

OpenCloudOS x SGLang x AMD Meetup Shenzhen

2025.12.20 (周六)

13:30-18:00

深圳市深南大道10000号

腾讯大厦2F多功能厅 (image楼上)

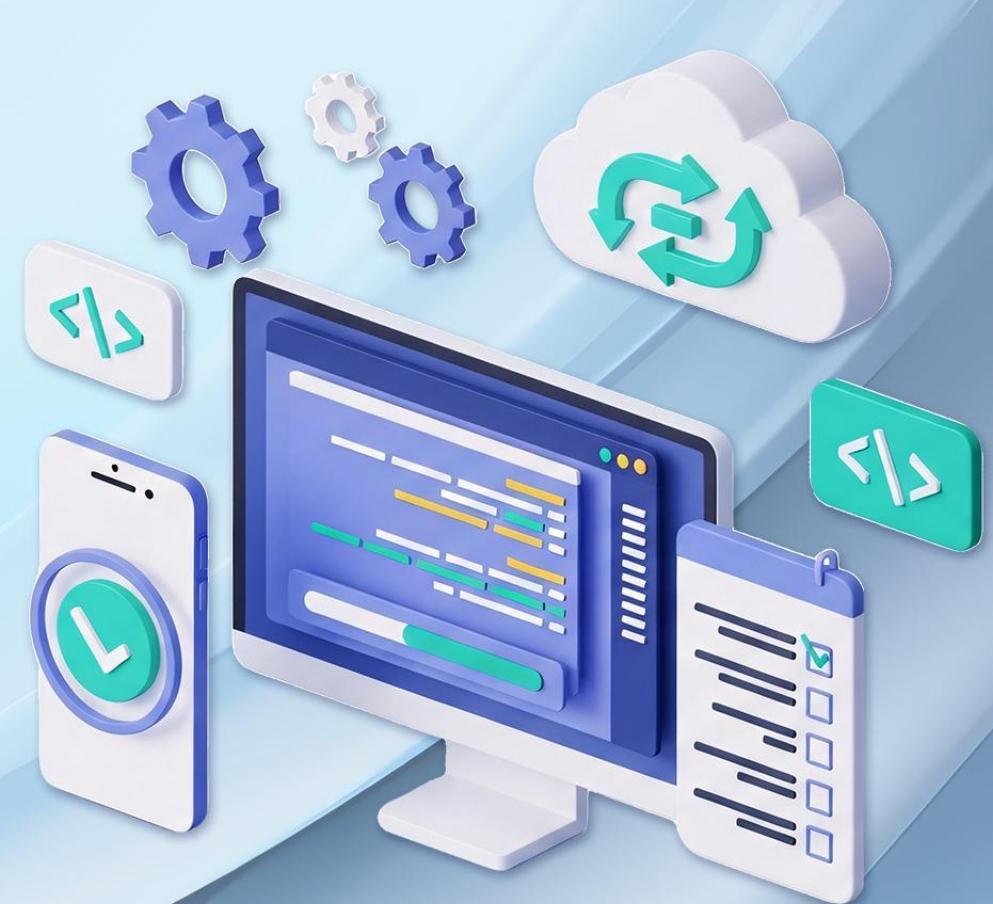
主办方:



合作伙伴:



协办方:





群聊: 深圳 SGLang Meetup
开发者群



该二维码7天内(12月26日前)有效, 重新进入将更新

17:10–18:10

讲师	环节内容
王佳	开场&欢迎致辞
梁杰鑫	SGLang Milestones & Roadmap
谢承兴	面向Agentic RL的高效训练框架Slime
程馨	OpenCloudOS 在 AI 场景上的支持与应用
	第一轮幸运抽奖+合影+Tea Break
何亚豪	大规模专家并行在ROCM平台上的最佳实践
冯明发	聊聊基于Slime的低精度训练
	第二、第三轮幸运抽奖
主讲导师: Xun Wang 技术指导: Vincent Fang, Daniel Huang, Bill He, Haichen Zhang	GPU实操工作坊: Triton Kernel开发

OpenCloudOS x SGLang x AMD Meetup Shenzhen

2025.12.20 (周六)

13:30-18:00

深圳市深南大道10000号

腾讯大厦2F多功能厅 (image楼上)

主办方:



合作伙伴:



协办方:





SGLang: An Efficient Open-Source Framework for Large-Scale LLM Serving

JieXin Liang

SGLang Core Developer @ LMSYS

Outline

- SGLang Overview
- Recent Feature Highlight
 - DeepSeek Large-Scale Deployment
 - Hierarchical Caching Integration
 - Zero-overhead Speculative Decoding
 - Efficient Speculative Draft Model Training
 - Deterministic Inference
 - Reinforcement Learning Integration
 - Model Deployment Orchestration
 - SGLang Diffusion
- Open-source Community & Future Roadmap (2025 Q4)

SGLang Overview

sgl-project/sqlang

SGLang is a fast serving framework for large language models and vision language models.



 GitHub

[GitHub - sgl-project/sqlang: SGLang is a fast serving framework for large ...](#)

SGLang is a fast serving framework for large language models and vision language models. – sgl-project/sqlang

 798 Contributors

 524 Issues

 286 Discussions

 19k Stars

 3k Forks



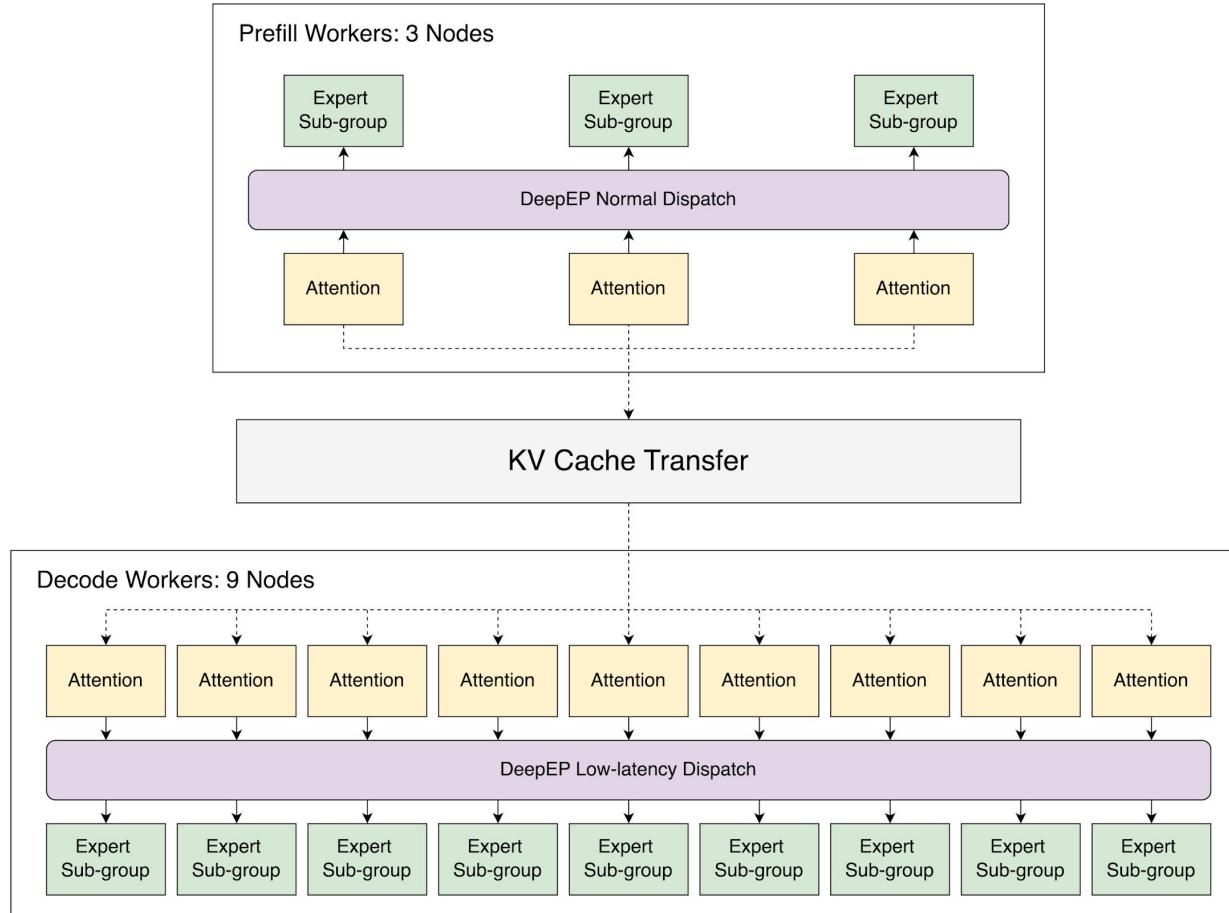
SGLang is a fast serving framework for large language models and vision language models.

Designed to deliver **low-latency** and **high-throughput inference** across a wide range of setups, from a single GPU to large distributed clusters.

Open-source, incubated by **LMSYS Org** and supported by a vibrant community with **widespread industry adoption**, powering over **300,000** GPUs worldwide.

Recent Feature Highlight

DeepSeek Large-Scale Deployment



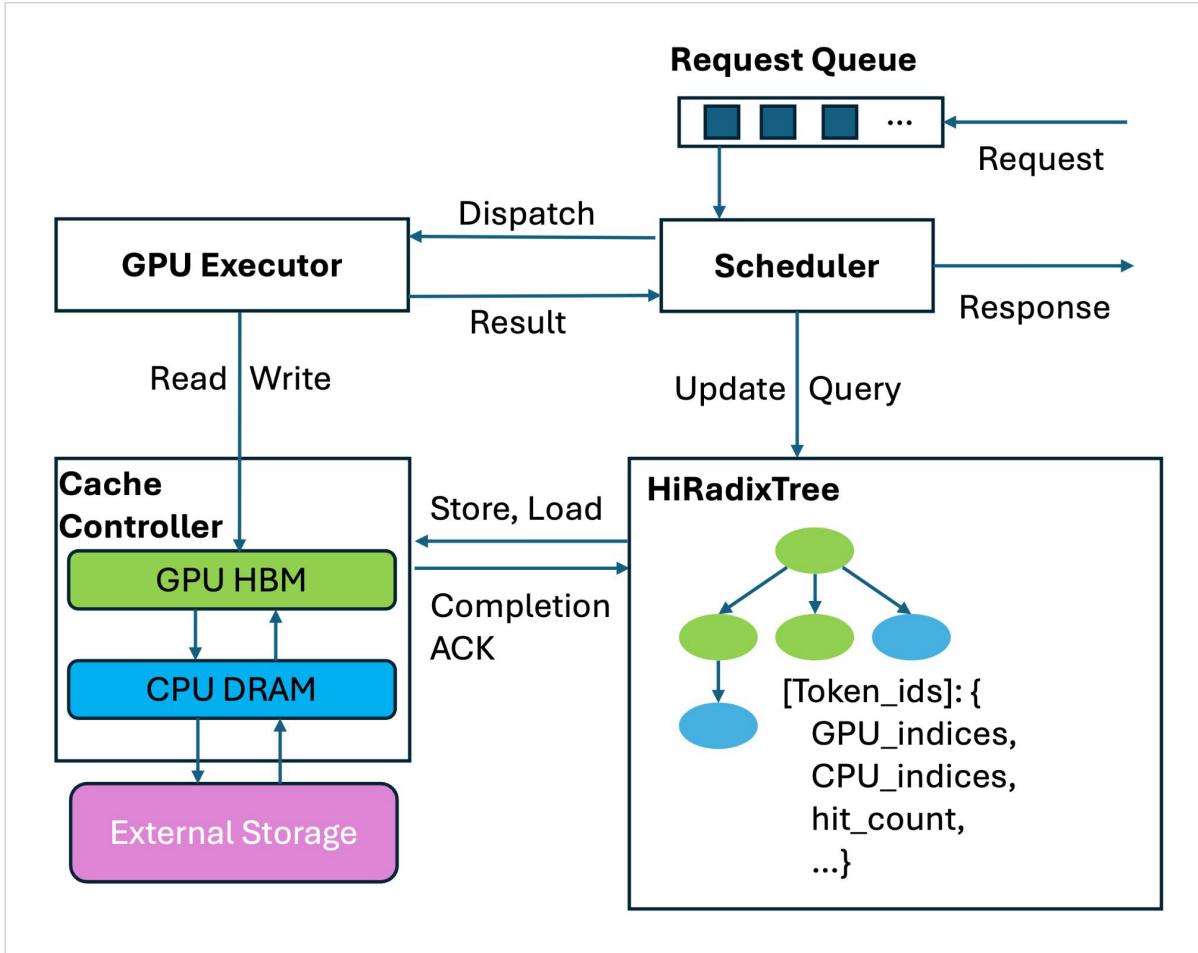
SGLang is **the first open-source system** that nearly match the performance of DeepSeek official blog with **PD disaggregation** and **EP** on 12 H100 nodes.

Performance at (May. 2025)

- 52.3k input token/s/node
- 22.3k output token/s/node
- **5x cheaper than DeepSeek API price**

Reproduced by 10+ other teams.

