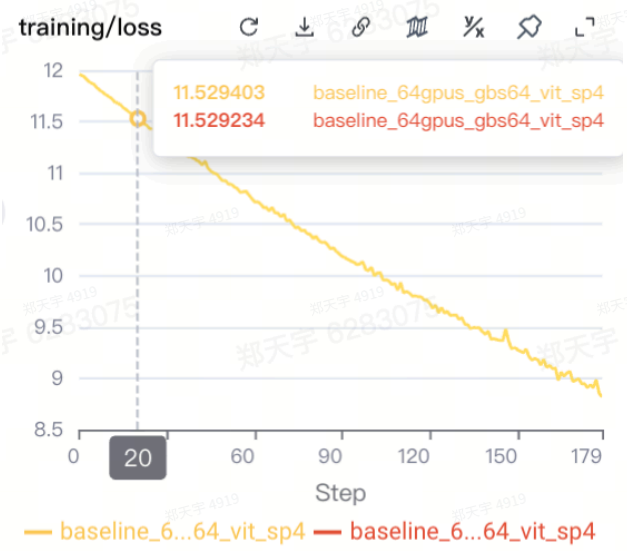
grad_norm和loss比较符合预期。但是为了增加容错，我们开启了megatron 的 auto resume，防止机器原因导致实验挂掉，而这时候发现megatron resume自己无法对齐了！



7. Megatron resume 自己的验证

- omnipt的save resume 测试，无法bitewise对齐：

- 入口不变，数据换成text，无法bitewise对齐：：



这里尝试和之前能对齐的config做消融，来定位具体是什么原因，中间做了大量尝试，最终定位到：当dsp=4时，无法对齐，而dsp=8时却能对齐了，这里比较诡异？

于是尝试dump megatron resume 前后，vpp chunk每层参数的输出来做bitewise，具体定位哪一层参数出现的无法对齐，结果发现，第7层开始(idx从0开始计)，两边就无法对齐了。

于是又尝试dump 第7层的所有输入，发现swa_v的md5无法对齐：

- `forward_step`：32 条齐全，20 条一致，12 条 diff。
- 差异仍然是 `pp_rank=0, vpp_rank=2/model_chunk_id=2` 的 `forward_step_output` 先不一致，然后传到 `vpp_rank=3` 的 input/output。
- BigOp chunk2 内部按 microbatch 对齐后，最早 diff 是：`phase=v1gb_swa_value` 四个 microbatch 都在这里首次不一致。
- 在它之前，`forward_input`、`ln1_output`、`attention_input`、`query`、`swa_query`、`key`、`value`、`swa_key` 都是一致的。


陈默

李想

朱家成

郑天宇 一起讨论后定位big op的实现这里存在两个问题：

1. 缩卡后的yoco与num_head配置存在问题，且yoco 共享 kv 目前只适配了 生产层与消费层kv heads相等的情况：

代码块

```
model.network.full_kv_yoco_layer=[2,5]
model.network.flash_attn_num_kvheads=[8,4,4,4,4,4,4,8,4,4,4,4,4,4]
```

这里消费层大于生产层，理论不应该出现这样的config，而dsp=8能对齐，原因在于，消费层的kv heads=4，此时小于dsp=8，因此会有一个最小为1的兜底，来复制生产层的4个heads，刚好每个dsp rank 1组KV，分配消费层，恰好分对。

而dsp=4的时候，不会兜底，每个 dsp rank 分到一个kv heads，并且这里full attention 和 swa 的 q，k，v做了一个fuse 写在同一块连续buffer。当消费层大于生产层的时候，消费层要读2个k v

cache的数据，此时就会offset错位往后读，读到异常的数据。那为什么只有swa_v没对齐，我也很好奇，所以去具体看了一下，原因如下：

此时被dsp切分后，正常的FA 的 heads是1个，而SWA的heads是2个，而这里在实现KV buffer的时候，按照最大head 数来初始化，所以会有8个位置，但是生产层只往填了6个buffer，而消费层在取数据的时候，会取8个数据，所以其实resume 前后，消费层取的前6个buffer的数据，虽然取错了，但是都是固定的；但是最后两个位置没填充数据，resume前后这里的数据不一致，导致我们发现了SWA_V没对齐。

