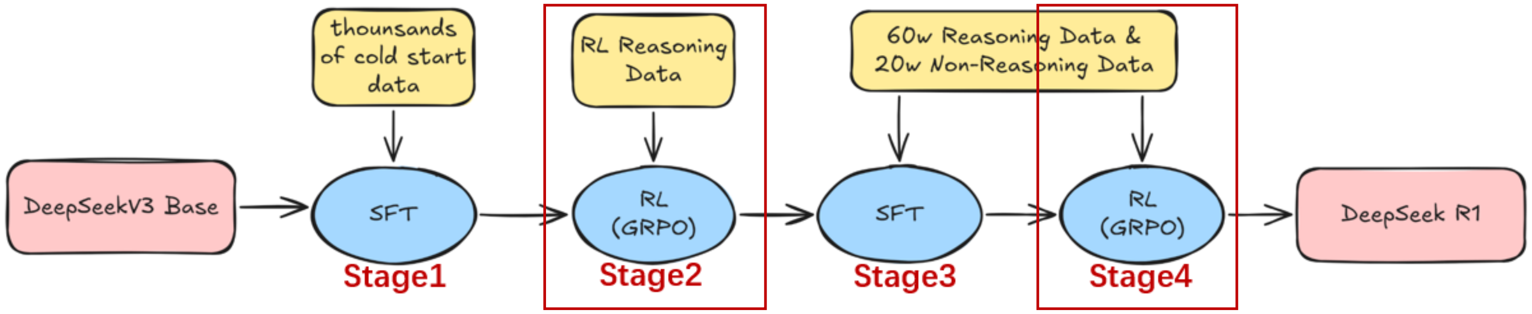
DeepSeek-R1-671B RL 训练优化实践

https://gitcode.com/cann

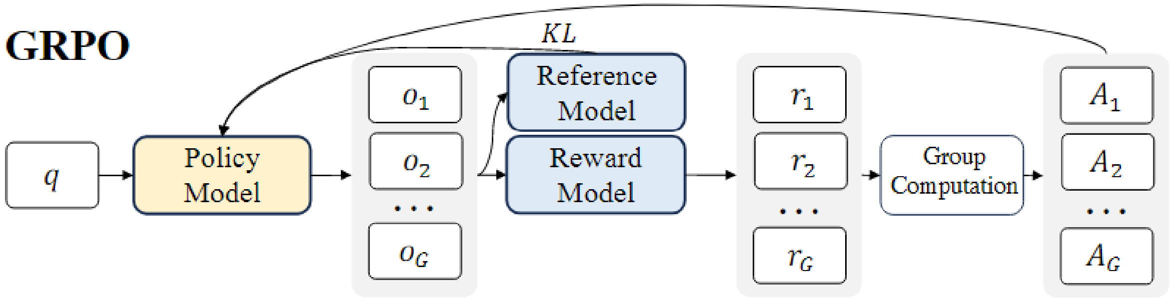
目录

1.1 背景

DeepSeek-R1-671B发布后，大模型的RL训练得到了广泛的关注，业界进行了大量RL训练实践，致力于构建更高效的RL训练系统。



➤ **DeepSeek R1训练流程**



➤ **GRPO算法流程**

verl: Volcano Engine Reinforcement Learning for LLMs

OPENRLHF

AReaL-lite Lightning-Fast RL

MindSpeed RL

ReaLHF

➤ **业界流行的多个RL训练框架**

1.2 基于verl框架的DeepSeek-R1-671B模型RL实践

本实践已经开源至gitcode代码仓：https://gitcode.com/cann/cann-recipes-train/blob/master/rl_train/deepseek/README.md

基于开源的verl框架，搭配MindSpeed与vLLM-Ascend框架，经过深入优化与适配，在Atlas 900 A3 SuperPoD超节点上实现了DeepSeek-R1-671B模型的高性能训练，相关代码正在向verl社区贡献(https://github.com/volcengine/verl/pull/3427)

加载DeepSeek-V3真实权重，限制最大推理长度为3K并使能性能优化各项配置，验证实际负载情况下的训推性能，**系统吞吐达到123TPS/卡。**

数据生成(s)				训练(s)	训推切换(s)		总耗时(s)	推理吞吐(tokens/s/卡)	训练吞吐(tokens/s/卡)	系统吞吐(tokens/s/卡)
523.38				319.3	17		859.68	271.05	331.6	123.16
rollout (s)	ref prefill (s)	reward (s)	adv (s)	update(s)	reshard(s)	offload等(s)				
373.65	144.5	5.02	0.21	319.3	11.5	5.5				

模型	部署方式	部署环境	RL框架	推理引擎	训练引擎	数据集	step	平均输入长度	平均输出长度	batchsize	采样数	rollout部署	actor部署	ref部署
DeepSeek-R1 -671B	全共卡	128 * A3	verl	vLLM+vLLM-Ascend	Megatron+MindSpeed	deepscaler	2	77.8	1576.6	512	16	256die **DP128TP2EP256**	256die **TP4EP8PP8**	256die **TP4EP8PP8**

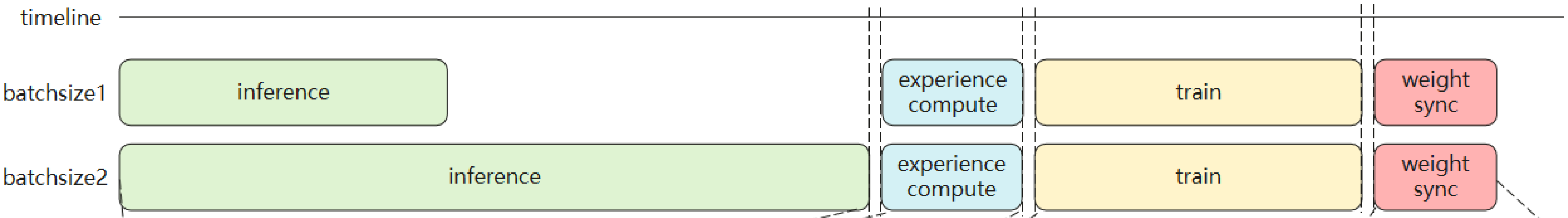
1.3 相关框架介绍

DeepSeek-R1-671B RL实践涉及到较多开源框架，如下表所示：

框架名称	定位	介绍
verl	RL框架	开源LLM强化学习框架，结合单控制器（控制流）与多控制器（计算流）优势，简化了RL算法（如PPO、GRPO）实现。其训练框架支持FSDP、FSDP2与Megatron-LM，推理支持vLLM、SGLang与HFTransformers
vLLM	推理引擎	为大语言模型提供**高吞吐量**和**内存高效**推理服务的开源引擎，使用PagedAttention技术高效实现KVcache的内存管理，并支持许多高性能推理特性。
vLLM-Ascend	推理引擎NPU插件	vLLM的Ascend NPU适配插件，支持NPU上的诸多优化。本实践过程中的一些优化贡献到了该开源仓。
Megatron-LM	训练引擎	超大规模语言模型训练框架，支持多种分布式并行策略。
MindSpeed	NPU训练加速库	针对NPU的大模型训练加速库，提供Megatron-LM框架对NPU的基本功能适配以及一些亲和NPU的优化特性。
MindSpeed RL	RL框架	基于CANN生态的强化学习加速框架，本优化实践对其进行了一些借鉴参考。