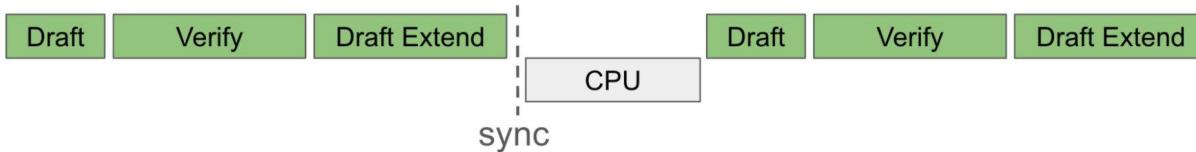
Hierarchical Caching Integration



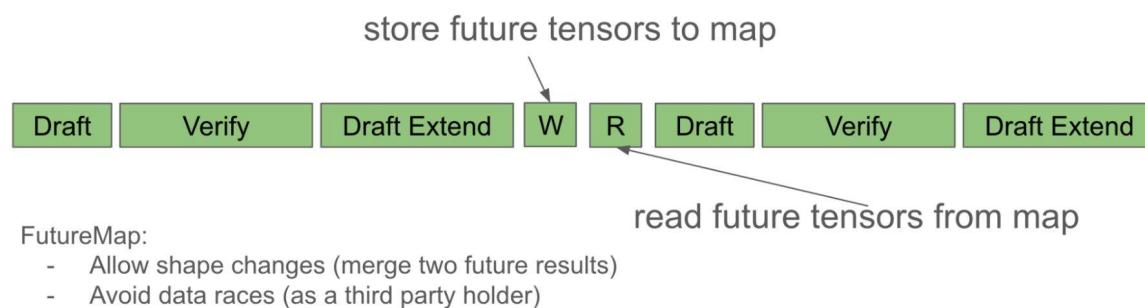
- Up to **6x** throughput and **84%** lower TTFT via **hierarchical KV reuse**
- HiRadixTree + GPU-assisted I/O → efficient cross-tier caching (GPU / CPU / storage)
- Smart prefetch & write policies hide transfer latency, boost cache hit rate
- Pluggable backends: Mooncake, 3FS — easy get/exist/set integration
- Adopted by Ant Group, Novita AI, Alibaba Cloud and others

Zero-overhead Speculative Decoding

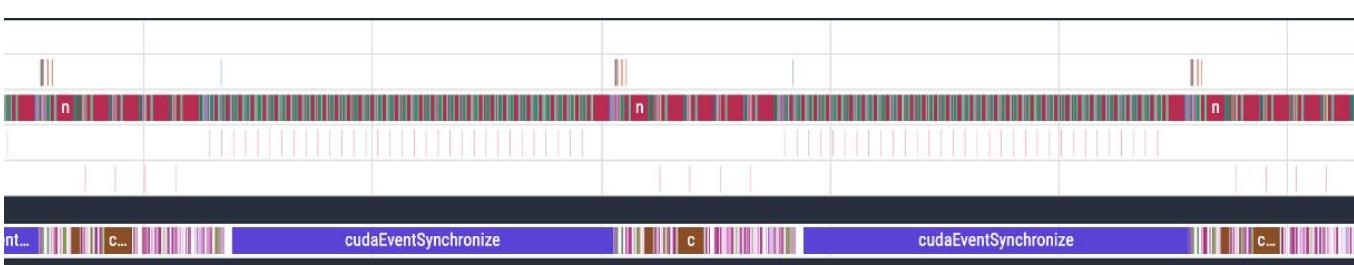
Normal scheduling (sync after each forward)



Overlap scheduling (no CPU overhead)



- SGLang has been pioneering the zero-overhead CPU runtime for LLM runtime since last year.
(<https://lmsys.org/blog/2024-12-04-sqlang-v0-4/>)
- We also tune the scheduler for speculative decoding and seeing **10% - 20%** speedup across the board.



For more detail: <https://github.com/sql-project/sqlang/issues/11762>

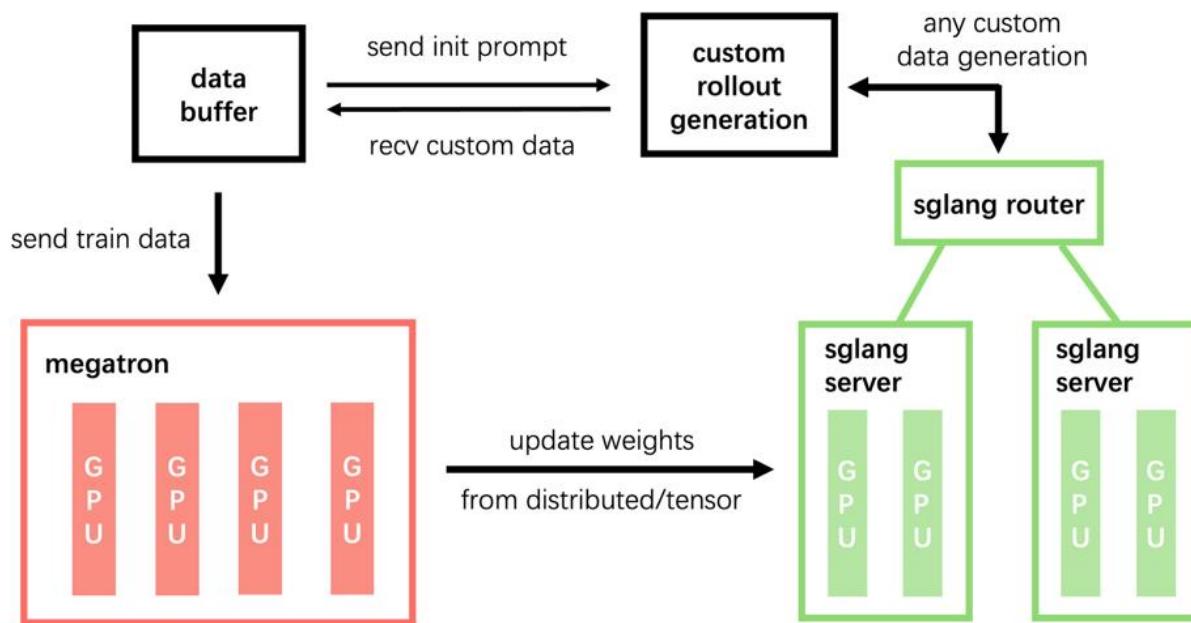
Efficient Speculative Draft Model Training



Train Eagle 3 for SGLang with SpecForge

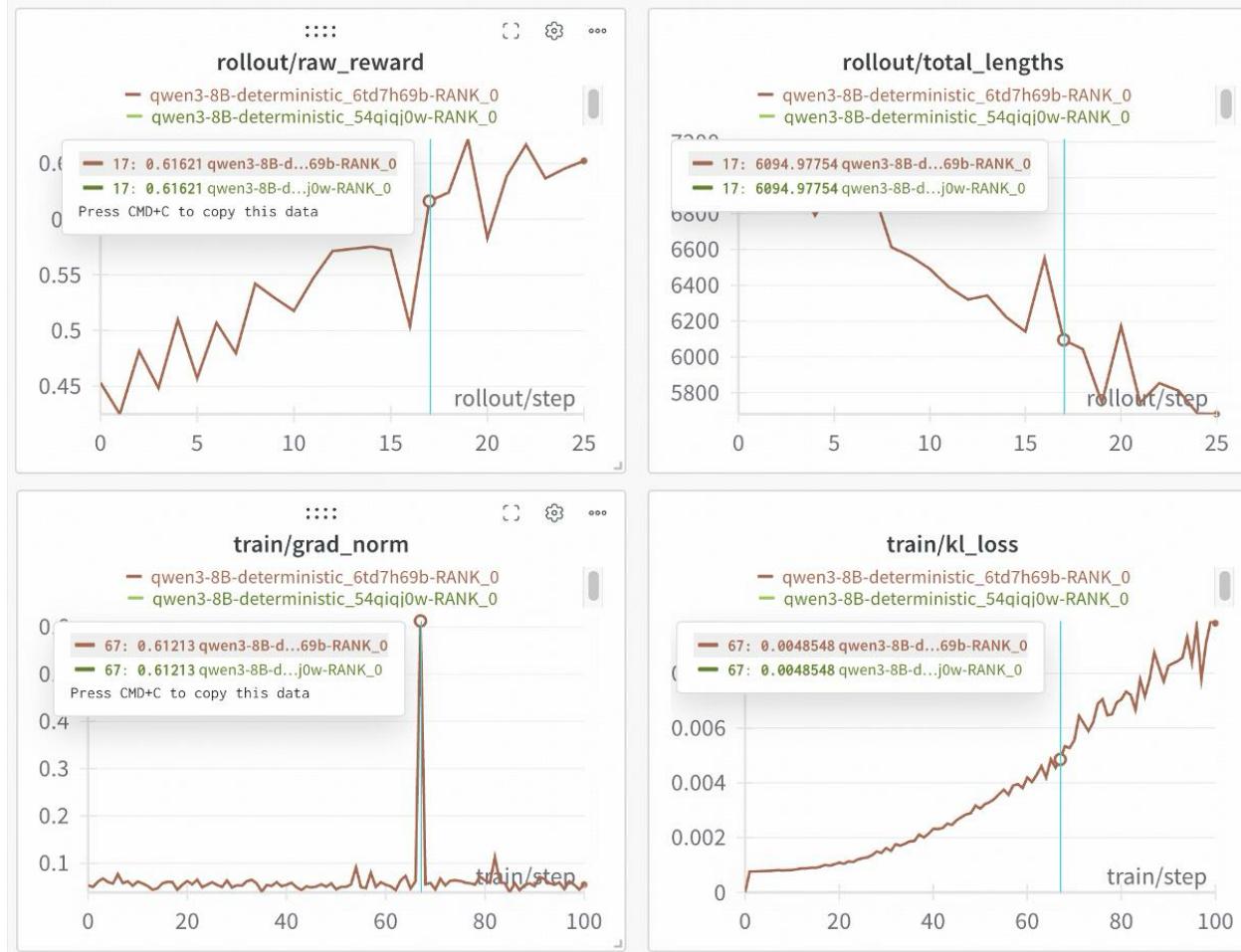
- Training draft models that integrate natively with SGLang
- Native Support for Advanced Model Architectures
 - Llama 4, DeepSeek, Qwen3 MoE, GTP-OSS...
- Scalable Distributed Training
- Memory-Efficient Training
- Collaborate with the EAGLE team

Reinforcement Learning Integration



- SGLang-exclusive reinforcement learning framework: **slime**
 - Architecture: Built on Megatron-LM and SGLang for extreme scalability
 - Use case: Powers large-scale training for GLM 4.5 and GLM 4.6
- SGLang other general integration: **veRL**, **AReAL**

Deterministic Inference



- SGLang enables **fully deterministic inference** with batch-invariant kernels
- Full compatibility with common inference features
- Reduce the slowdown from 61.5% (TML's Blog¹) to **34.35%** on FlashInfer and FlashAttention 3 backends
- **slime** achieves **100%** reproducible RL training on Qwen3-8B

For more detail: <https://lmsys.org/blog/2025-09-22-sqlang-deterministic>

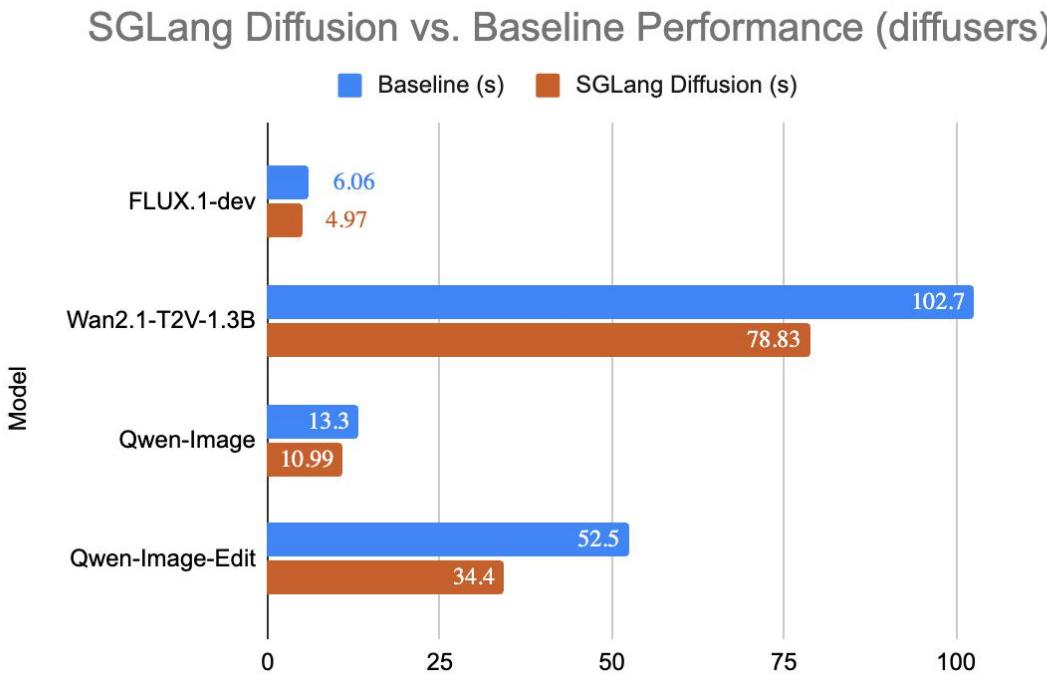
¹He, Horace and Thinking Machines Lab, "Defeating Nondeterminism in LLM Inference", Thinking Machines Lab: Connectionism, Sep 2025.

Model Deployment Orchestration



- **OME** is a Kubernetes operator for enterprise-grade management and serving of Large Language Models (LLMs)
- Use case: Launch a 128-GPU cluster for deploying Kimi-K2 with PD and EP with one click

SGLang Diffusion



- **Accelerate image and video generation** for diffusion models for production level serving
- **Major model support:** Wan, Hunyuan, Qwen-Image, Qwen-Image-Edit, Flux...
- Integrate highly-optimized **sgl-kernel** and advanced **parallelism** techniques
- Collaboration with the FastVideo team

Open-source Community & Future Roadmap

Community Adoption



AMD

NVIDIA

intel

LinkedIn

CURSOR

ORACLE

Google Cloud

Microsoft Azure

aws

Atlas Cloud



VOLTAGE PARK

NEBIUS

Data
Crunch

Novita

innomatrix



Ucla



Stanford

Berkeley
UNIVERSITY OF CALIFORNIA



Jam & Tea
STUDIOS

baseten



Meituan



ANT
GROUP

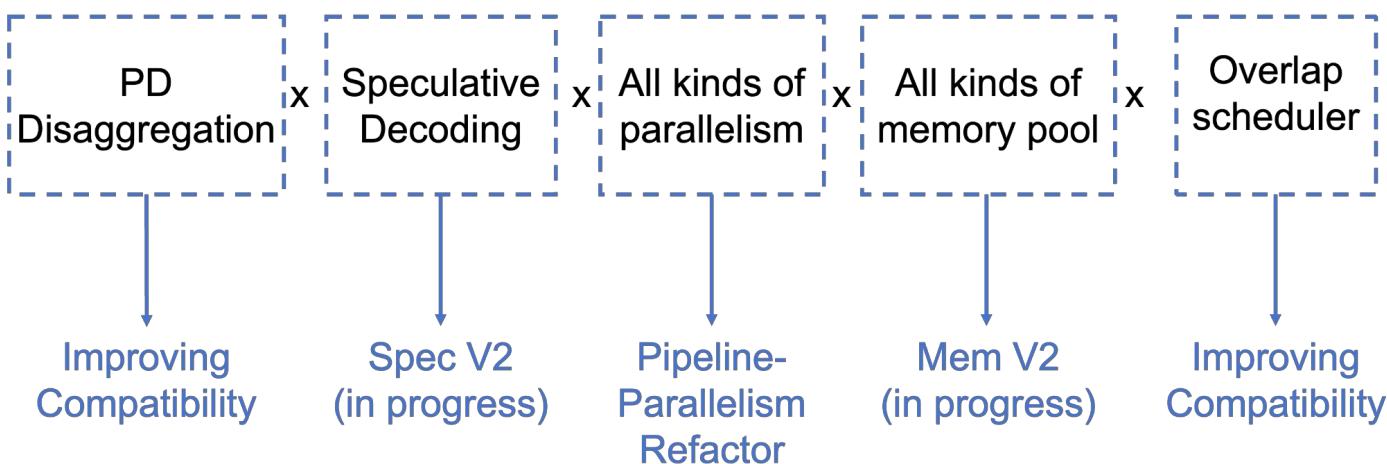
Alibaba Cloud



Tencent Cloud

Roadmap

- Development Roadmap (2025 Q4)
 - <https://github.com/sjl-project/sjlang/issues/12780>
- Focus
 - Feature compatibility & reliability.
 - Usability for easy installation, simple large-scale deployment.
 - Kernel optimization for next-gen hardware.
 - Reinforcement learning framework integration and training-inference mismatch mitigation.