2. 当前Ulysses的fuse实现存在问题

这里以一个正常的config举例，producer=8，consumer=4，dsp=8的例子：

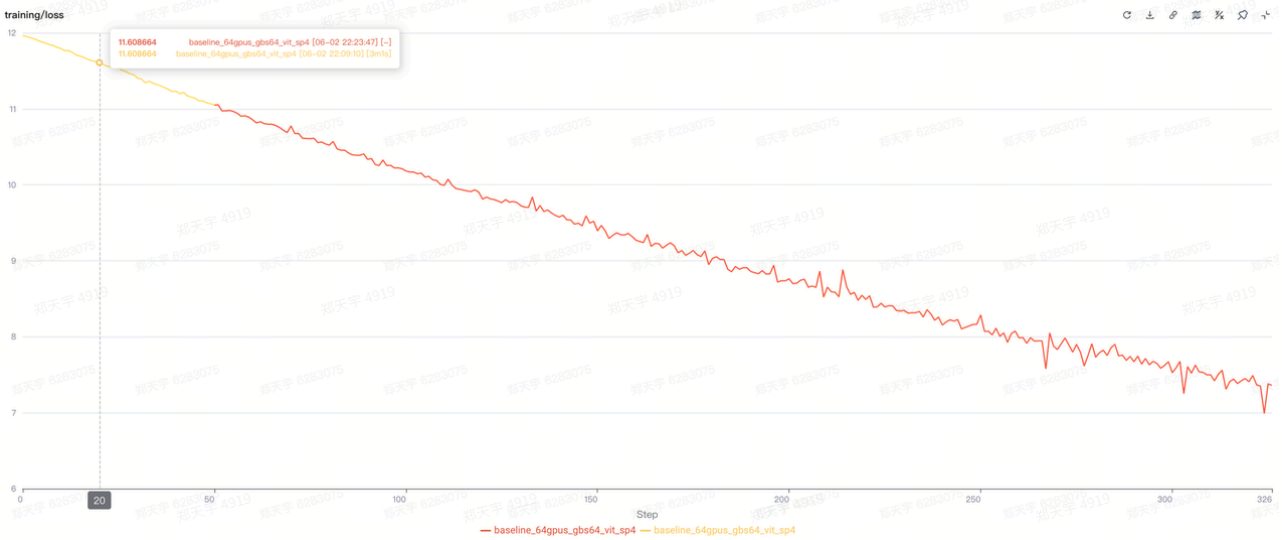
首先正常流程是：

先all_to_all，从seq_len维度转到head维度切分，这时候，dsp=8，消费层的kv heads=4，按算法要求只取生产层构造的前4个 kv heads，然后做repeat，到8个，再做dsp切分，每个dsp rank1个，如左图所示；

而现在，这里在做attention计算的时候，为了做通信和计算的overlap，把all_to_all和gemm绑定，直接跳过了slice的阶段，这就导致，在切dsp的时候，消费层虽然是4个 kv heads，但他实际按照生产层的8个 kv heads来取，并做dsp的切分，就导致每个dsp rank其实是分别拿到了各不相同的8个kv，而不是做了repeat的，因此这里也存在问题。