1.4 verl适配和优化实践总览

除了训练和推理引擎的NPU适配，本实践在如下几个方面对RL训练进行了性能优化：



stage	加速措施
调度	TorchAir整图下沉
	vllm DP域同步通信下沉
通信	零冗余TP转EP通信优化
算子	MLA多流并行
	昇腾亲和NZ格式支持
	冗余算子消除

stage	加速措施
advantage传递	任务间共用数据管理
	进程间D2D传输数据流设计

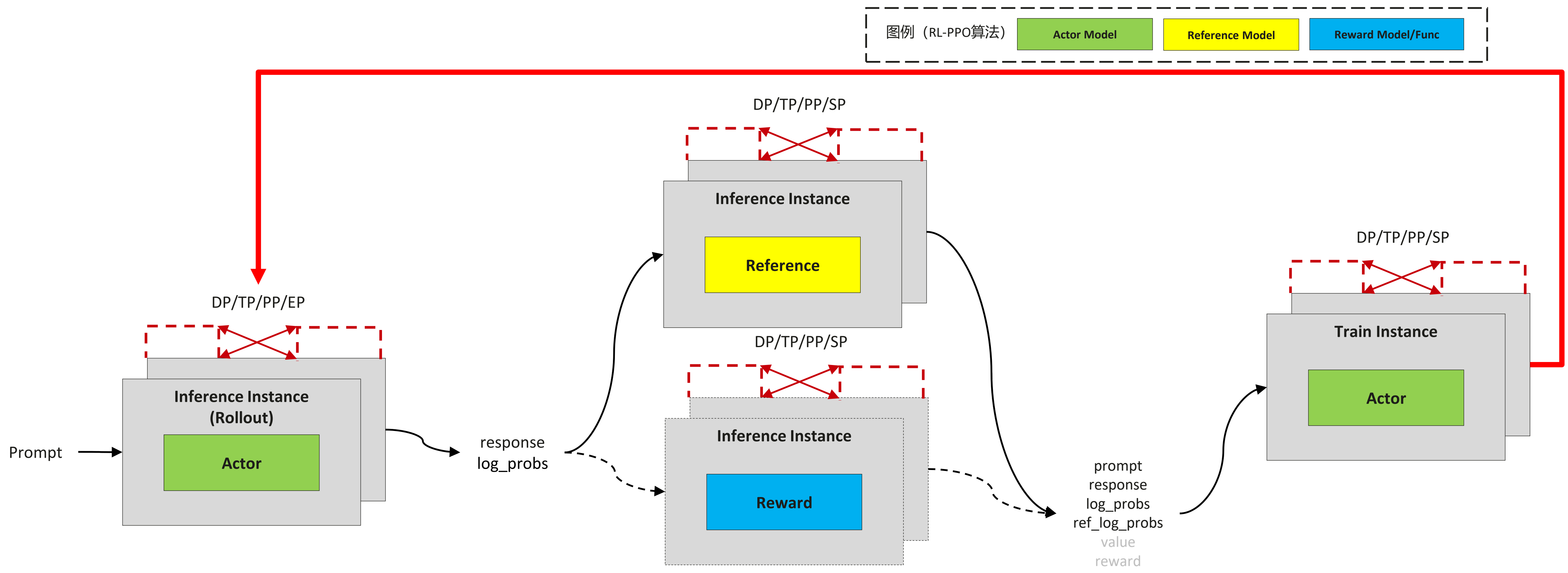
stage	加速措施
训练	MindSpeed预训练优化复用
	old_log_prob免计算

stage	加速措施
训推权重reshard	基于AlltoAllv的专家参数定向路由
	对PP域Broadcast通信优化

本实践基于verl 指定版本优化
comimit id：54c9b7364c2d188b2ba4107404cfa3c2b446df19

目录

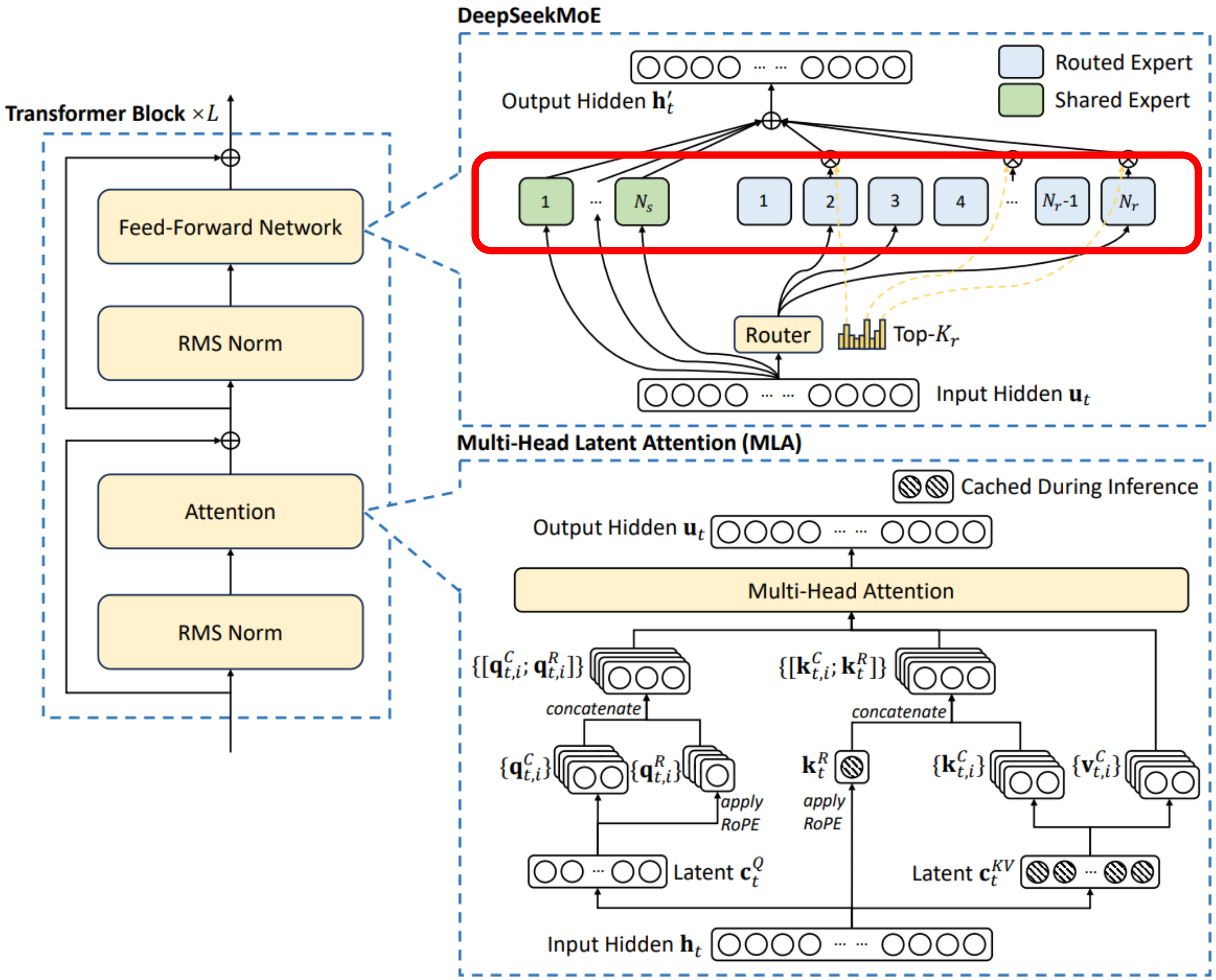
4.1训推切换



图例（RL-PPO算法）
Actor Model
Reference Model
Reward Model/Func
DP/TP/PP/EP
Inference Instance
(Rollout)
Actor
Prompt
response
log_probs
DP/TP/PP/SP
Inference Instance
Reference
DP/TP/PP/SP
Inference Instance
Reward
prompt
response
log_probs
ref_log_probs
value
reward
DP/TP/PP/SP
Train Instance
Actor