Roadmap

- **Base Engine Features**

- Overlap scheduler compatibility with speculative decoding and all features
 - <https://github.com/sgl-project/sglang/issues/11762>
- Prefill CUDA graph
 - <https://github.com/sgl-project/sglang/issues/11490>
- Memory cache v2 refactor
 - <https://github.com/sgl-project/sglang/issues/12587>
- Torch compile stack
 - <https://github.com/sgl-project/sglang/issues/10118>
- Pipeline parallelism refactor
 - <https://github.com/sgl-project/sglang/issues/11857>
- SGLang Diffusion
 - <https://github.com/sgl-project/sglang/issues/12799>

Community Links

Github: <https://github.com/sgl-project/sglang>

Slack: <https://slack.sglang.ai>

Documentation: <https://docs.sglang.ai>

Blogs: <https://lmsys.org/blog>

Learning Materials: <https://github.com/sgl-project/sgl-learning-materials>

X: <https://x.com/lmsysorg>

LinkedIn: <https://www.linkedin.com/company/sgl-project>



Q & A

01

slime: 面向Agentic RL的大规模 MOE训练框架

谢承兴



slime: A LLM RL Training Framework

The screenshot shows the GitHub repository page for 'slime'. The repository is owned by 'THUDM' and has a public status. It features 106 branches and 4 tags. The main branch is selected. The repository has 334 forks and 2.9k stars. The code tab is active, showing recent commits from 'zhuzilin' and others. The commit details are as follows:

- zhuzilin [docker] optimize r3 with base64 encode (#1124) - 8a825f7 · 1 minute ago · 1,004 Commits
- .github/workflows add ckpt load save ci (#1104) - 3 days ago
- docker [docker] optimize r3 with base64 encode (#1... - 1 minute ago

The repository page also includes sections for About, Readme, and Apache-2.0 license.



Slime 框架最新进展



模型侧

- 大规模 MOE 模型的训练:
 - Kimi K2
 - Qwen 235B
 - Deepseek V3
- FP8 training + FP8 Rollout



Agent侧

- slime router
- 各类 IS 算法支持
- 多模态模型支持
- True on policy (Dense model & VLM)
- Rollout routing replay 支持





RL 范式的转变



从Reasoning Task-> Agentic Task



单轮 rollout-> 多轮 rollout



逐渐引入 Agent Framework/ Agent Infra...

什么样的框架设计可以丝滑的支持各种任务?





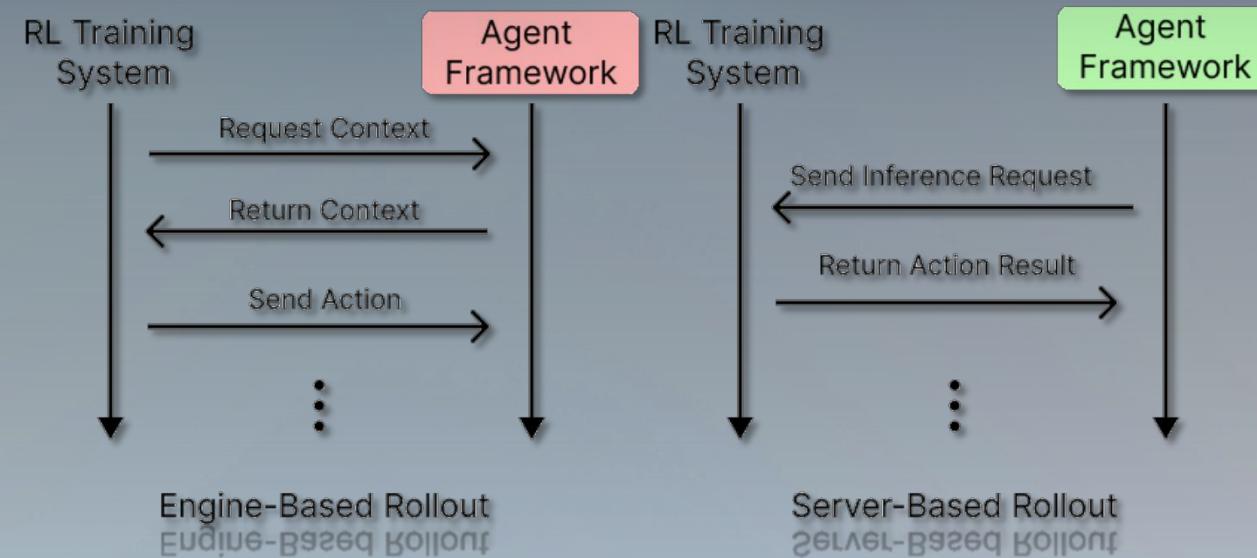
Server-based Rollout

Engine-Based Rollout

- 和各类agent framework生成轨迹方式背道而驰

Server-Based Rollout

- 提供api接口给用户进行rollout, 天生和 Agent 框架兼容





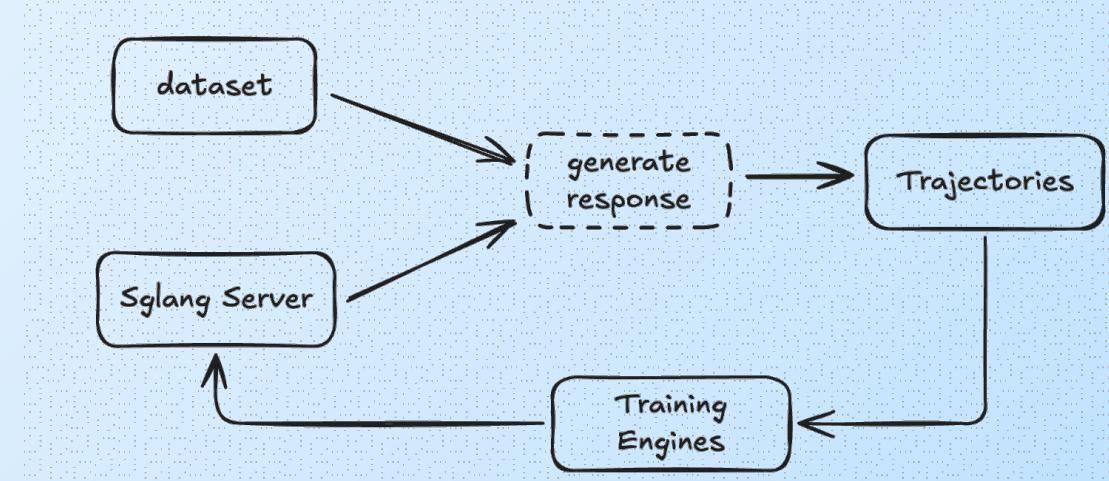
自定义 Rollout 接口

背景

不同 Agent 任务生成逻辑和环境可能差异巨大，很难统一标准。

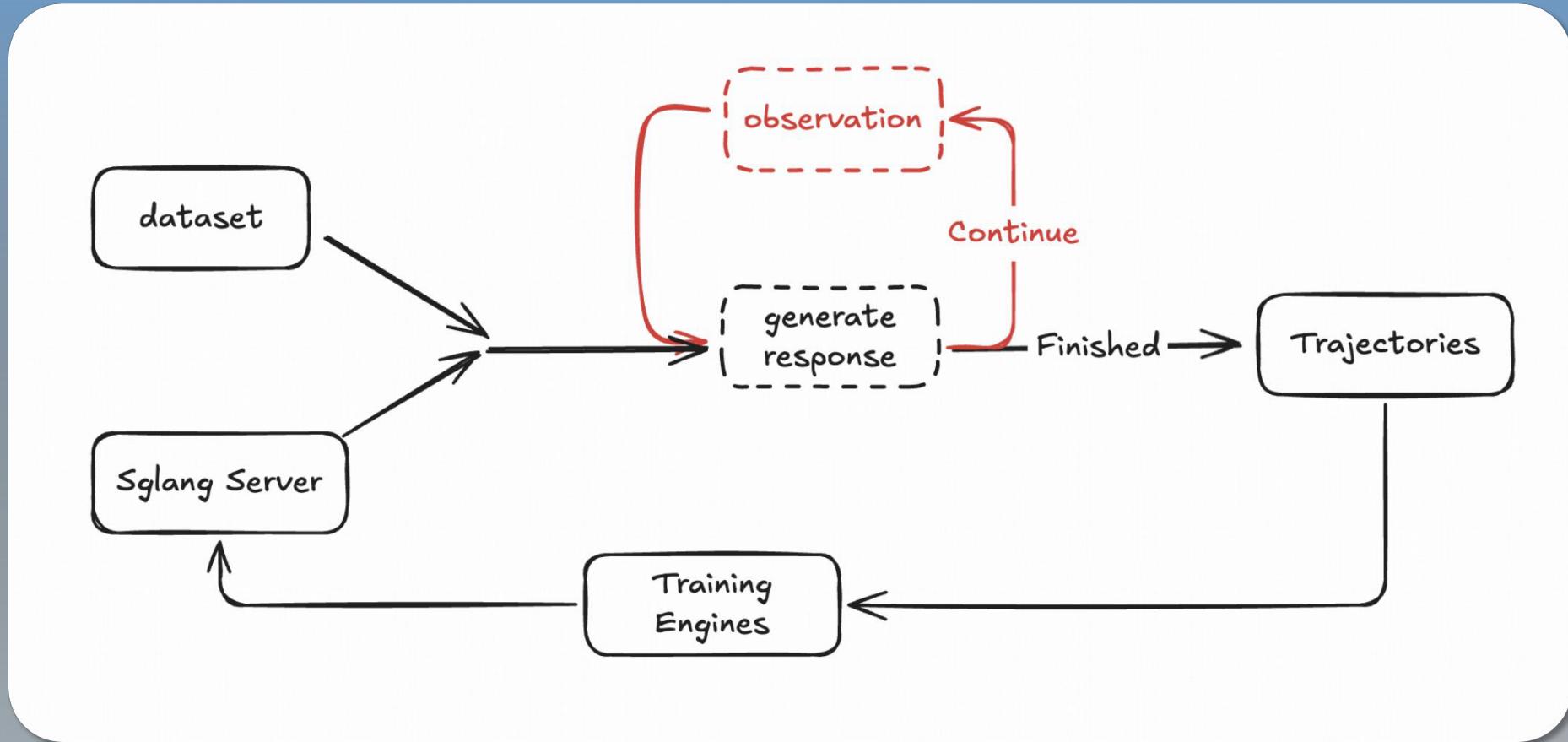
slime 的数据生成逻辑

- 提供给用户的
sglang server 接口
- 用户返回
轨迹和对应奖励

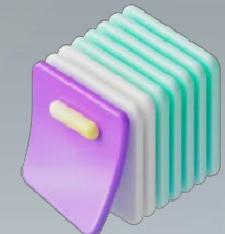




如何从之前单轮 rollout ->多轮 rollout ?