这里也跟之前观察到swa相关指标差距越来越大的现象呼应上了。

修复后，resume auto 即可bitewise对齐：



8. 为了更准确复现线上的实验，新增数据集

新增5类混合数据	text、image、video、video-audio、audio

最新数据实验发现，loss_next2在百分位误差，不符合预期：

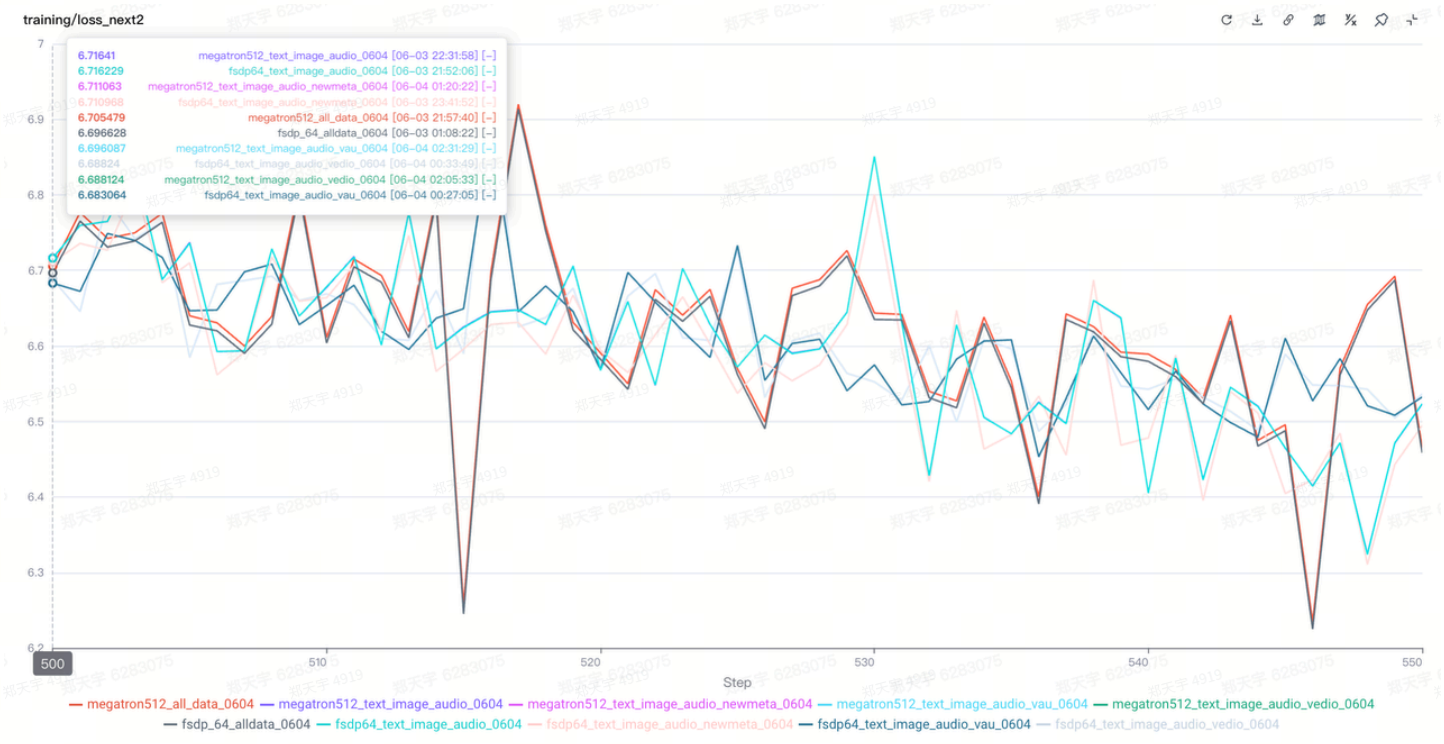
于是尝试做数据集类型的消融，来定位原因：

消融项目	不能对齐配置	fsdp	megatron	说明
消融数据				

	1. train_path 2. data.data_sets_meta_path	无video： healthy_text_vit_usm_mix.yaml 有video： healthy_text_vit_vedio_usm_mix	无video： healthy_text_vit_usm_mix.yaml 有video： healthy_text_vit_vedio_usm_mix	用最新的fsdp和megatron的配置和分支测： 1. 验证对齐 (text + image + audio)+旧datasets_meta_path 2. 验证对齐 (text + image + audio)+新datasets_meta_path 3. 验证不对齐 (text + image + audio+video+vau)+新datasets_meta_path 4. 补充验证 (text + image + audio+video)+新datasets_meta_path 5. 补充验证 (text + image + audio+vau)+新datasets_meta_path

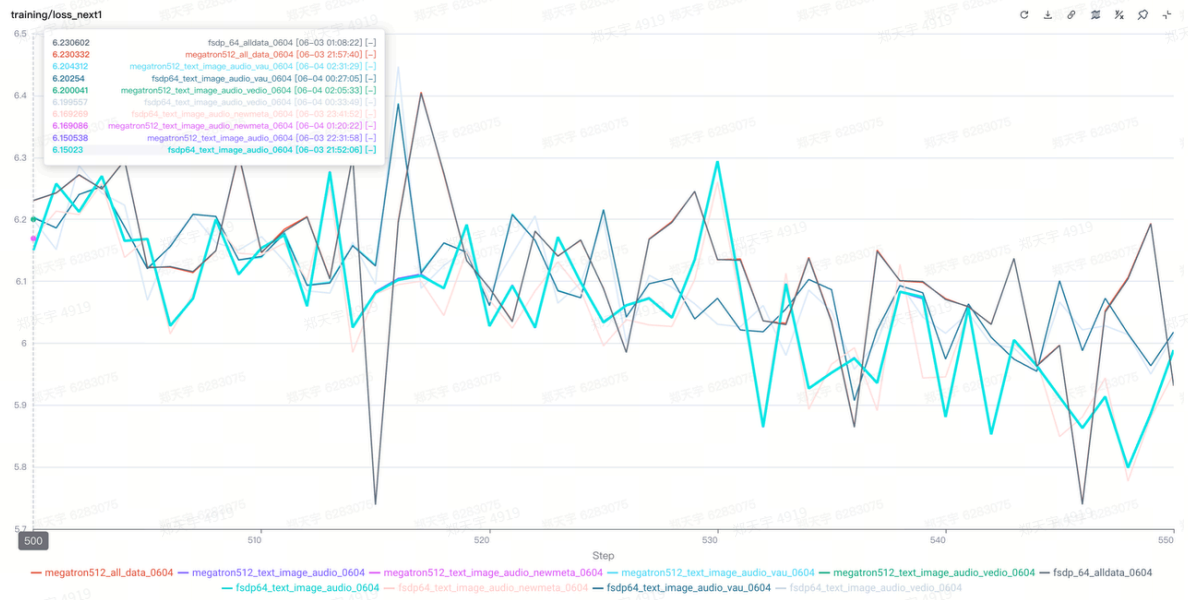
消融实验trial：https://seed.byteintl.net/experiment/tracking/detail?Id=project_20260512_2fe9faed&selectedTrial=&tab=CHART&viewId=view_20260603_ae97d43c