4.1训推切换

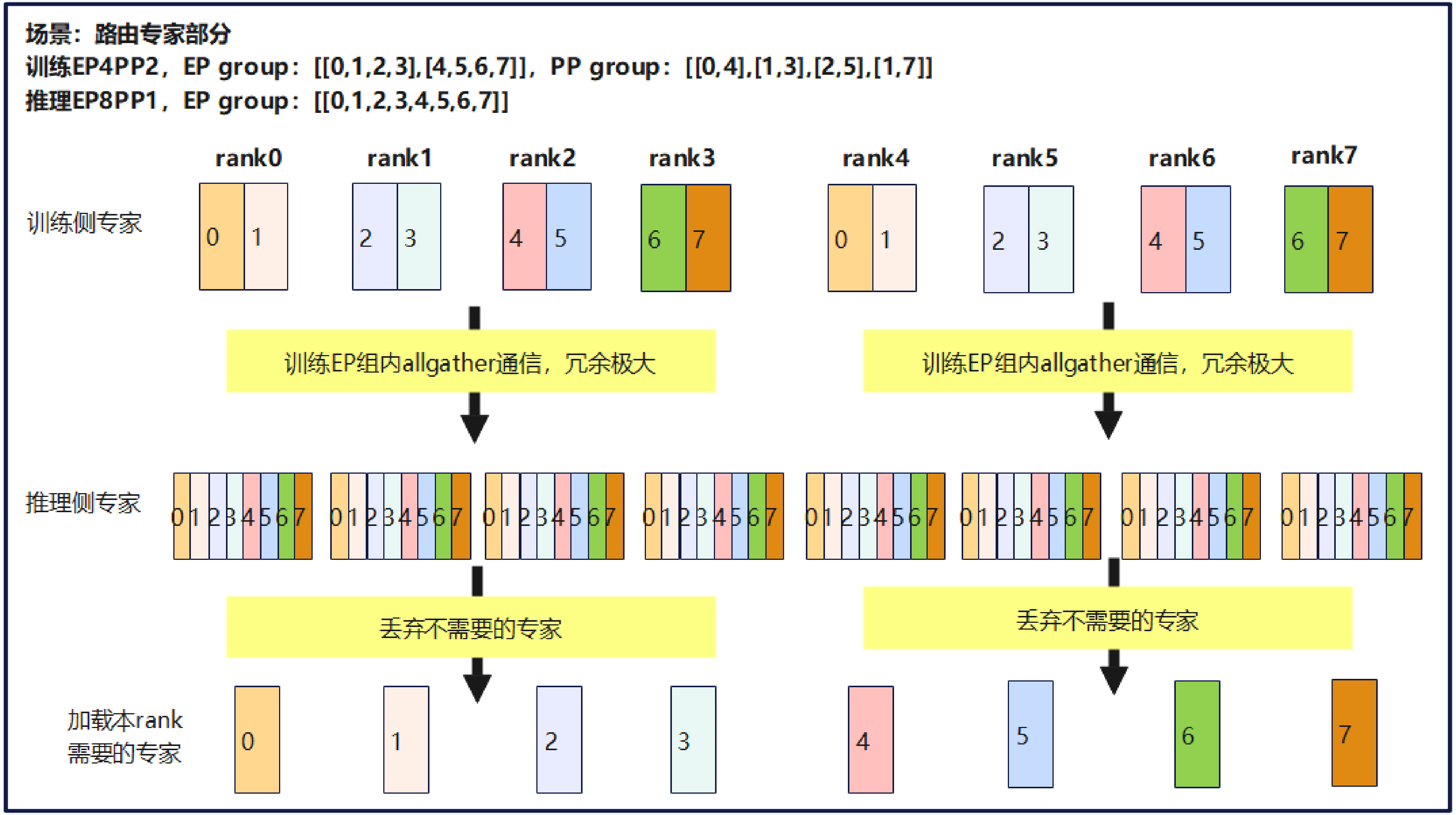
DeepSeek-R1-671B作为典型的MoE架构模型，其专家层（Expert Layer）参数规模占比超70%，且采用动态路由机制，使得专家参数（EP）的分片迁移成为训练转推理阶段的关键瓶颈之一



CANN

4.1训推切换——路由专家迁移的性能问题分析

原始方案

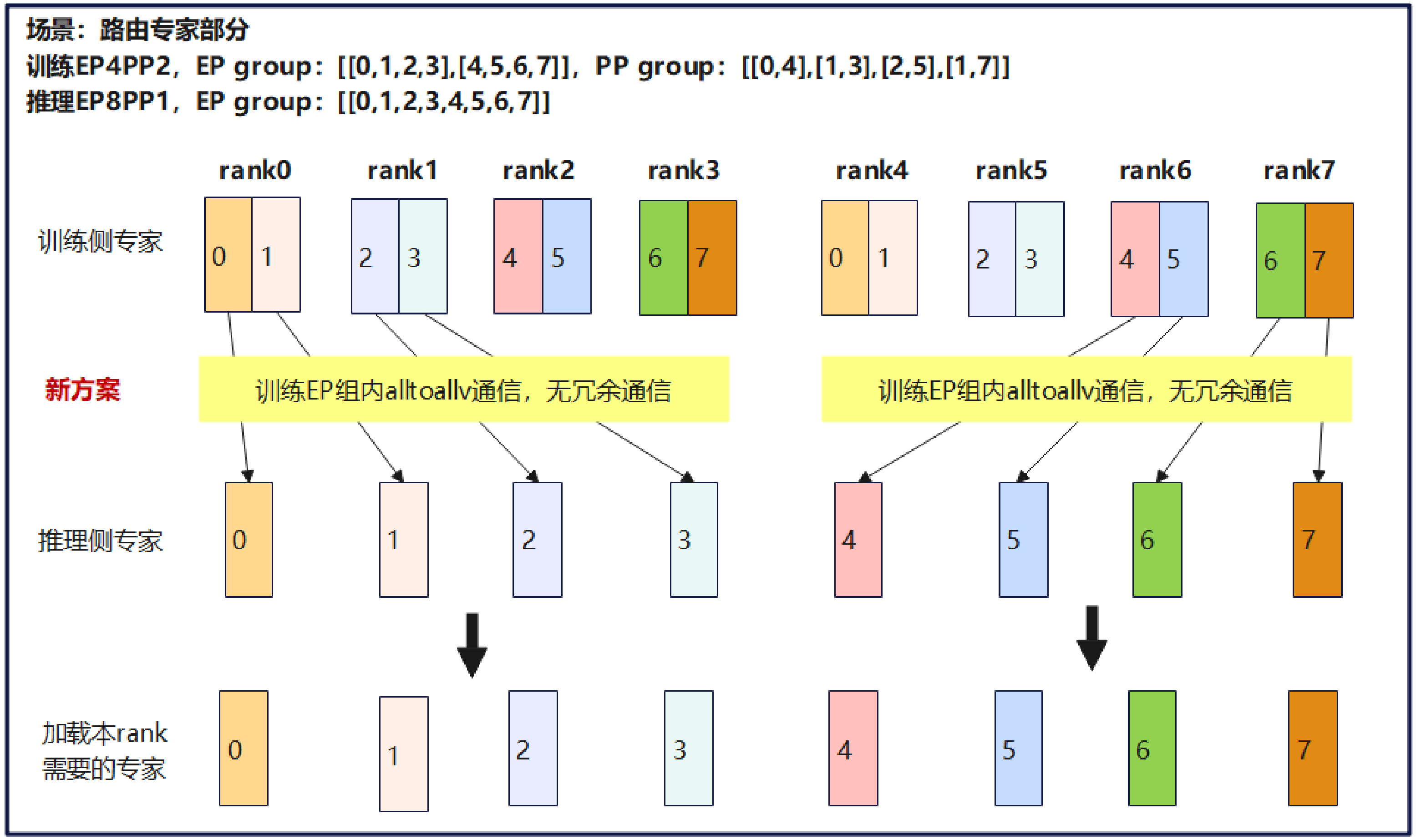


①通信冗余
②内存峰值大

https://gitcode.com/cann

代码地址：https://gitcode.com/cann/cann-recipes-train/blob/master/rl_train/deepseek/verl_patches/utils/megatron_utils.py