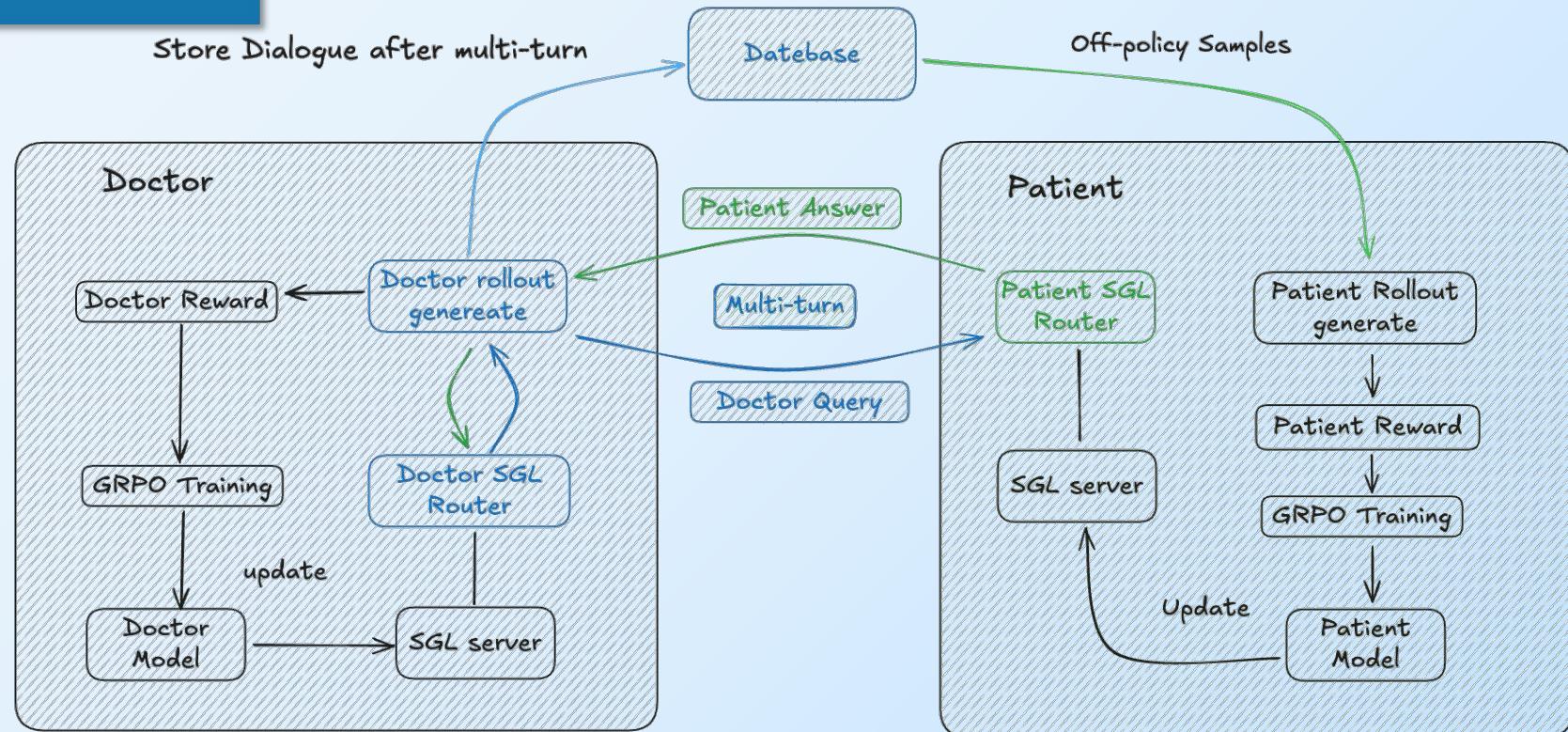
单轮/多轮的生成本质上没有任何区别!





Multi Agent Training

■ 蚂蚁集团 AQ Team^[1]



Doctor Reward :
information gathering
format compliance
agnostic accuracy

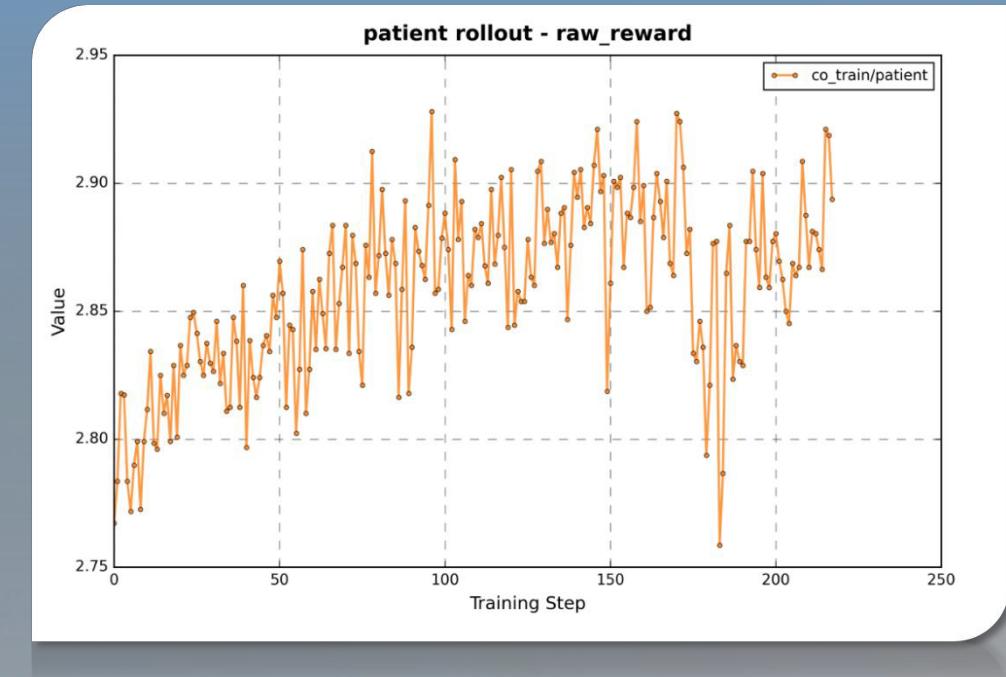
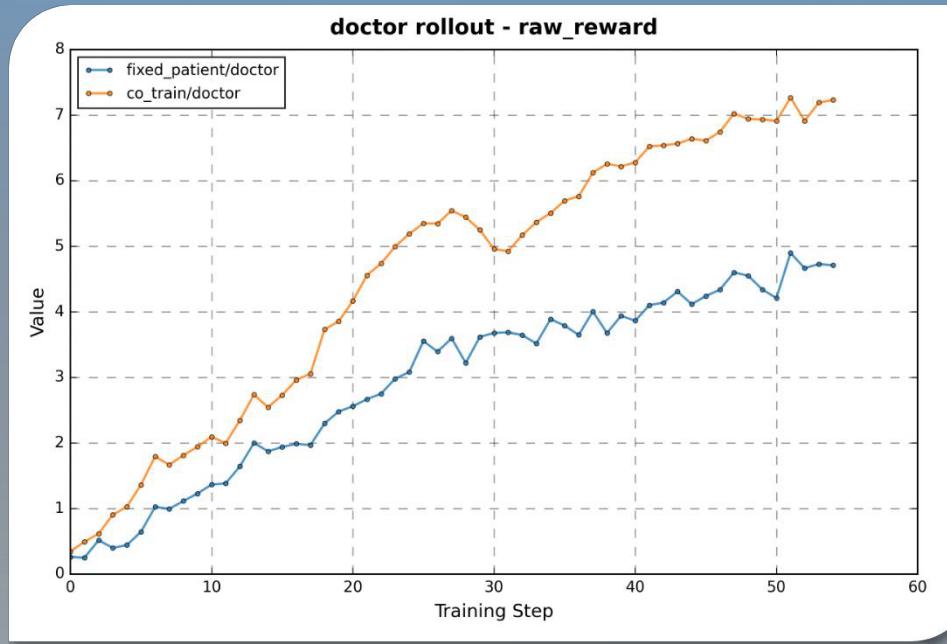
Patient Reward :
information control,
response completeness
factual conflict

[1] <https://aq-medai.github.io/MrlIX/>





Multi Agent Training



结论

在多智能体训练期间，相比使用固定的单一的病人模型，协调训练中的医生模型reward显著增加，而患者的reward在训练过程中也会同步上升。

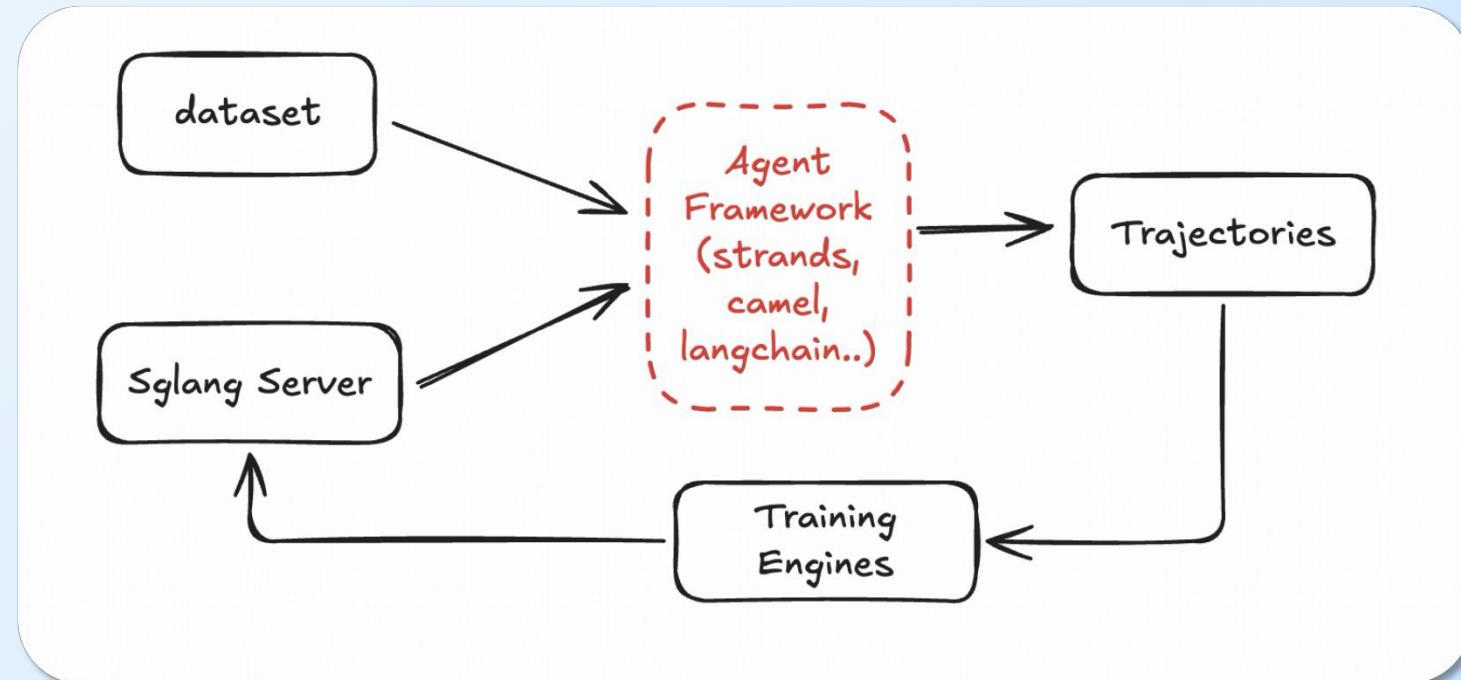


RL based on Agent Framework

现有 Agent 框架均基于server url进行轨迹生成，和slime设计完美吻合

将基于框架的推理代码变为 slime 的 rollout function
即可

slime 已经有基于 strands-agent 框架^[1]的 RL 代码^[2]，
理论上可迁移至任何 Agent Framework



[1] <https://github.com/strands-agents/sdk-python>

[2] <https://github.com/THUDM/slime/tree/main/examples/strands-agents>





对于 Agentic RL，仅仅可以训练就够了吗？



从能训->能训好需要什么？

动机：Agentic rl 通常会引入外界的
observation->训练不稳定^[1]



框架需要支持解决各类的mismatch! [1][2][3]

- Retokenization mismatch
- Training-inference mismatch^{[2][3]}

[1] Dong G, Mao H, Ma K, et al. Agentic reinforced policy optimization[J]. arXiv preprint arXiv:2507.19849, 2025.

[2]<https://fengyao.notion.site/off-policy-rl>

[3] When Speed Kills Stability: Demystifying {RL} Collapse from the Training-Inference Mismatch



Retokenization Mismatch

如果使用纯文本进行多轮交互，模型输出 token 经过解码变为文本，随后该文本重新被编码进行下一轮生成。前后 token 可能不一致

南京市

长江大桥

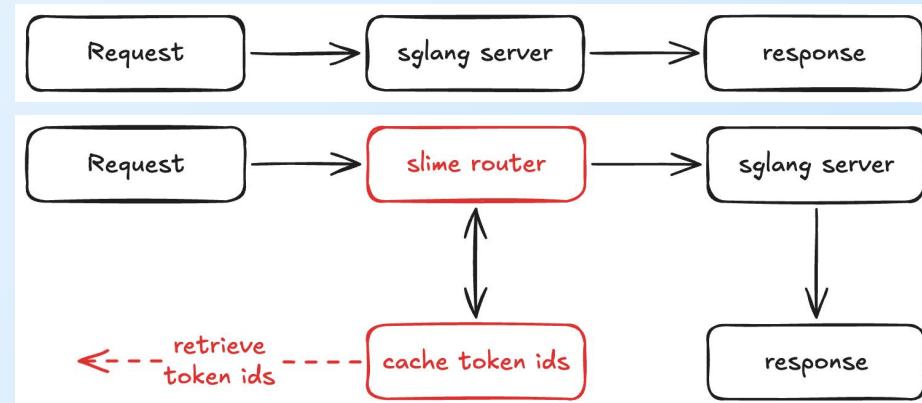


南京

市长

江大桥

用户需要手动保存 token ids ->实现了 slime router 自动 cache token ids





Training–Inference Mismatch

什么是 Training–Inference Mismatch?

- 推理框架 (SGLang) 输出的概率分布 ≠ 训练框架 (Megatron, FSDP) 输出的概率分布。
- 该特性会对训练的稳定性产生影响，从而可能导致训练崩溃。



产生原因



算子的不确定性

由于输入batch size/算子 tile size 的不同
可能导致概率分布不同

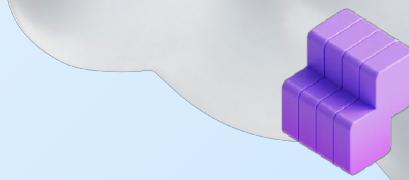


数据类型的不同

量化操作引入误差（比如BF16训练，FP8推
理）



MOE模型在训练和推理引擎上激活的
专家可能不同，引入更加严重的
mismatch问题



解决方案1：Importance Sampling (IS)

从算法上减少 mismatch 对于训练稳定性所带来的影响

$$\mathbb{E}_{a \sim \pi_{\text{SGLang}}(\theta_{\text{old}})} \left[\frac{\pi_{\text{Megatron}}(a, \theta_{\text{old}})}{\pi_{\text{SGLang}}(a, \theta_{\text{old}})} \cdot \text{policy-objective}(\pi_{\text{Megatron}}, \theta_{\text{old}}, \theta, a) \right]$$

在框架上我们该支持什么？

不同的IS计算方式

Token level, Sequence level, geometry level..