最终定位，是vau数据引起的误差：



根据 陆琦 提供的线索，最终定位是对于extra_head的mtp，mask两边处理不一致，fsdp会将全双工数据给mask掉，即不算loss_next2，而megatron这边没有这部分处理逻辑，而本次实验使用的vau就是全双工数据。修复后，loss_next2符合预期。

9. Audio encoder 和 Vision encoder的bitewise对齐：



同时发现消融实验中，loss_next1体现出，其它数据组合都在万分位差距，只有包含vau的数据在千分位差距，目前认为vau中还有未对齐的部分，因此决定先对齐audio和vision两部分，来判断是否是audio和vision部分的问题，如果不是，则能确定是llm部分与vision和audio相关部分的问题，从而缩小范围，这里分两部分对齐：

a. 数据对齐：dump input_ids、loss_mask、shift_labels

b. audio和vision encoder bitewise对齐

考虑到缩小规模在单机4卡上定位，要比大规模机器定位节约成本，且效率高很多，因此整理了一个基本对齐fsdp/megatron线上任务的4卡debug trial。

1. 4卡debug trial

fsdp：https://seed.byteintl.net/development/instance/jobs/3f5118b4a87d2ce4?tabState=run_config&trialId=301143484

megatron：M14_60B_Precision_Alignment_megatron_resume(do not change!)_copy_2

bash脚本：4卡 fsdp megatron 对齐脚本

2. 数据对齐

首先为了方便对齐，以及为了后续encoder输出的bitewise对齐，我们首先要确保每个rank上处理的数据一致，因为fsdp送入数据是loop的，而megatron是一整个batch数据送入的，并且在多

模态的encoder部分，megatron有个特殊的处理，它的encoder的参数是replicate的，而且会在pp rank上做dp，因此我们将参数设置成：

megatron：gbs=4，pp=2，dsp=2，dp=1

fsdp：gbs=4，dp=2，dsp=2

这样4个rank分别处理一个micro_batch，同时这里针对audio config、params、embeds、以及shift_laels、loss_mask、input_ids做了对比：

对比项	Megatron 字段	FSDP 字段	状态	说明
Audio Config	`init_config`	同左	基本一致	2 处格式/默认值差异
Audio Params	`params`	同左	一致	226/226 match
Audio Embeds	`embeds` + `masks`	`embeds`	不一致	dump 粒度不同，本地分片数值不 match
shift_labels	`raw_shift_labels`	`shift_labels`	一致	0 diff
loss_mask	`loss_mask`	`loss_mask`	一致	同一 batch
input_ids	`input_ids`	同左	一致	同一 batch

参考文档： Dump audio-encoder/ celoss_label（fsdp vs megatron）、Fsdp vs megatron（对比全双工数据）

发现此时audio encoder的输出是无法bitewise对齐的。

3. Audio/Vision encoder bitewise对齐

根据 刘波 的文档分析，发现这里主要存在三个问题：

a. audio的balance，两边控制的参数不一致，导致其实megatron没开balance，而fsdp开了。

b. megatron和fsdp虽然都加载一份audio和vision的ckpt，但是这里不包含adapter部分(我们freeze的时候也不包含这两部分)，所以这里在做bitewise对齐的时候，因为没有用megatron resume fsdp，会导致这部分参数，两边各自随机初始化，而导致无法对齐。

参考文档： 4卡 fsdp megatron 对齐任务

修改后，发现vit和audio的encoder完全能够bitewise对齐：

对比项	FSDP 文件	Megatron 文件	Shape
Audio mb1 concat	rmpad_audio_embeds_dp0_dsp0_pp0.pt + rmpad_audio_embeds_dp0_dsp1_pp0.pt	rmpad_audio_embeds_dp0_dsp0_pp0.pt + rmpad_audio_embeds_dp0_dsp1_pp0.pt	(1304, 32768)
Audio mb2 concat	rmpad_audio_embeds_dp1_dsp0_pp0.pt + rmpad_audio_embeds_dp1_dsp1_pp0.pt	rmpad_audio_embeds_dp0_dsp0_pp1.pt + rmpad_audio_embeds_dp0_dsp1_pp1.pt	(726, 32768)
Vision mb1 dsp0	rmpad_vision_embeds_dp0_dsp0_pp0.pt	rmpad_vision_embeds_dp0_dsp0_pp0.pt	(3306, 32768)
Vision mb1 dsp1	rmpad_vision_embeds_dp0_dsp1_pp0.pt	rmpad_vision_embeds_dp0_dsp1_pp0.pt	(1980, 32768)
Vision mb2 dsp0	rmpad_vision_embeds_dp1_dsp0_pp0.pt	rmpad_vision_embeds_dp0_dsp0_pp1.pt	(3549, 32768)
Vision mb2 dsp1	rmpad_vision_embeds_dp1_dsp1_pp0.pt	rmpad_vision_embeds_dp0_dsp1_pp1.pt	(3146, 32768)
ViT vit_b0	fsdp_vit_b0_local_rank{0..3}_rank{0..3}_world_size4.pt	megatron_vit_b0_local_rank{0..3}_rank{0..3}_world_size4.pt	per-rank same
ViT vit_ve	fsdp_vit_ve_local_rank{0..3}_rank{0..3}_world_size4.pt	megatron_vit_ve_local_rank{0..3}_rank{0..3}_world_size4.pt	per-rank same