4.1训推切换——基于 AlltoAllV 的专家参数定向路由方案

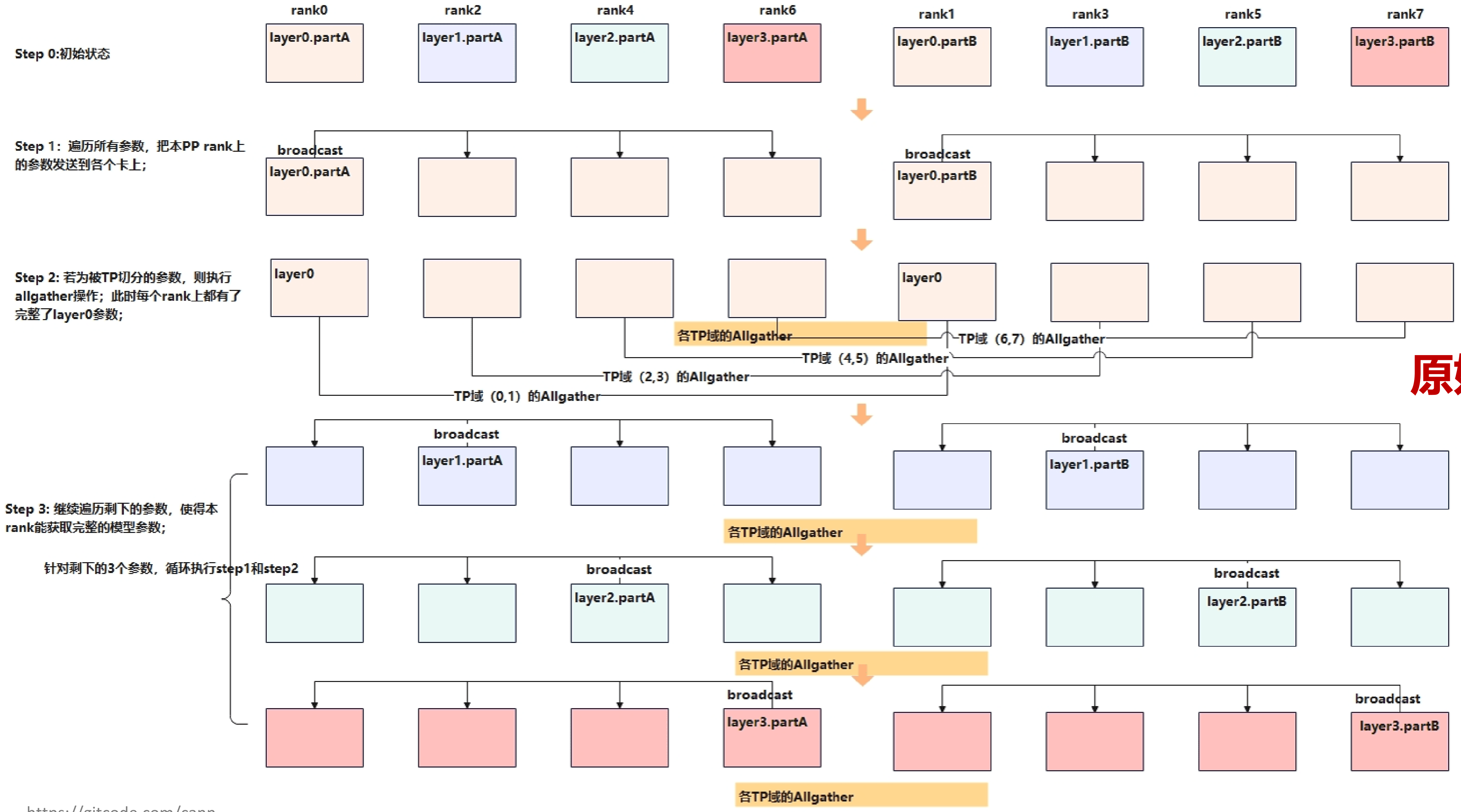


	单层专家数	单卡单层推理侧路由专家w1参数量	推理EP	单卡增量内存
优化前	256	2*(7168*2048)*2=0.0547 GB	256	AllGather需要准备全部专家所需空间：0.0547*256=14G
优化后	256	2*(7168*2048)*2=0.0547 GB	256	AlltoAllV最多需要一个专家所需空间：0.0547*(256/256)=0.0547G
相对收益	-	-	-	99.6%

CANN

代码地址：https://gitcode.com/cann/cann-recipes-train/blob/master/rl_train/deepseek/verl_patches/utils/megatron_utils.py

4.2训推切换——PP域Broadcast通信问题分析



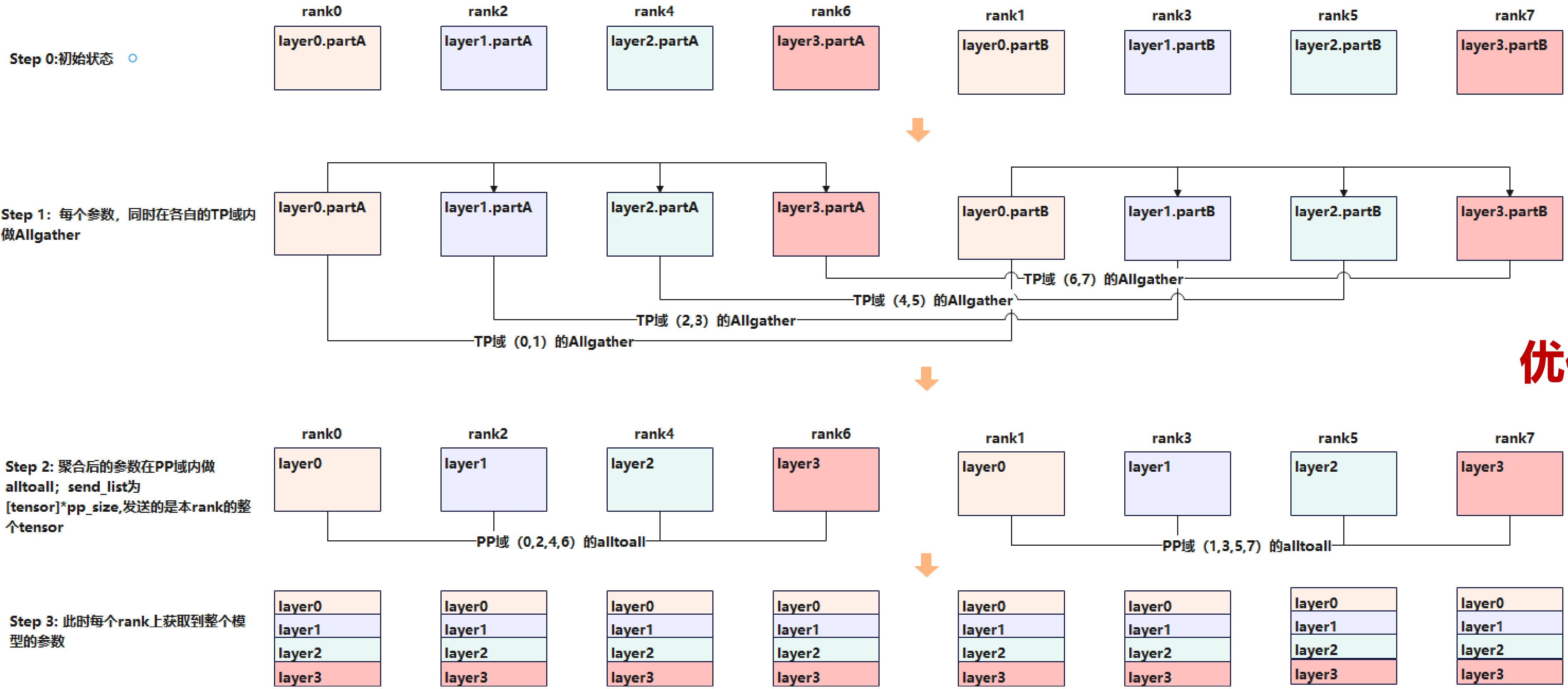
PP=4
TP=2
TP group: [[0,1] [2,3] [4,5] [6,7]]
PP group: [[0,2,4,6] [1,3,5,7]]
注释:
1）.partA和.partB表示参数被TP切分后的部分
2）这里的参数用layer0指代，实际上是layer里面对应的如“self_attention.linear_proj.weight”等参数;