不同的处理方式

限制权重大小，裁剪梯度...

slime 为了保持易用性和可扩展性，支持自定义不同 IS 函数以适应不同算法需求！^[1]

- 提供 training log, inference log 等数据
- 用户需设计自己的 IS 算法计算最后的损失

[1]https://github.com/THUDM/slime/tree/main/examples/train_infer_mismatch_helper

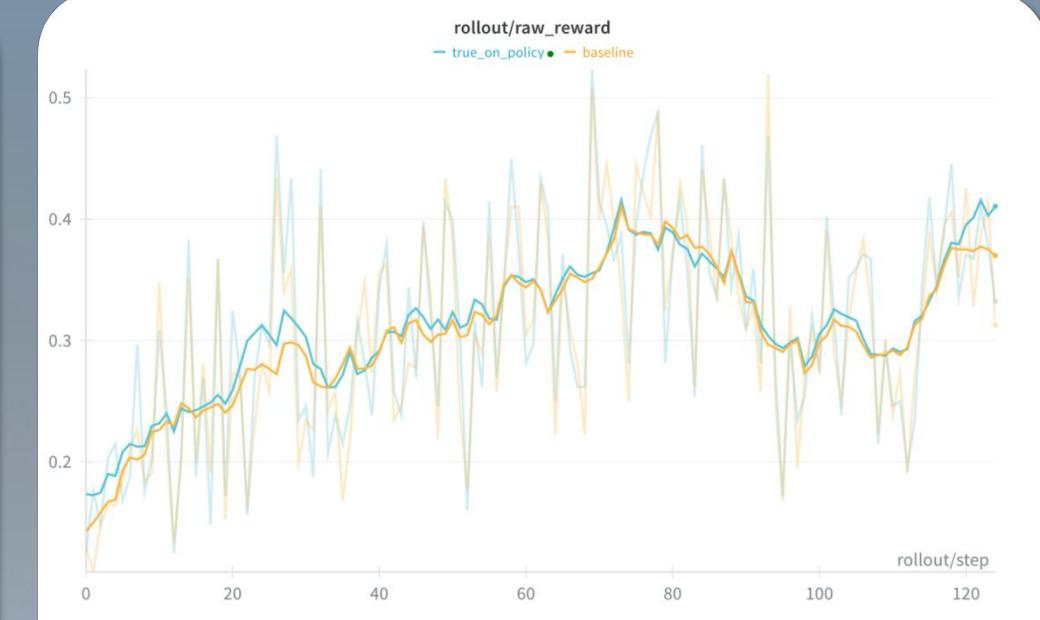
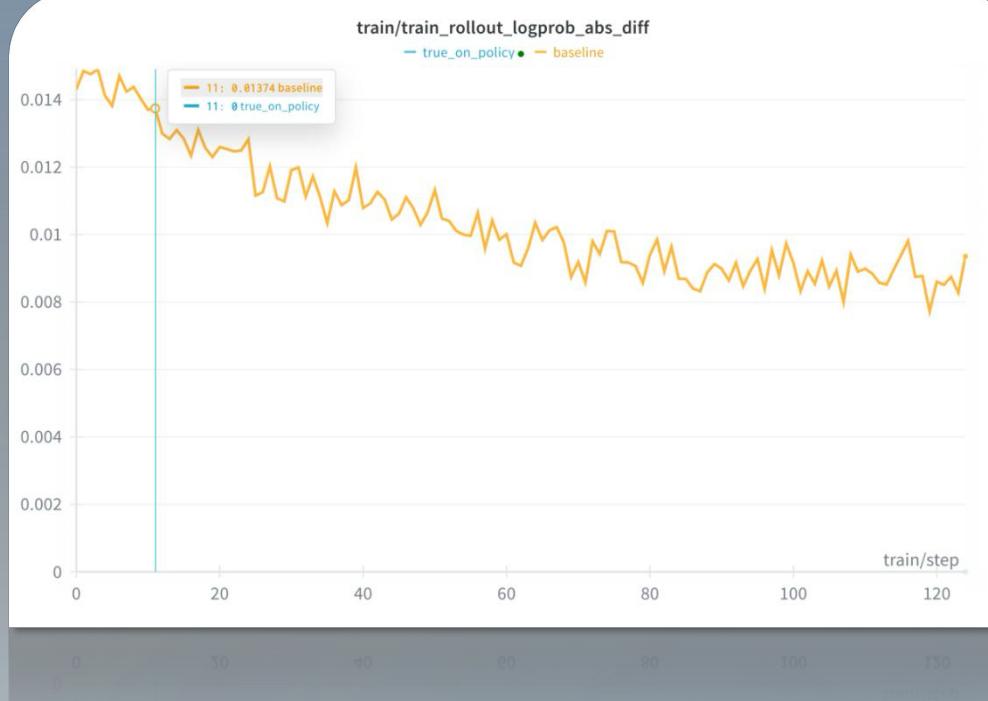




解决方案2: True on-policy (算子级对齐, 前向/反向算子一致)

算法上的设计终究不是完美解决方案, 能否对齐training engine和inference engine的精度呢?

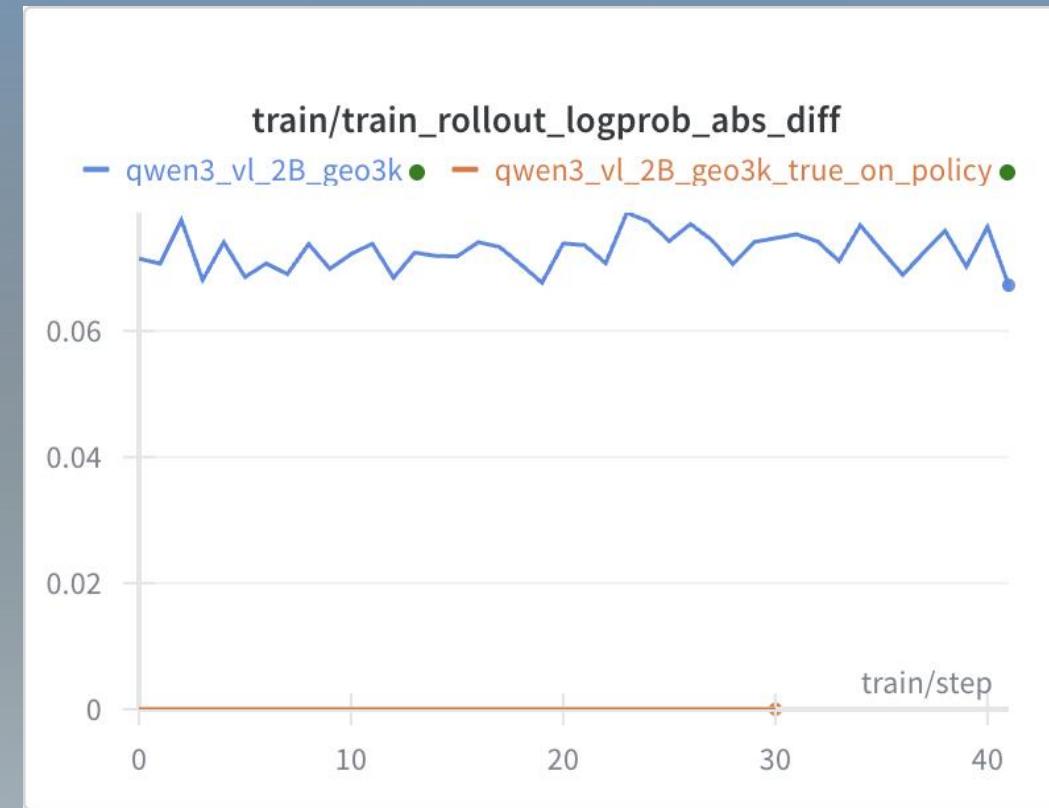
- slime 在训推一致取得了突破性成果
- SGLang deterministic rollout + FSDP Backend (Dense Model)^{[1][2]}



[1] <https://github.com/THUDM/slime/pull/566>
[2] [https://github.com/sql-project/sqlang/pull/12058](https://github.com/sgl-project/sqlang/pull/12058)

多模态模型的训推对齐

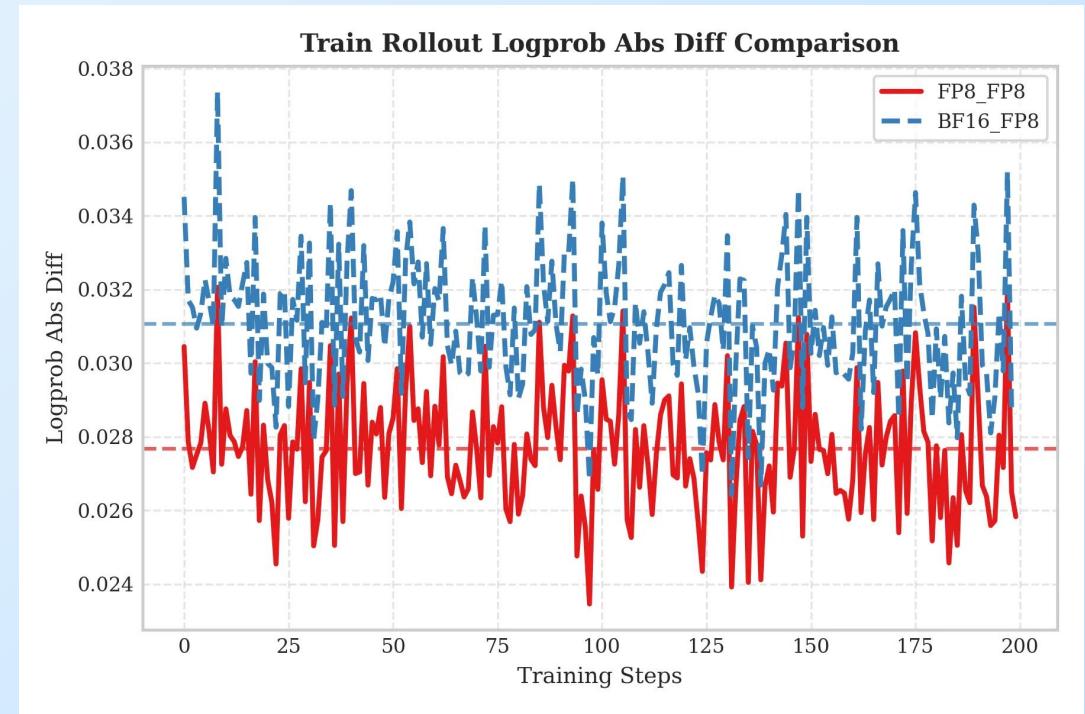
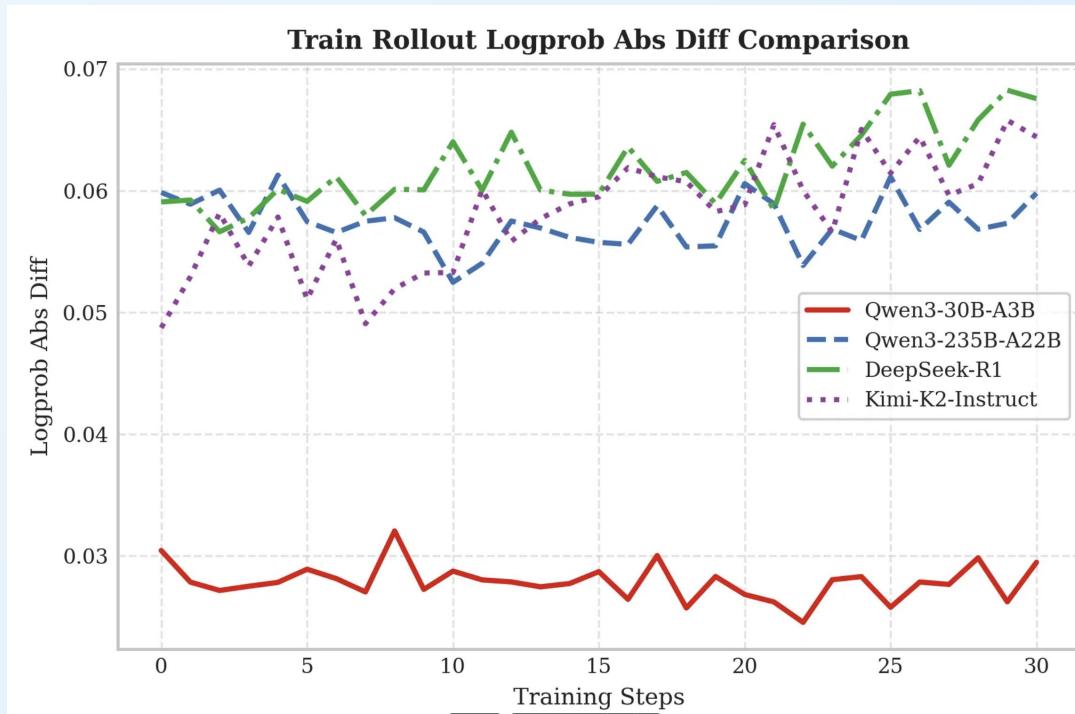
True on-policy training with Qwen3-VL dense model on FSDP backend.