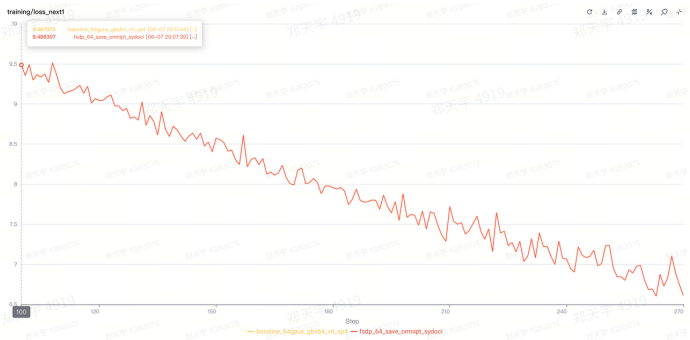
唯一差异：

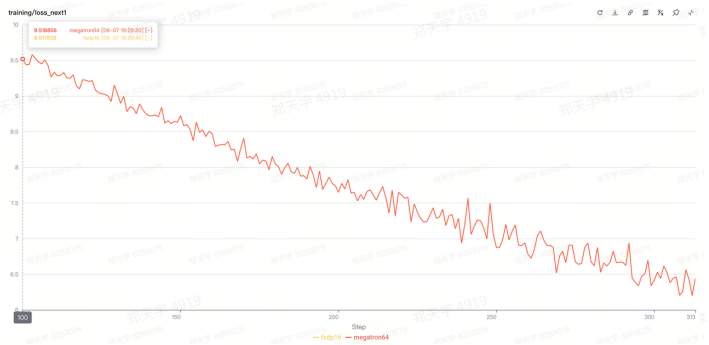
	dp0 段数	dsp0–6	dsp7
原始数据	21 段	—	—
Megatron slice_dsp_multimodal	n=ceil(21/8)=3	各 3 段（覆盖 0–20）	[20:21] → zero padding
FSDP slice_image_audio_in_batches_for_dsp	cp=3	各 3 段（覆盖 0–20）	start=min(21,20)=20 → [20:21] 复制第 20 段

AUDIO + VIT encoder 全链路（frontend→backbone→adapter/projector→rmpad）两边逐 bit 对齐。唯一差异是 FSDP 数据预处理切片器对空 DSP shard 复制最后一段（Megatron 留空），该复制段在 gather 后被截断，不影响 loss。

4. 在fsdp16+megatron64复现encoder bitewise对齐，同时对比loss差异

左边为原始配置，右边为从4卡配置扩充到16+64配置的实验，此时已经对齐了vision和audio两部分的encoder，但是loss差异仍然没变，还在千分位的差距：





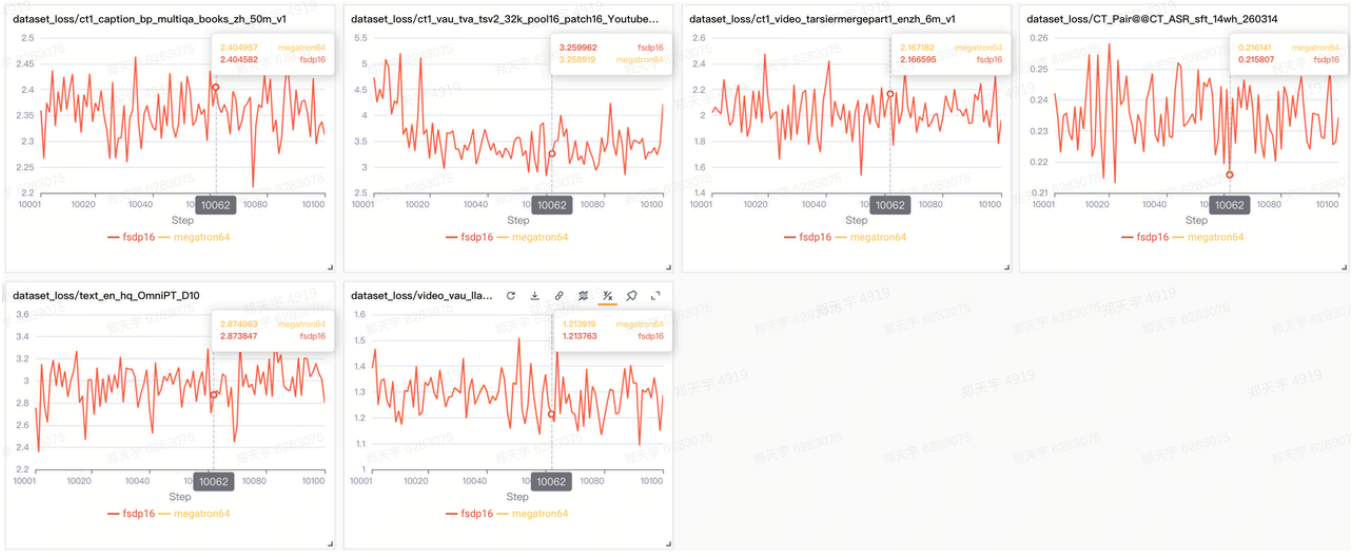
stage	exact
`vit_pe` / `vit_b0` / `vit_ve` / `vit_out`	100% ✅
`audio_fe` / `audio_bb` / `audio_out`	100% ✅
rmpad audio	100% (6734/6734) ✅
rmpad vision	100% (47850/47850) ✅