原始方案

https://gitcode.com/cann

代码地址：https://gitcode.com/cann/cann-recipes-train/blob/master/rl_train/deepseek/verl_patches/utils/megatron_utils.py

4.2训推切换——PP域Broadcast通信优化



PP=4
TP=2
TP group: [[0,1] [2,3] [4,5] [6,7]]
PP group: [[0,2,4,6] [1,3,5,7]]
注释：
1）.partA和.partB表示参数被TP切分后的部分
2）这里的参数用layer0指代，实际上是layer里面对应的如“self_attention.linear_proj.weight”等参数；

优化方案

https://gitcode.com/cann

CANN

代码地址：https://gitcode.com/cann/cann-recipes-train/blob/master/rl_train/deepseek/verl_patches/utils/megatron_utils.py

4.3训推切换——offload顺序优化

①降低内存峰值
②消除不必要的权重卸加载

原始流程

推理前：
1) load_megatron_model_to_gpu
2) inference_engine.wake_up(tags=["weights"])
3) reshard
4) inference_engine.wake_up(tags=["kv_cache"])
5) offload_megatron_model_to_cpu

推理：generate_sequences

推理后：
1) inference_engine.sleep(level=1)

compute_ref_log_prob

训练前：
1) load_megatron_model_to_gpu
2) load_megatron_optimizer

训练：update_actor

训练后：
1) offload_megatron_model_to_cpu（卸载权重及梯度）
2) offload_megatron_optimizer

当前流程

推理前：
1) rollout.onload_model_weights()
2) reshard
3) offload_megatron_model_to_cpu
4) rollout.init_cache_engine()

推理：generate_sequences

推理后：
1) rollout.free_cache_engine()
2) rollout.offload_model_weights()

compute_ref_log_prob

训练前：
1) load_megatron_model_to_gpu
2) load_megatron_optimizer

训练：update_actor

训练后：
1) offload_megatron_model_to_cpu（仅卸载梯度，权重保留）
2) offload_megatron_optimizer

https://gitcode.com/cann/cann-recipes-train/blob/master/rl_train/deepseek/verl_patches/workers/sharding_manager/megatron_vllm.py

4.4 训练——old_log_prob免计算

GRPO这类on-policy训练算法，其old_log_prob计算结果与update_actor前向计算所得的log_prob数值相同（因为训练模型参数一致），因此可直接使用log_prob.detach()替代old_log_prob，从而省去一次训练模型前向计算的耗时。

on-policy中与分子数值相等

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}[q \sim P(Q), \{o_i\}_{i=1}^{G} \sim \pi_{\theta_{old}}(O|q)]$$

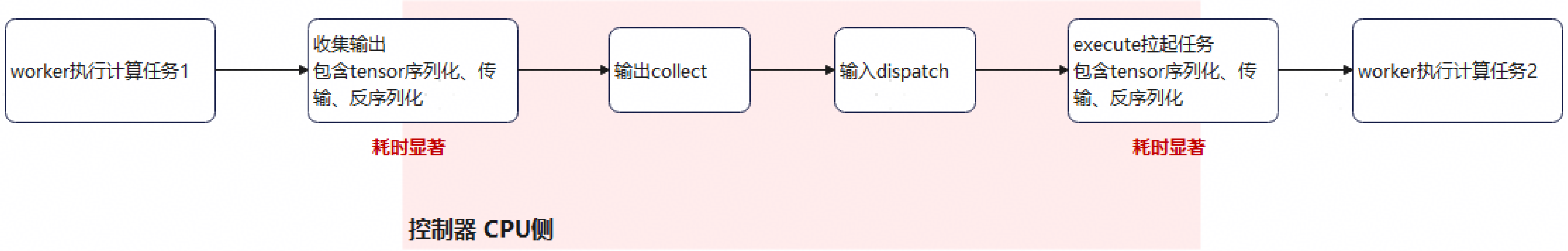
$$\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\left\{\min\left[\frac{\pi_\theta(o_{i,t}|q,o_{i,<t})}{\boxed{\pi_{\theta_{old}}(o_{i,t}|q,o_{i,<t})}}\hat{A}_{i,t}, \text{clip}\left(\frac{\pi_\theta(o_{i,t}|q,o_{i,<t})}{\boxed{\pi_{\theta_{old}}(o_{i,t}|q,o_{i,<t})}}, 1-\varepsilon, 1+\varepsilon\right)\hat{A}_{i,t}\right] - \beta\mathbb{D}_{KL}\left[\pi_\theta||\pi_{ref}\right]\right\}$$