解决方案3：FP8 RL 训练全流程

我们实现了在 RL 中完全使用 FP8 进行采样 (rollout) 和训练 (training)。实验表明，MoE 模型规模越大，BF16 训练 FP8 采样的训推差异越明显。训推过程统一使用 FP8 有效消除了量化误差导致的不一致性，提升了 RL 训练的速度和稳定性。^[1]



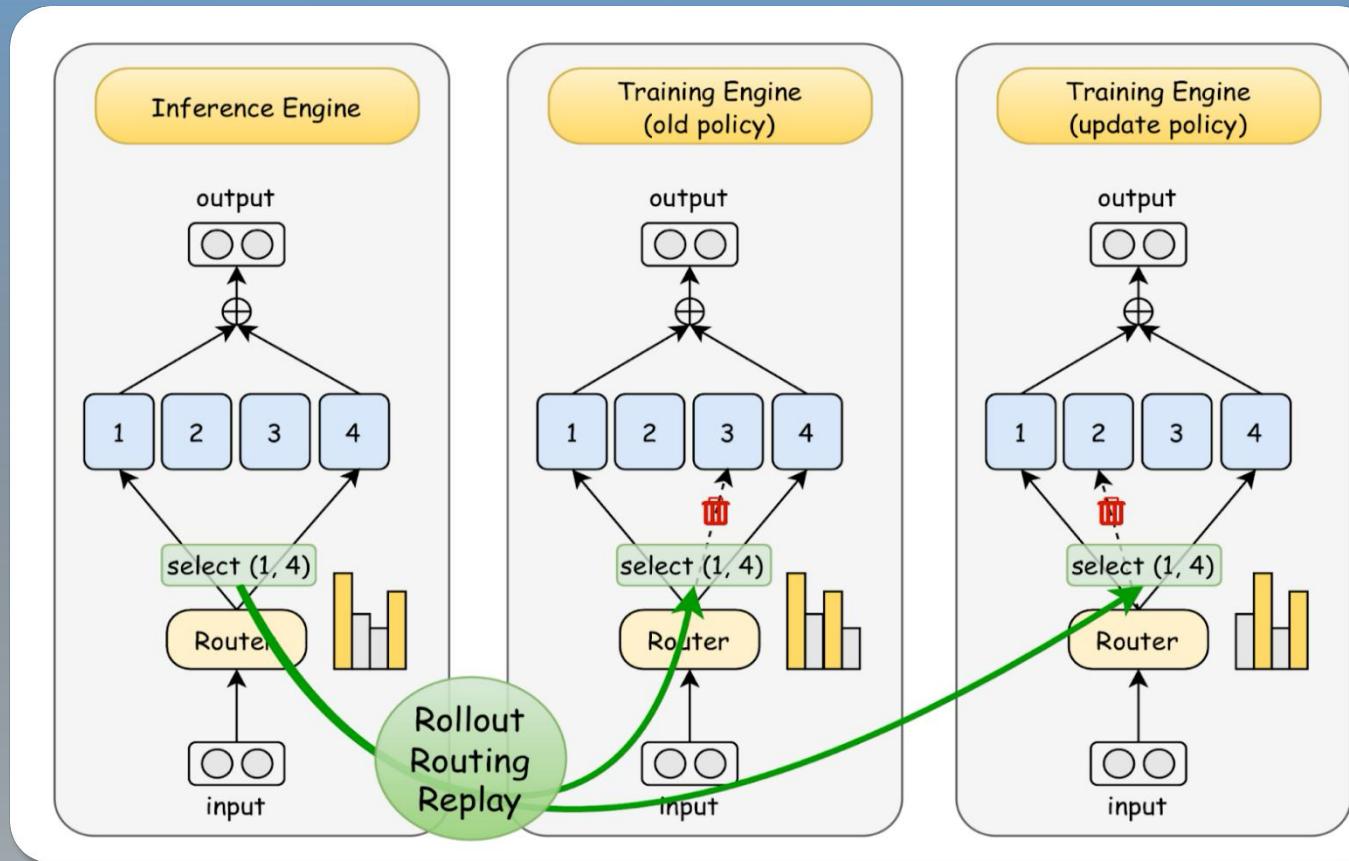
本工作由InfiXAI 团队、蚂蚁集团 AQ 团队、SGLang RL 团队及 slime 团队联合完成。



[1]https://github.com/THUDM/slime/blob/main/examples/low_precision/README.md



解决方案4: rollout routing replay



routing replay

- 在计算old logp和logp时确保专家一致^[1]。

rollout routing replay

- 在计算old logp和logp时都是用rollout时激活的专家,DeepSeek 3.2 的训练使用了该技术^{[2][3]}, 足以证明其重要性。

[1] Zheng C, Liu S, Li M, et al. Group sequence policy optimization[J]. arXiv preprint arXiv:2507.18071, 2025.

[2] Liu A, Mei A, Lin B, et al. DeepSeek-V3. 2: Pushing the Frontier of Open Large Language Models[J]. arXiv preprint arXiv:2512.02556, 2025.

[3] Ma W, Zhang H, Zhao L, et al. Stabilizing MoE Reinforcement Learning by Aligning Training and Inference Routers[J]. arXiv preprint arXiv:2510.11370, 2025.



解决方案4：Rollout Routing Replay

- 01 slime + SGLang 已经实现了 Rollout Routing Replay (训练DeepSeek 3.2的条件)
- 02 SGLang 在推理时会记录每一个 token 激活的专家 id
- 03 slime 会根据记录的专家 id 进行训练
- 04 显著地降低了Training-Inferencediff![1]



(True on-policy on MOE model)

[1] <https://github.com/THUDM/slime/pull/715>

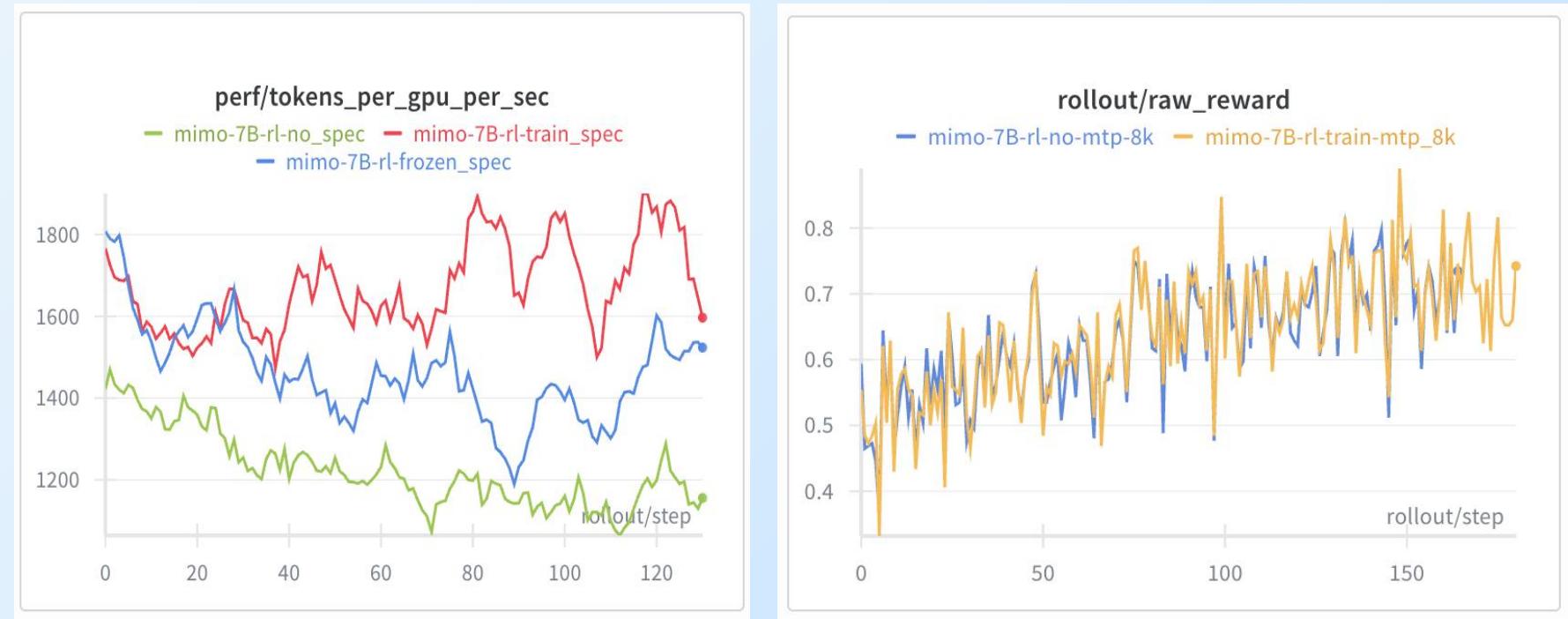


加速方案5：Speculative Decoding In Reinforcement Learning

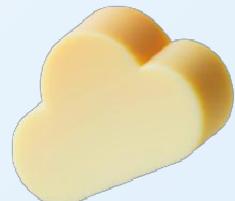
将 speculative decoding 引入到了 RL 的采样流程中，在 batch size 合适的情况下，采样速度得到了显著提升；并且，draft model 也会在训练过程中更新。相较于冻结 draft model 的做法，accepted length 持续维持在较高水平，产生长期稳定的正收益。

训练 spec
效率提升达 25%
冻结 spec

rollout 性能：训练 spec 对比不开启 spec 整体有 35% 左右的提升，对比冻结 spec 有 14% 的提升。这一差距在训练后期愈发明显，训练 spec 对比冻结 spec 效率提升达到 25%。



[1] <https://github.com/zhaochen yang20/Awesome-ML-SYS-Tutorial/blob/main/rhf/slime/spec/readme.md>





TODO



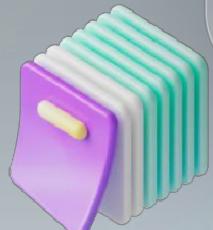
Agent 侧

- 多任务RL 训练支持
 - 如何更加方便的传 config
 - 如何平衡不同 task 速度
 - 更好的监控指标
- VLM Multi-turn RL (GUI Agent)
- VLA RL



Infra 侧

- slime router for openai endpoint
- True on policy (MoE Model)
- QAT INT4 Rollout in RL
- prefill decode disaggregation in RL



Thanks



OpenCloudOS在AI场景上的 支持与应用

讲师姓名- 程馨

2025-12-20

- **OpenCloudOS背景介绍**
- **AI场景软硬件生态支持**
- **云原生AI场景上的容器优化**
- **AI for OpenCloudOS**

- **OpenCloudOS背景介绍**

OpenCloudOS 社区技术路



现有国内外路线无法满足腾讯业务自用及云上国产化、商业化需求



满足腾讯业务复杂度需求
满足业务对新技术的需求

满足业务定制化需求
资源成本优化需求



基础设施稳定可持续



提供安全增强OS底座

全链路自研
服务器操作系统发行版

发行版

L0
上游社区



KVM



OpenJDK



Apache

OpenSSL

Cryptography and SSL/TLS Toolkit

APACHE

Software Foundation

其他用户态软件
上游开源社区

L1
源社区

Linux内核+GNU
组件+其他开源组
件

L2
商业版

L1版本加固+
技术支持/服务

L3
社区稳定版

经企业大规模验
证，基于L2的重
编译或定制优化

L4
衍生发行版

基于L3优化改
造的发行版

OpenCloudOS

Stream 23

TencentOS

Server 4



OpenCloudOS V9

OC9的再发行版本

打通L1社区上游版本到L3社区稳定版本研发，提供创新先进的基础软件栈和稳定高性能操作系统平台服务

- 打通产业上下游，持续提供自主L1-L3演进能力，全链路自主可控，创新先进、独立开放，全面解决CentOS断供风险，避免购买商业版本带来的软件授权成本，聚焦为最终用户提供经过大规模企业级验证、稳定可用的服务操作系统发行版（L2、L3版本）；
- 持续投入版本研发和稳定性测试，继承早期发行版核心自研特性，安全稳定
- 结合公司内近10年操作系统技术积累，提供高效经济的自研特性加持，满足定制需求

OpenCloudOS for AI 生态总览



AI 开箱即用	Agent容器	推理/训练容器	计算卡SDK容器	CodeBuddy命令行集成	自动构建镜像平台	腾讯HAI平台开箱即用镜像	AI镜像小型化	AI镜像/模型下载加速
-------------------------	-------------------------	-------------------------	--------------------------	--------------------------------	--------------------------	-------------------------------	-------------------------	-----------------------------

AI 软件 支持 生态	AI 模型	DeepSeek系列	QWEN系列	Hunyuan系列	Llama系列	GPT系列	Gemini系列
	Agent / MCP	MCP框架 (LangGraph等)	安全沙箱 (语言级沙箱 / 进程级沙箱 / 系统级 沙箱)	Agent运行时 [Browser use / Mobile use / Computer use / 代码块执行]			
	部署管理	部署框架 (Ollama/Dynamo/ LangChain等)	加速和优化 (3FS DeepEP/ Taco-LLM / FlexKV 等)	集群部署 (nccl支持 k8s支持等)			
	推理和训练	Pytorch	TensorFlow	vLLM	SGLang	Paddle Paddle	TensorRT-LLM
	AI SDK	开发工具 (IDE 编译器等)		Runtime 多版本 多版本Python/Nodejs等		多媒体编解码器 (图片/音视频/物理引擎)	
					pypi软件重新适配		

AI 硬件 支持 生态	NVIDIA CUDA	AMD ROCm	INTEL OneAPI	昇腾 CANN	海光 DTK	沐曦 MXMACA	昆仑芯	寒武纪	摩尔 线程	天数 智芯	泰尚 TopsRide r
----------------------	----------------	-------------	-----------------	------------	-----------	--------------	-----	-----	----------	----------	---------------------

- AI场景软硬件生态支持

AI软件栈支持面临更复杂局面

应用	智能体应用	web chat	图像识别	工业应用	
Agent	mcp	Browser-use	Computer-use	Mobile-use	
推理加速库	Flashinfer	FlashAttention	TensorRT-LLM	TinyAttention	
推理和部署框架	Vllm	Sglang	LLama-cpp	triton	LightRuntime
模型训练框架	Colossal-AI	DeepSpeed	Megatron-LM		
计算库 / 通信库 / 数据库	MAGMA	BLAS	OpenCL	MPI	Milvus
深度学习框架	 PyTorch	 TensorFlow	 mxnet		
GPU配套 Toolkit	CUDA	ROCM	CANN		

AI 软件需求量大

热门AI软件栈 > 2000，大部分基于Ubuntu开发；部分软件对TencentOS 4兼容性较差，需要海量的测试和适配工作；

维护/开箱 工作量大

作为操作系统方，既要长期维护这些软件，又要为用户提供开箱支持；集成镜像各自制作，重复工作，难以复用。

组件依赖复杂

框架依赖底层深度学习库、编译器、硬件驱动等，栈层复杂且版本兼容性要求严格

版本更新成本高

框架、模型、底层库均高速迭代，需要持续跟进更新，维护成本高

GPU/计算卡支持



通过rpm软件源支持Nvidia-driver 525~580 多版本，并进行持续更新；支持CUDA 11/12/13等多版本，并提供相应的标准容器。



支持沐曦C500/C550/C588/C600/N260系列计算卡。



支持海光BW1000/ K100_AI 系列计算卡。



原生支持 AMD ROCm 计算框架，支持至 MI300系列计算卡。



对于天数智芯智铠系列推理卡，支持其 Iluvatar Corex计算SDK，并提供对应的容器。另外提供适配的VIIm等软件栈。



支持华为昇腾910B/310P系列计算卡。



支持Intel CPU推理One API框架，正在对接Gaudi系列计算卡适配。



支持寒武纪MLU590系列加速卡。



原生支持紫霄C100 / C200计算卡，支持 TopsRider SDK，并支持VIIm和LightRuntime，提供对应的容器。

AI软件栈自动验证自动打包

方案

- 搭建流水线，预计覆盖跟踪、安装测试、运行测试和自动打包镜像4个流程，目前已实现跟踪、安装测试和运行测试；
- 结合github api，让 agent 分析软件 readme 给出安装命令，自动测试安装，并反馈安装结果；
- 利用 agent，自动生成执行用例，并反馈结果。