参考文档： fsdp16->megatron64(对齐)

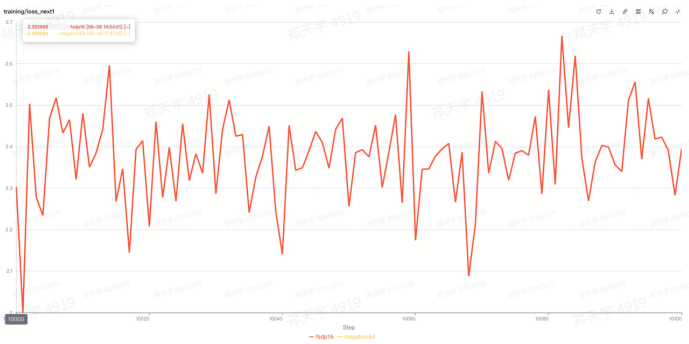
10. llm部分的终极定位

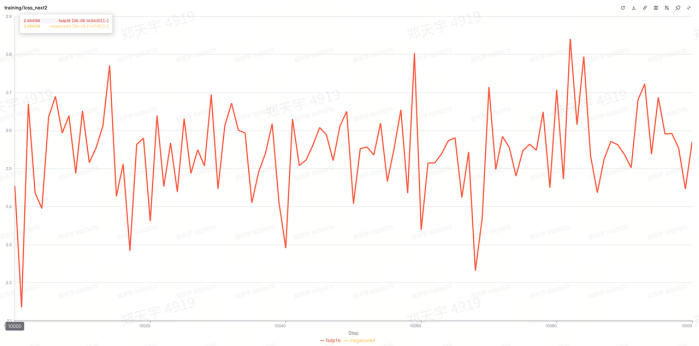
因为上面我们排除了audio/vision encoder的影响，并且 李磊 李磊 发现vit数据部分的loss其实也存在异常，那可能的影响就在llm部分 与vit相关的部分，最终 刘波 刘波 定位到是vrope加在了swa 和full attention上，而实际只需要加在swa 上。最终对齐效果如下：