目录

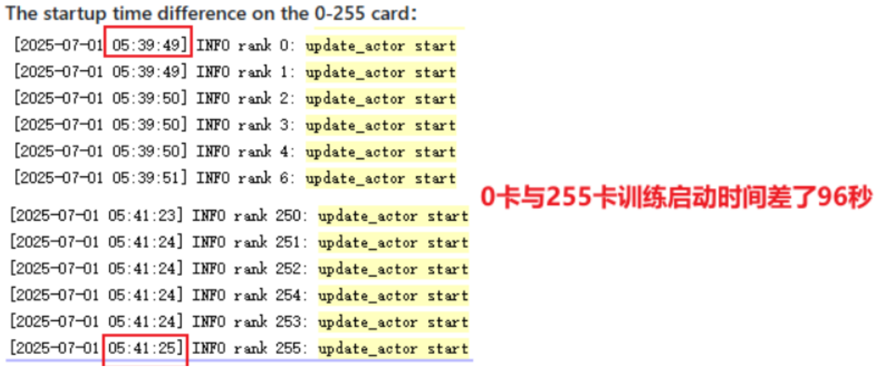
CANN

3.1 训练任务下发瓶颈分析

实践中发现并逐步分析训练性能问题，最终定位到verl任务调度时的数据传输是性能瓶颈。



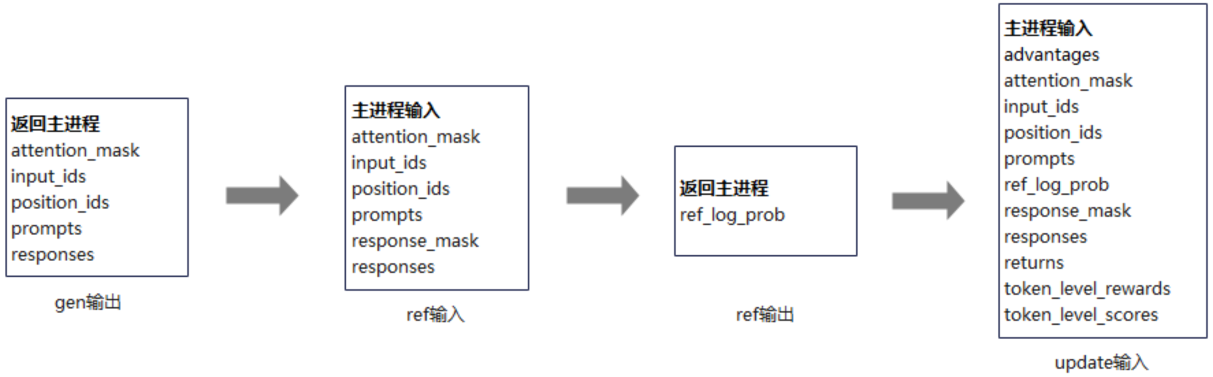
- Step1：Profile发现训练侧第一个通信耗时异常
- Step3：确认verl控制器任务调度逻辑，串行任务下发时，数据在Host和NPU间传输可能成为瓶颈



The startup time difference on the 0-255 card:

```
[2025-07-01 05:39:49] INFO rank 0: update_actor start
[2025-07-01 05:39:49] INFO rank 1: update_actor start
[2025-07-01 05:39:50] INFO rank 2: update_actor start
[2025-07-01 05:39:50] INFO rank 3: update_actor start
[2025-07-01 05:39:50] INFO rank 4: update_actor start
[2025-07-01 05:39:51] INFO rank 6: update_actor start
[2025-07-01 05:41:23] INFO rank 250: update_actor start
[2025-07-01 05:41:24] INFO rank 251: update_actor start
[2025-07-01 05:41:24] INFO rank 252: update_actor start
[2025-07-01 05:41:24] INFO rank 254: update_actor start
[2025-07-01 05:41:24] INFO rank 253: update_actor start
[2025-07-01 05:41:25] INFO rank 255: update_actor start
```

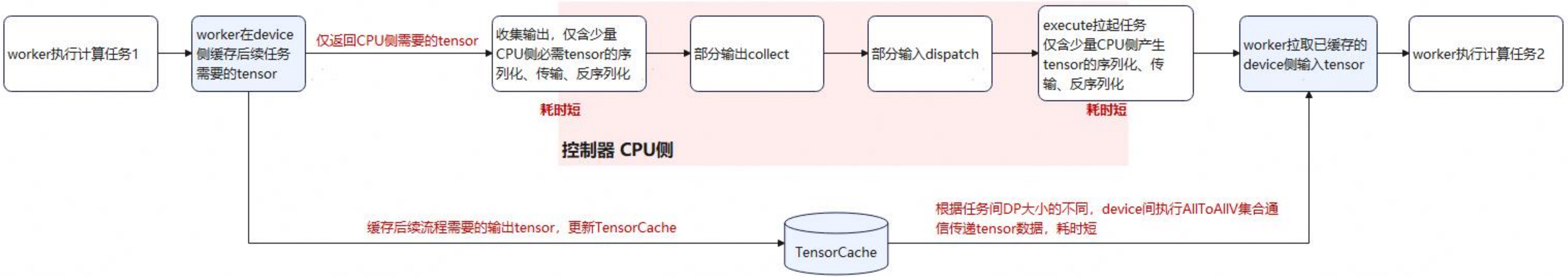
0卡与255卡训练启动时间差了96秒



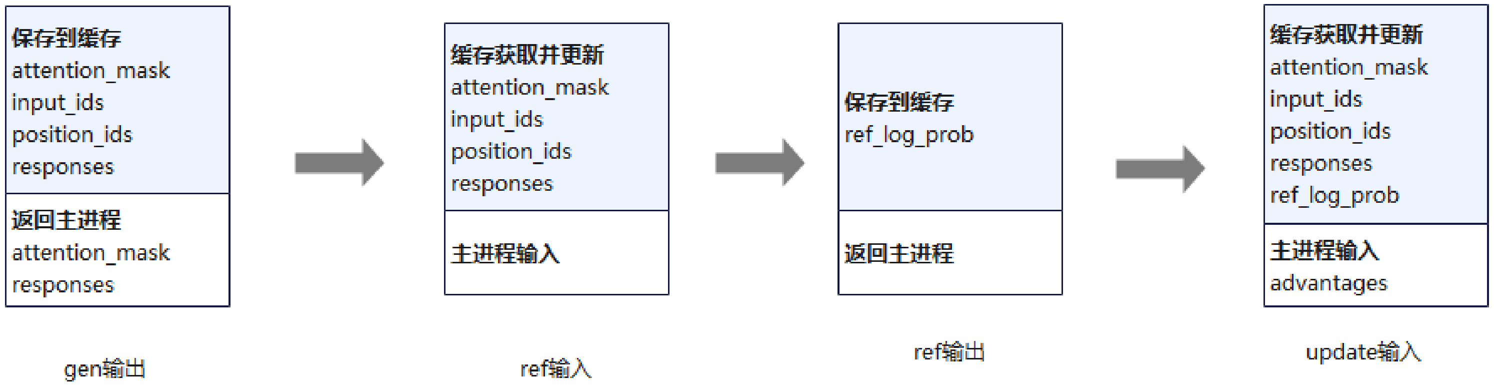
- Step2：识别到不同worker启动时间差异导致通信耗时异常
  修改位置：verl/single_controller/ray/base.py : execute_all_async
- Step4：整理verl任务间数据流，任务下发耗时与任务输入数据量成正相关，验证猜想

3.2 基于TensorCache的任务间数据传输性能优化

优化目标：减少数据在任务间传递的耗时，同时维持verl原本的控制器逻辑。
设计方案：使用TensorCache在NPU上管理前后任务间共用的tensor，减少tensor在Host和NPU间传输，降低任务下发耗时。