结果

- 自动化安装成功率 **64.5%** (635/930)；
- 针对验证出的不兼容软件，进行专门适配，提高 OpenCloudOS 9开箱使用体验。



支持1000+热门AI软件

BASE LEVEL

ToolKit



DeepLearning



CORE LEVEL

Libraries



Training



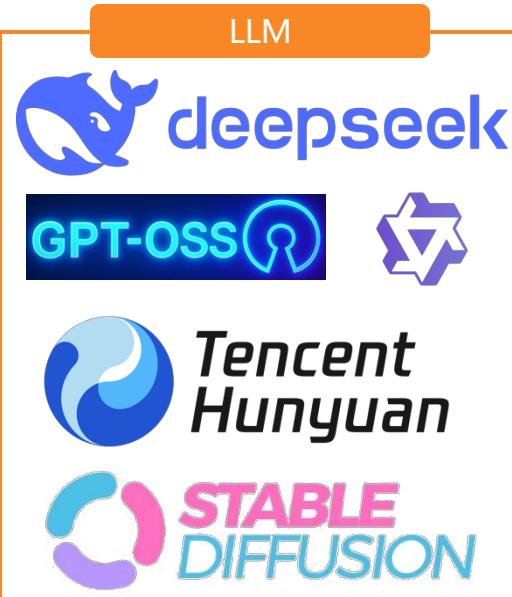
Inference & Deploy



Language



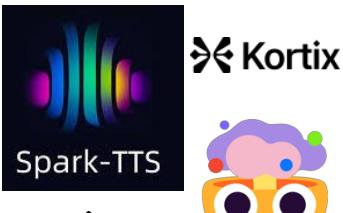
LLM



APP LEVEL



Dify-



AI Agent 运行环境支持



系统接口

[Browser use](#)

[Computer use](#)

[Mobile use](#)

[Code Runtime](#)

沙箱

语言级

[Python沙箱](#)

[Javascript沙箱](#)

进程级

[Docker](#)

[Nsjail](#)

[Freejail](#)

系统级

[gVisor](#)

[kata](#)

AI软件栈开箱支持



容器镜像	版本覆盖	概述
pytorch	PyTorch: 2.8.0, CUDA: 12.9.1	<ul style="list-style-type: none">基于cuda-devel:12.9 镜像构建镜像内同时包含 PyTorch 官方 example 仓库用于演示
SGLang	SGLang:0.5.2, CUDA:12.8	<ul style="list-style-type: none">基于cuda-devel:12.8 镜像构建内置了SGLang官方示例代码库，可用于快速验证环境和学习
vllm	vllm:0.10.2, CUDA:12.8	<ul style="list-style-type: none">基于cuda-devel:12.8 镜像构建内置了vllm官方示例代码库，可用于快速验证环境和学习
TensorRT-LLM	TensorRT-LLM:1.0.0, vllm:0.10.2, CUDA:12.8	<ul style="list-style-type: none">基于cuda-devel:12.8 镜像构建内置了TensorRT-LLM官方示例代码库，可用于快速验证环境和学习
transformers-deepspeed	Transfomers:4.56.2, deepspeed:0.16.8, CUDA:12.9.1	<ul style="list-style-type: none">基于cuda-devel:12.9 镜像构建内置了一个基于bert-base-uncased微调的情感微调
cuda-runtime	Cuda 11.8、12.0-12.9全系列、13.0	<ul style="list-style-type: none">包含cuda所有.so库、可执行文件及数据文件适用于轻量级生产部署
cuda-devel	Cuda 11.8、12.0-12.9全系列、13.0	<ul style="list-style-type: none">除cuda-runtime镜像内容外，另包含.a静态库及.h头文件适用于代码编译/模型训练
cuda-toolkit	Cuda 11.8、12.0-12.9全系列、13.0	<ul style="list-style-type: none">除cuda-devel镜像中内容外，另包含nvidia性能分析工具适用于模型性能调优及算法研究
OpenCloudOS计划后续持续加深对AMD ROCm的容器化支持		

- 云原生AI场景上的容器优化

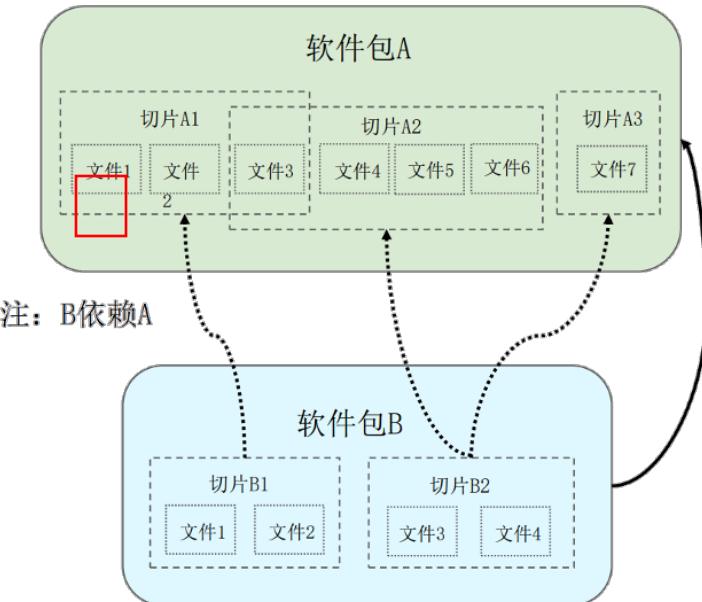
OC Slimtrace – AI容器镜像小型化



OC Slimtrace通过对文件粒度的切片以及动静态结合的文件依赖分析，对容器镜像进行瘦身，从根源上解决AI容器镜像过大导致拉取启动慢的问题。

chisel切片：

将rpm包粒度的依赖精细化为文件粒度的依赖，一个切片包含一个或多个文件，以及这些文件（来源于rpm包）的最小依赖的集合，构建最小化运行环境

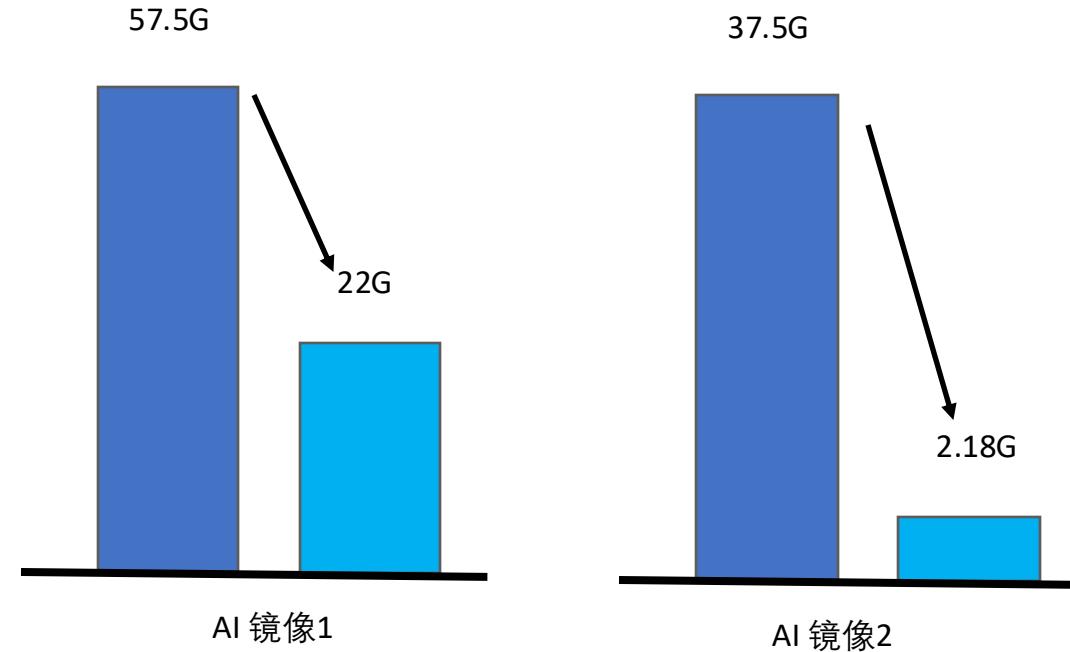


busybox基于rpm的依赖大小为24M，基于切片的依赖大小为3.2M

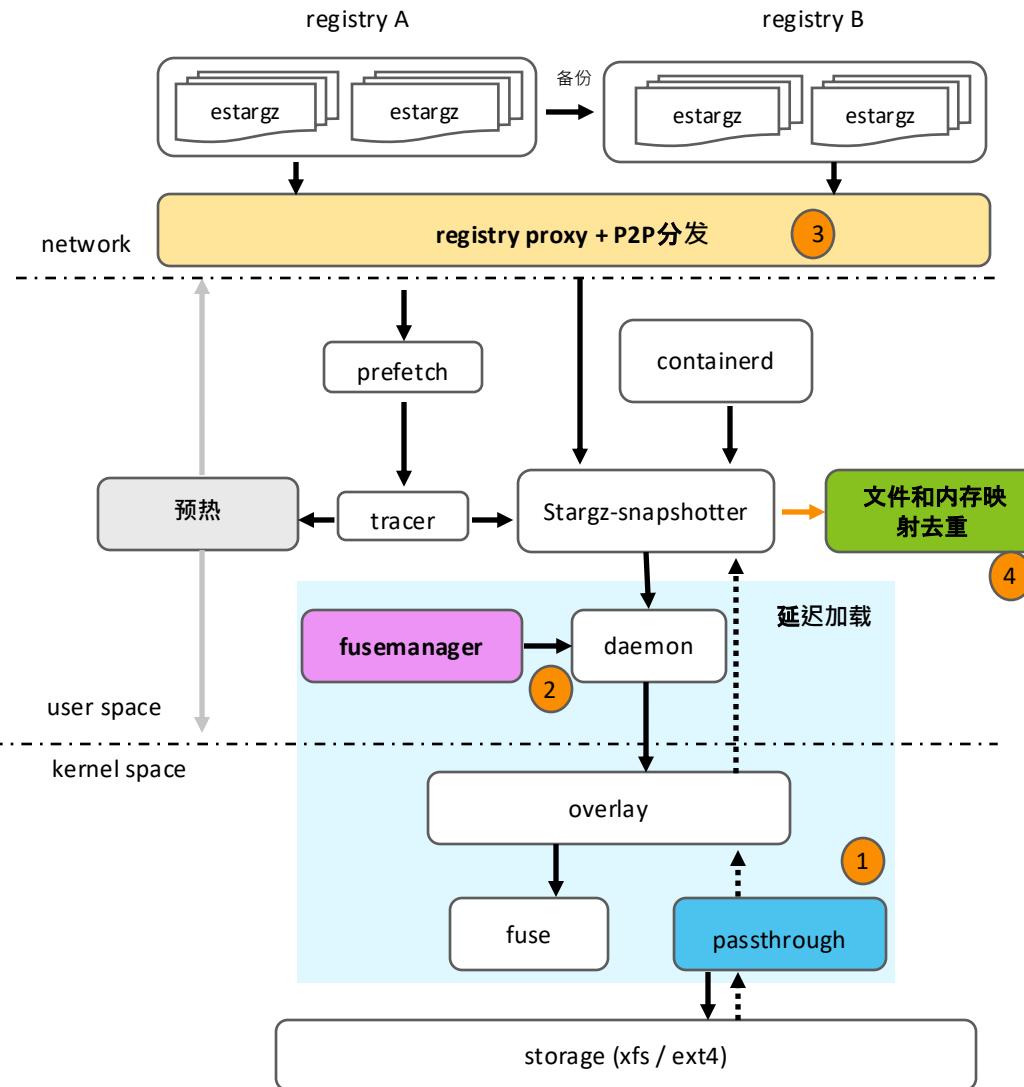
软件包依赖分析：

通过动静态结合分析技术，解析软件源码及二进制文件，自动化精准捕获文件依赖关系。初步完成AI场景常用的python软件包的依赖分析

依赖分析效果：



OC Flip - 容器镜像加速



主要问题

- 镜像拉取时间占容器启动时间的 **76%**, 但拉取的镜像仅有 **6.4%** 的数据被读取
- 镜像分层机制去重粒度较大, 重复文件最高达 **90%**
- AI 场景下镜像和模型过大, 受限于中心仓库带宽, 集群冷启动时下载速度慢, 耗时长