DataSet_loss： 10-4 位diff



Loss next1/next2 : 10-5 位diff





最终保证m14.2模型在万卡(12288)成功上线训练。

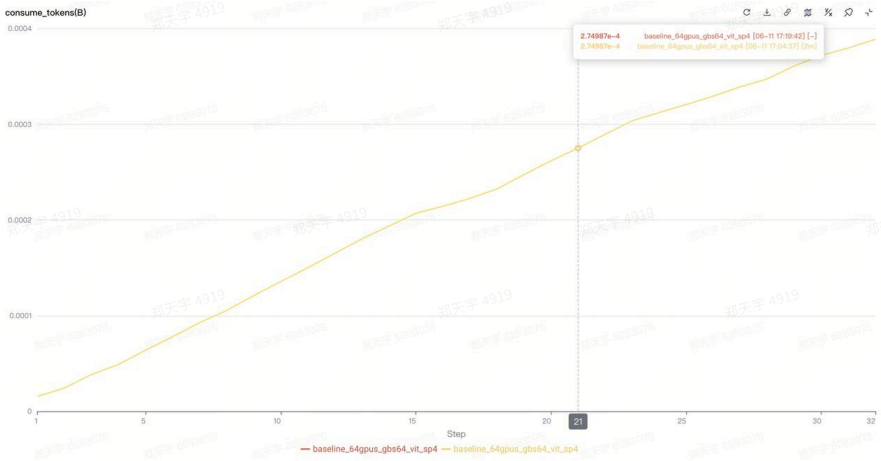
11. consume_tokens(B)对齐

上线后发现，consume_tokens在resume后存在两个不对齐的现象：

a. megatron resume自己无法对齐，每次从0开始计数

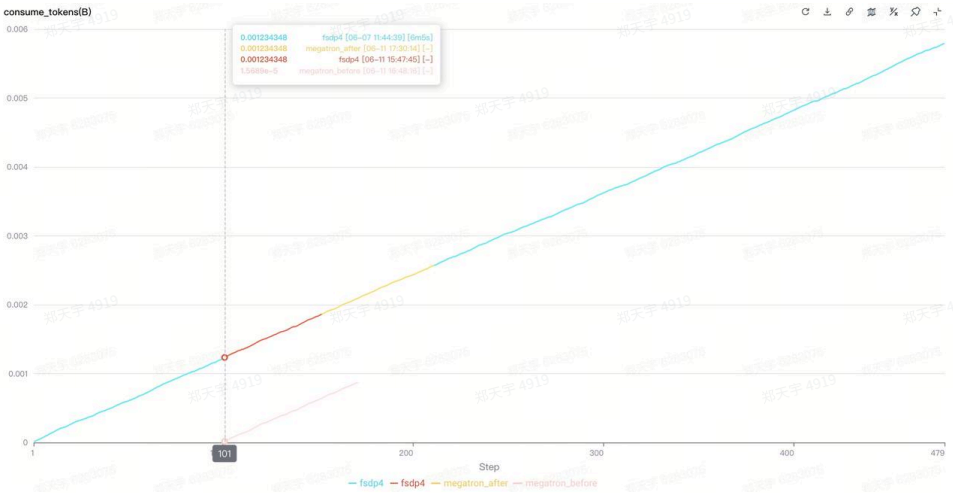
b. Megatron resume fsdp无法对齐

a问题最终查到，该指标主要是resume_logger_state来控制，之前精度对齐时，未检查该指标，因此没开这个flag，



b问题最终查到：

问题查找与修复：该指标未在 megatron omnipt resume fsdp时做兼容，主要问题在于，logger_state在两个框架下保存的位置不一样，fsdp默认保存在dp0dsp0，而megatron保存在pp-1所在的group，对应的dp0，dsp0；导致megatron resume的时候，发现对应rank上的这些需要的数据都是0，导致未加载上，因此加一个broadcast即可。



恢复线上数据：同时对于线上异常数据，通过读取fsdp的ckpt中的consume token相关的数据，手动hack代码来做累计，从而恢复数据。

mr：
https://code.byted.org/seed/mariana/commit/f09421cce18dff83b4247c983a35bf43016af995

12. 60B 数据 diff

排查fsdp和megatron 消费的数据突然对不齐了：

- **audio transform不同：**发现是一个mr引入的,主要原因是这个mr引入了对audio脏数据过滤逻辑，megatron加了这个逻辑，而fsdp分支并没有加这个逻辑，因此导致消费数据无法对齐：

  Revert MR :https://code.byted.org/seed/mariana/merge_requests/16213?dv_filepath=tasks%2Fseed2%2Fconfigs%2Fdata%2Fspeech%2Fomnipt_m14_data.yaml

  Revert 后，avg_effective_len对齐。

  tracking：https://seed.byteintl.net/experiment/tracking/detail?Id=project_20260509_c3108233&selectedTrial=&tab=CHART&viewId=view_20260612_d3052f3e

贾伟 曹率帅 王晨渊 杨世蛟 通过dry run发现：

- **vlm transform不同：**视频多帧尺寸不一致的样本会transofrm fial扔掉，但触发概率低，不是本次diff原因
- **Max sample len控制逻辑：**当前线上用的是datamodule v1，而v1会设置MARIANA_MAX_LLM_LEN，即用max_sample_len来限制单个sample的最大截断长度，而megatron omnipt的train_megatron没有设置这个东西，导致fallback到了max_seq_len，线上max_seq_len=49152,max_sample_len=32768,和fsdp行为不一致。当sample_len超过32768，fsdp会截断，而megatron不会。