➢ **新方案中任务调度的流程**

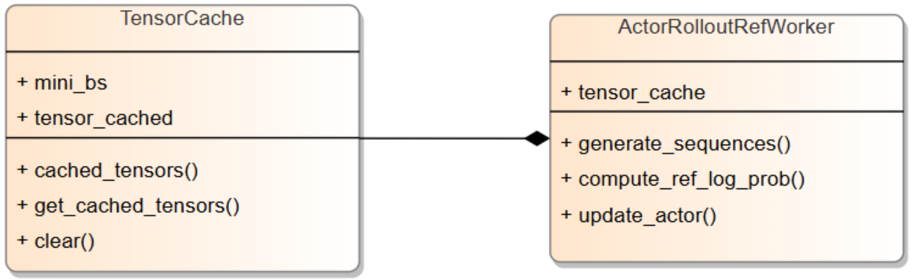


➢ **优化后的任务间数据流，Host和NPU间数据传输量大大减少**

CANN

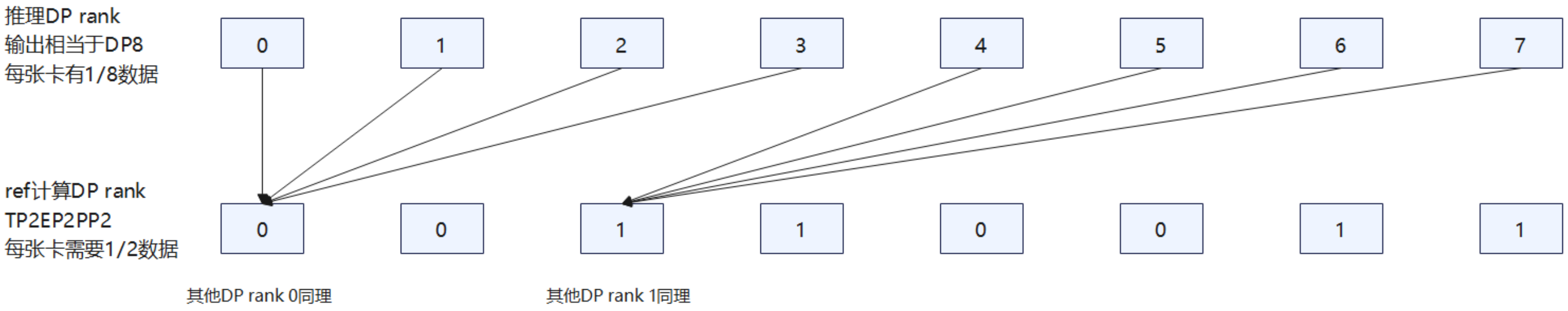
3.2 基于TensorCache的任务间数据传输性能优化

每个Worker实例，即每块NPU，将持有一个TensorCache管理RL训练中的数据。



属性或方法	作用说明
mini_bs	GRPO中单步训练的batchsize大小，结合cache中tensor的batch轴大小计算其src数据对应的DP size。
tensor_cached	cache中缓存的tensor字典，key是tensor的name。
cached_tensors()	给定DataProto数据，缓存其中的tensor。
get_cached_tensors()	从cache中获取指定name的tensor，返回一个DataProto实例。
clear()	清空cache，一般在一轮训练结束后调用。

代码地址：https://gitcode.com/cann/cann-recipes-train/blob/master/rl_train/deepseek/verl_patches/tensor_cache.py



➢ **当前任务与前序任务的DP rank排布不一致，获取缓存tensor时需要利用AllToAllV通信来重新切分tensor**

3.3 数据传输优化收益

在Atlas A3 128卡环境上进行新方案验证，prompt长度为0.075K，response长度为3K。

	优化前启动最大时差	优化后启动最大时差	收益
compute_ref_log_prob	57s	<1s	98.2%
update_actor	73s	5s	93.2%

➢ **优化后，训练侧任务下发耗时基本被消除**

	优化前	优化后	收益
timing_s/ref（s）	156.804	105.061	33.0%
timing_s/update_actor（s）	400.891	334.152	16.6%
perf/throughput（tps）	208.5	238.46	14.4%

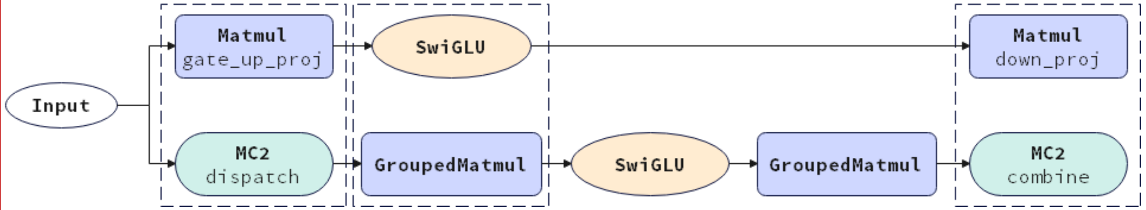
➢ **另一组相同规模的实验，优化后端到端性能收益显著**

目录

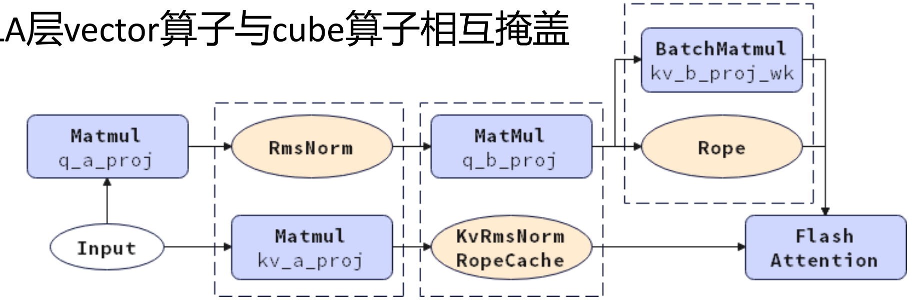
CANN

5.1 推理——算子侧优化

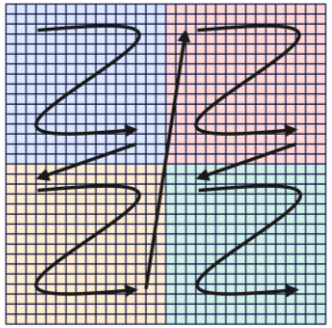
通算并行：DeepSeek-R1-671B共享专家计算与MoE层EP通信相互掩盖

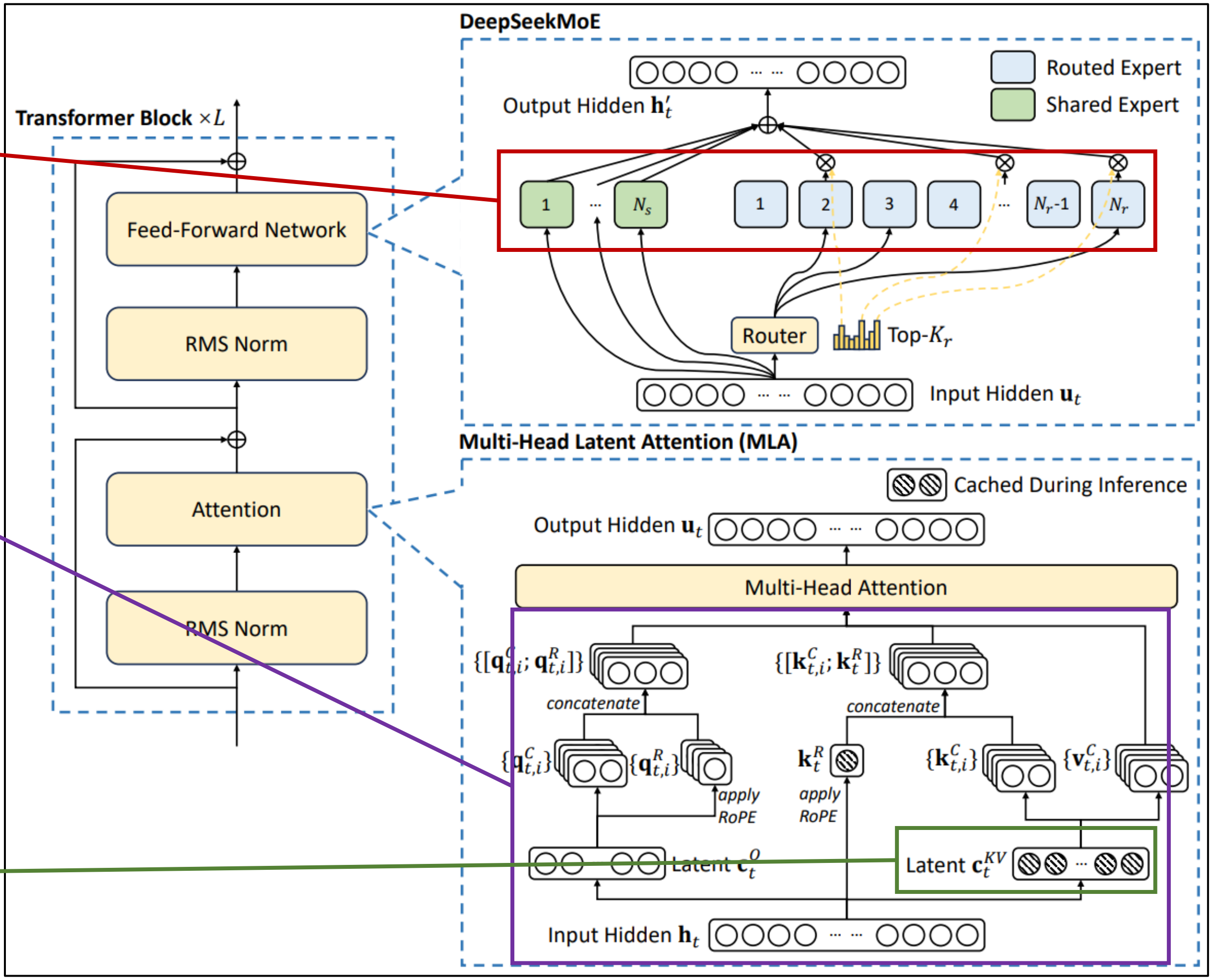


CV并行：MLA层vector算子与cube算子相互掩盖



Ascend亲和NZ数据格式：KVCache使用NZ数据格式，更加硬件单元访存更加亲和

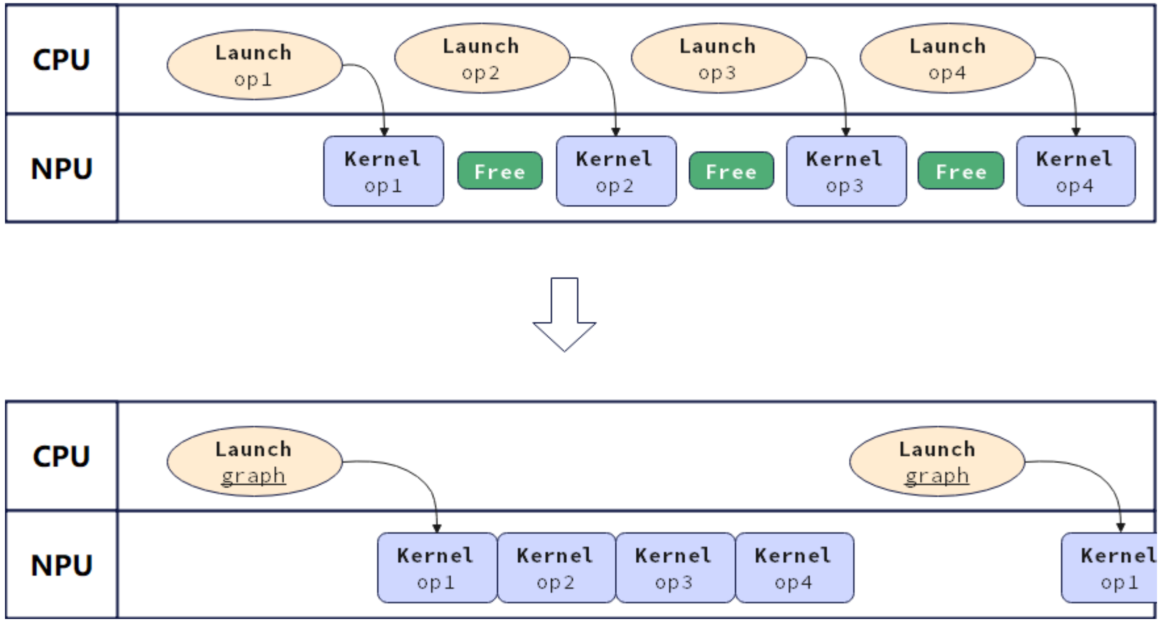




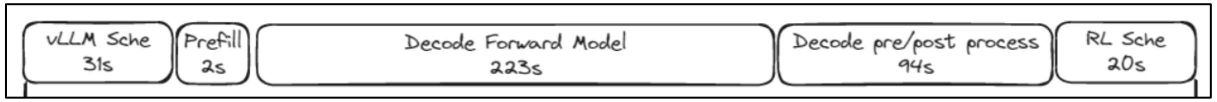
CANN

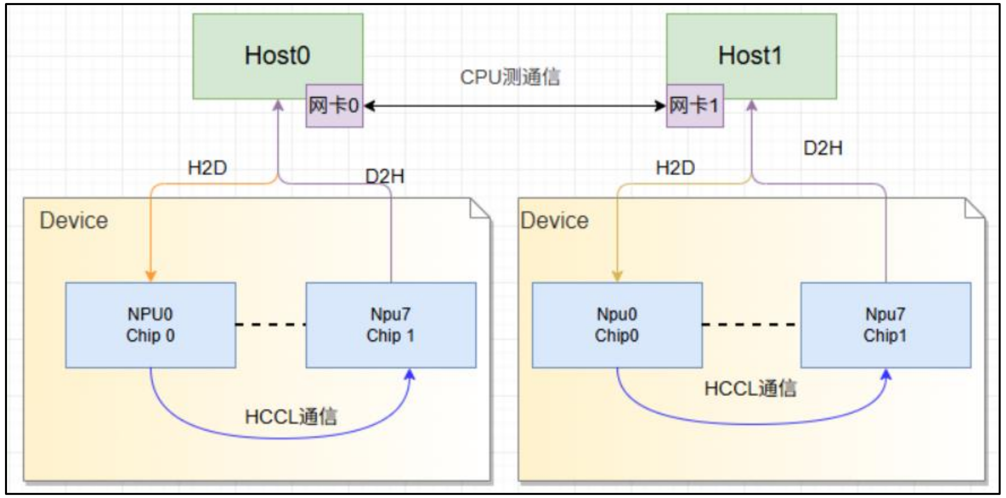
5.2 推理——框架侧优化

TorchAir图模式：整网推理算子编译成图统一下发，消除推理eager模式算子下发导致的NPU 空闲时间



框架调度通信优化：step间控制信息通信由Gloo切换为Hccl





对比项	NPU-HCCS (HCCL)	CPU-TCP (Gloo)
延迟 (Latency)	极低（微秒级，通常 < 10μs）	较高（数十到数百微秒，受OS调度/协议栈开销影响）
协议开销	专用硬件协议，绕过OS内核，处于数据链路层（L2）	需经过TCP/IP协议栈（内核态-用户态切换），处于传输层（L4）
大集群扩展性	近线性扩展	随节点数增加，延迟显著上升（树形/环形拓扑的跳数增加）

欢迎到仓库提交issue/PR

欢迎通过SIG联系我们

Thank you.

社区愿景：打造开放易用、技术领先的AI算力新生态
社区使命：使能开发者基于CANN社区自主研究创新，构筑根深叶茂、跨产业协同共享共赢的CANN生态

Vision: Building an Open, Easy-to-Use, and Technology-leading AI Computing Ecosystem
Mission: Enable developers to independently research and innovate based on the CANN community and build a win-win CANN ecosystem with deep roots and cross-industry collaboration and sharing.

上CANN社区获取干货　　关注CANN公众号获取资讯

https://gitcode.com/cann