由于之前对齐始终用的data module v2与fsdp对齐，且max_sample_len和max_seq_len保持一致，因此未发现后面两个问题。

13.线上任务Category reduced_loss突然消失

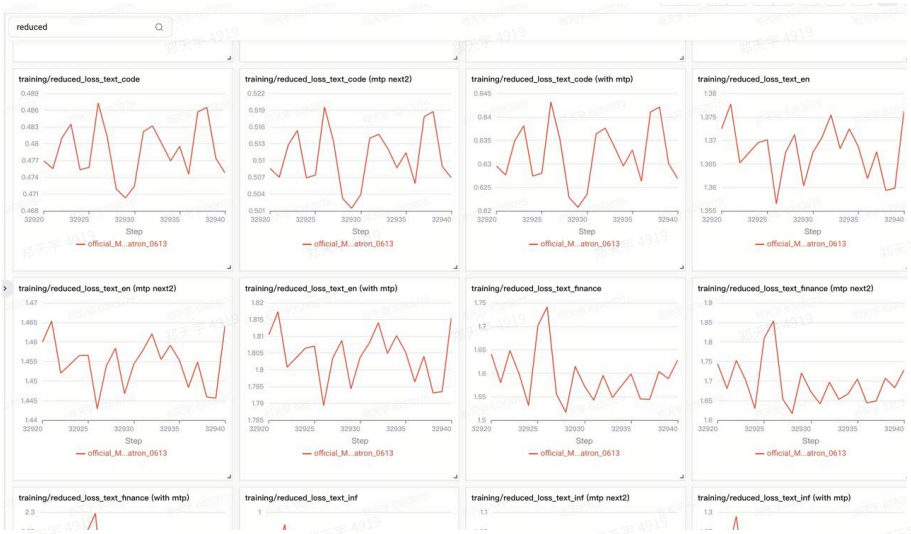
消失原因：为了resume consume token 线上任务打开了++resume_logger_state=True，然而这个在fsdp和megatron之间resume的时候存在问题，没有适配。

具体原因：主要原因在于两边原本的category reduced loss 是靠data_logger中的amortize_steps来控制的，然而fsdp那边之前有个mr 的修改，直接给amortize_steps设置成了1000000，不再用data_logger来控制打点，而是在外面用直接log_every_n_steps来控制，导致megatron resume到这个值后，会每1000000打一次点。

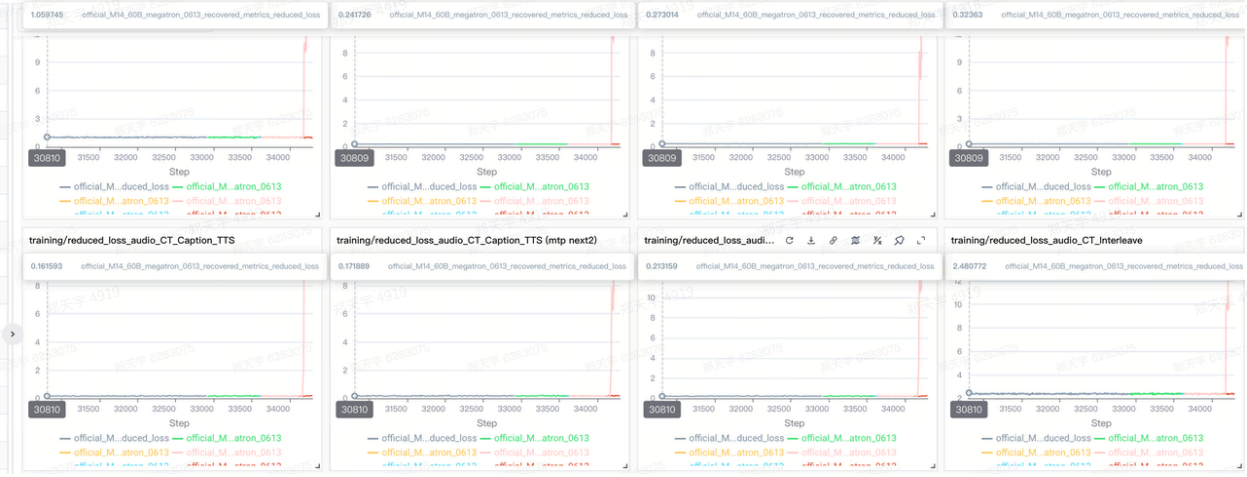
修复：

- 临时hack修复：将amortize_steps设置成与log_every_n_steps对应，并且清空之前的category缓存列表，否则会出现list 转 tensor时，列shape不一致的现象。
- 代码修复：将两边重新统一成amortize_steps控制打点步数，统一用data_logger。
- 线上数据恢复：
  - 确定数据在每个pp-1 rank
  - 异步抓取每个pp-1 rank上的数据，并按位计算sum
  - hack代码获取每个reduced loss相关list，对应的key（因为ckpt只保存了数据，没保存key）
  - 根据key进行匹配，并计算reduce_loss、reduce_loss(mtp2)、reduce_loss(mtp)。

修复后：



恢复的reduced_loss(从resume fsdp开始)：



tracking：https://seed.byteintl.net/experiment/tracking/detail?Id=project_20260509_c3108233&selectedTrial=&tab=CHART&viewId=view_20260615_5293819a

mr：
https://code.byted.org/seed/mariana/commit/78022df33f9c1341d2878691e8621da50485484e

14. 最终对齐实验

Tracking: https://seed.byteintl.net/experiment/tracking/detail?Id=project_20260509_c3108233&selectedTrial=&tab=CHART&viewId=view_20260612_d3052f3e

Loss 万分位对齐、grad_norm 千分位对齐，fsdp和megatron上下交错，符合预期：