解决方案

- ✓ 懒加载
- ✓ 镜像文件去重
- ✓ P2P 加速镜像和模型文件分发

容器镜像加速- 懒加载

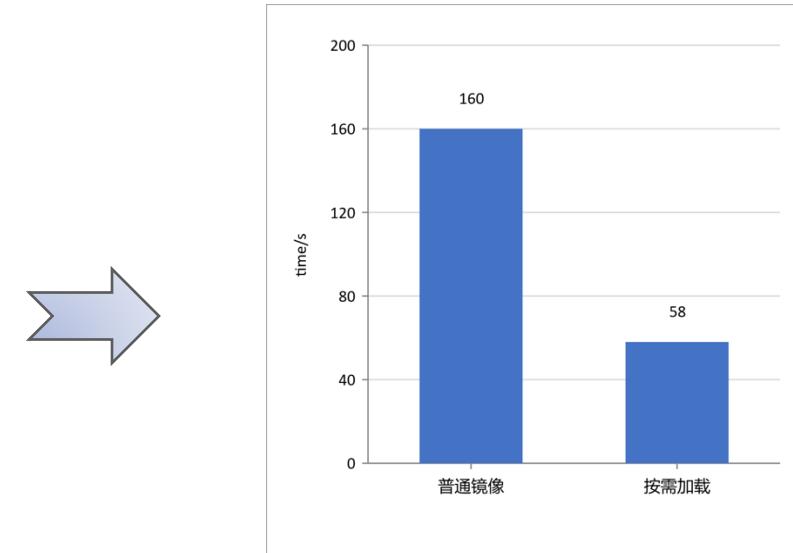
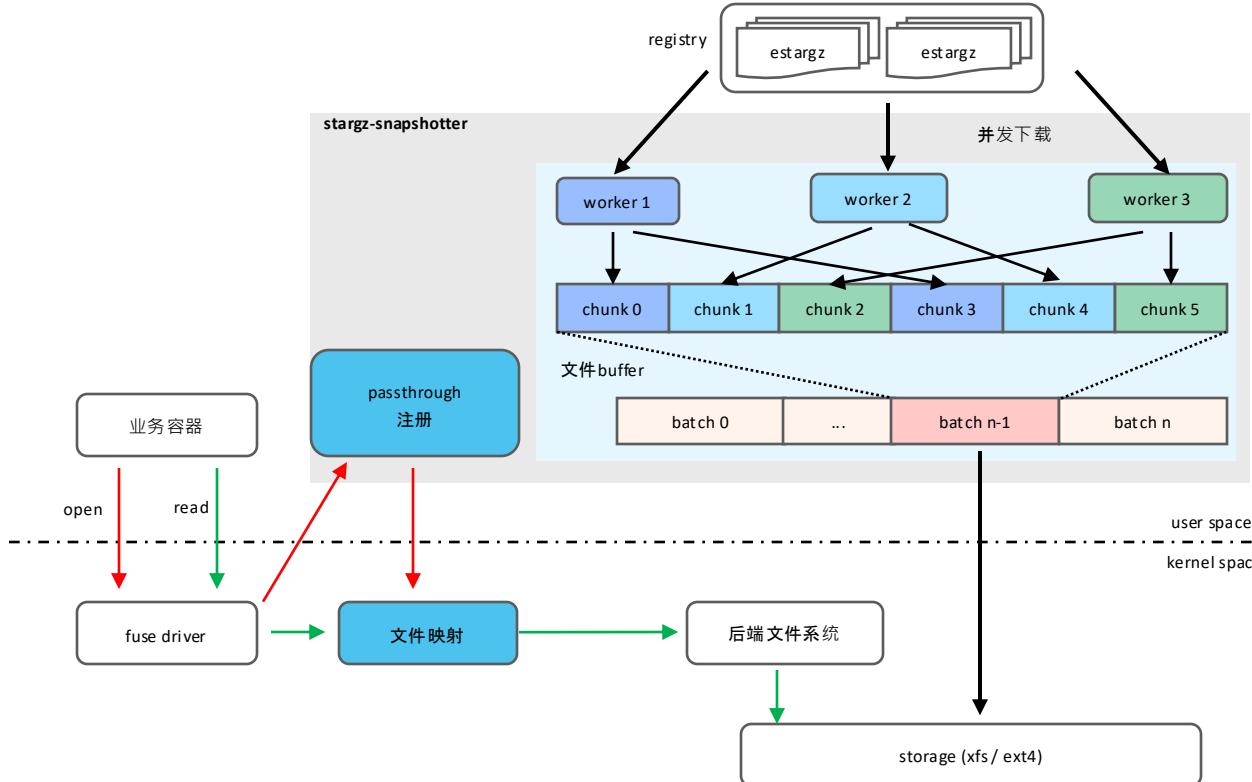


fuse 架构开销大, AI **大体积** 镜像冷启动耗时长
传统镜像格式转换为按需加载格式耗时较长
AI 场景按需预取精度差



考虑 OCI 兼容性, 选型 stargz-snapshotter 方案, 并进行了一系列优化:

- passthrough 配合并发和缓存落盘提升冷启动性能
- 优化解压, 拷贝, 并发度等关键路径, 耗时下降50%
- 优化预取, 增加监控 oci-hook 等关键阶段, 性能提升20%



50G AI 镜像sglang场景冷启动 (下载+服务运行) 时间缩短 **60%**

容器镜像加速 – 镜像文件去重

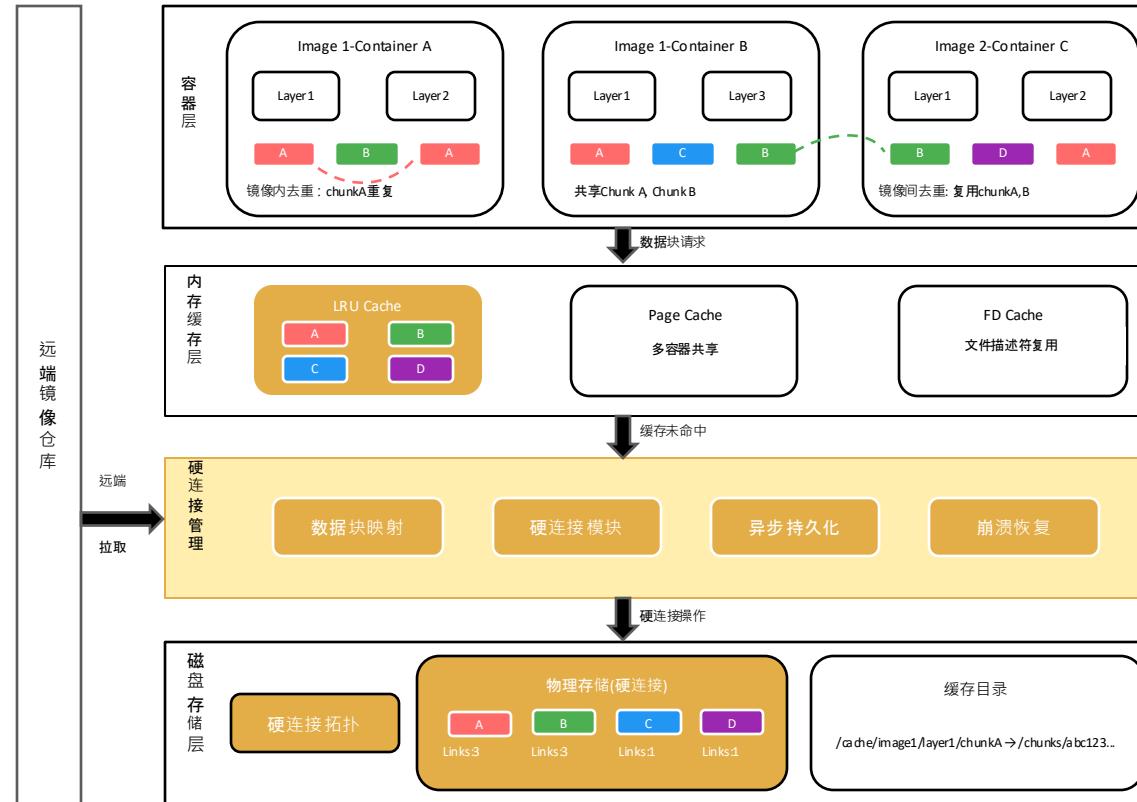


磁盘冗余：相同文件内容跨层 | 层内重复存储
缓存冗余：相同内容无法共享页缓存
I/O 冗余：跨层 | 层内访问触发重复读取操作
网络冗余：相同数据块重复远端拉取

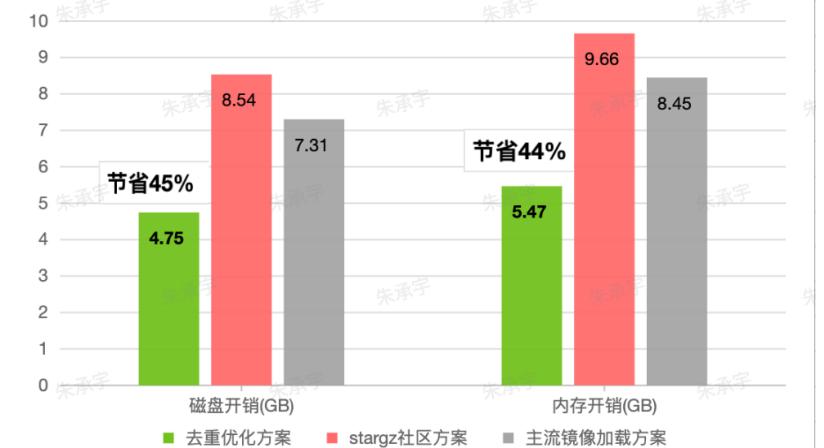


增加映射机制，减少相同内容的数据块的重复IO读写

用hardlink共享相同数据块的pagecache，减少内存空间占用



镜像方案	磁盘开销(GB)	内存开销(GB)	磁盘占用比	内存占用比
去重优化方案	4.75	5.47	1 (基准)	1 (基准)
stargz社区方案	8.54	9.66	1.80	1.77
主流镜像加载方案	7.31	8.45	1.54	1.55



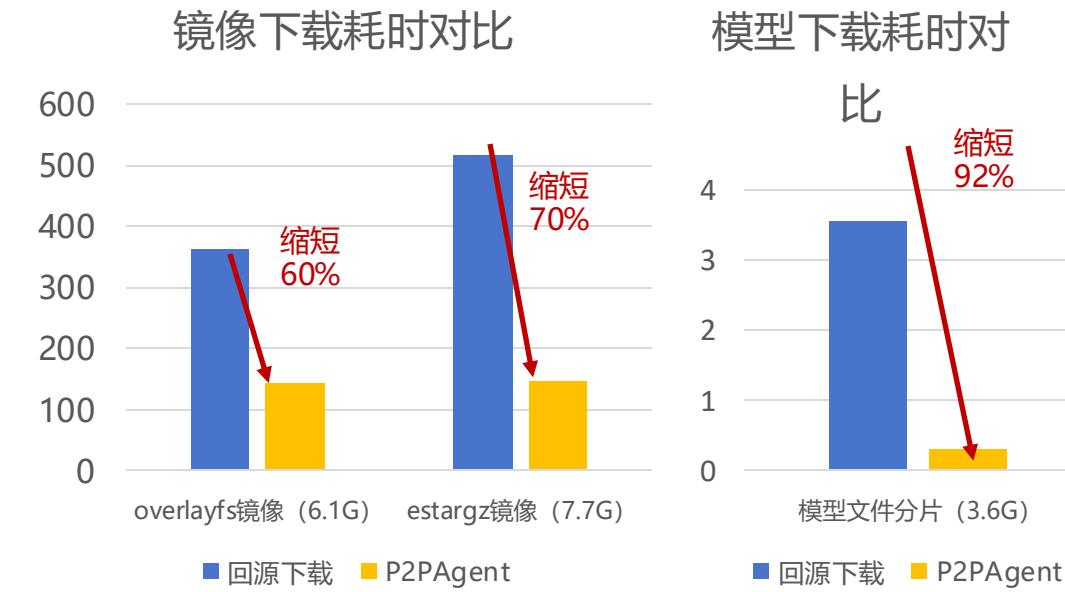
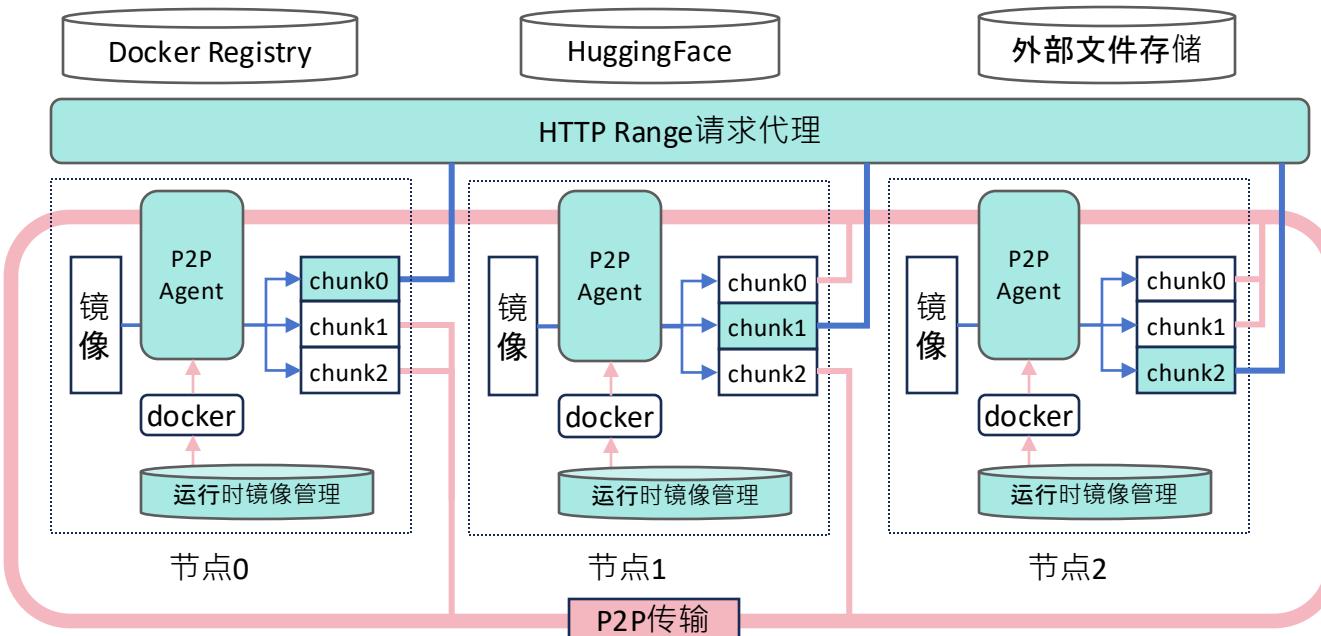
磁盘和内存开销降低 45%

OC Flip - 容器镜像加速

传统P2P需要完整下载后才能启动P2P传输，不支持镜像按需加载
P2P传输的镜像层需要额外存储
文件传输采用BT协议，节点资源控制能力弱，影响业务进程
标准TCP/IP带宽无法满足大模型文件的高速传输需求



支持分片并发和乱序下载，通过代理Range请求兼容按需加载
复用运行时管理的镜像，降低存储占用
支持节点资源监控、限速
通过RDMA、GDR技术，缩短模型在集群内分发耗时



- AI for OpenCloudOS

AI for OpenCloudOS



AI辅助操作系统开发维护

AI辅助自维护

上游监控
RPM-TRACKER

软件包自维护
PkgAgent

PR变更总结及审核

AI辅助开发

源码自动打包
并制作对应容器镜像

容器镜像对标替换

OCAI智能助手：AI辅助操作系统管理

集成系统管理能力

主机信息查询

RPM包信息查询

性能数据分析

tai-agent

系统预装，
终端集成AI能力

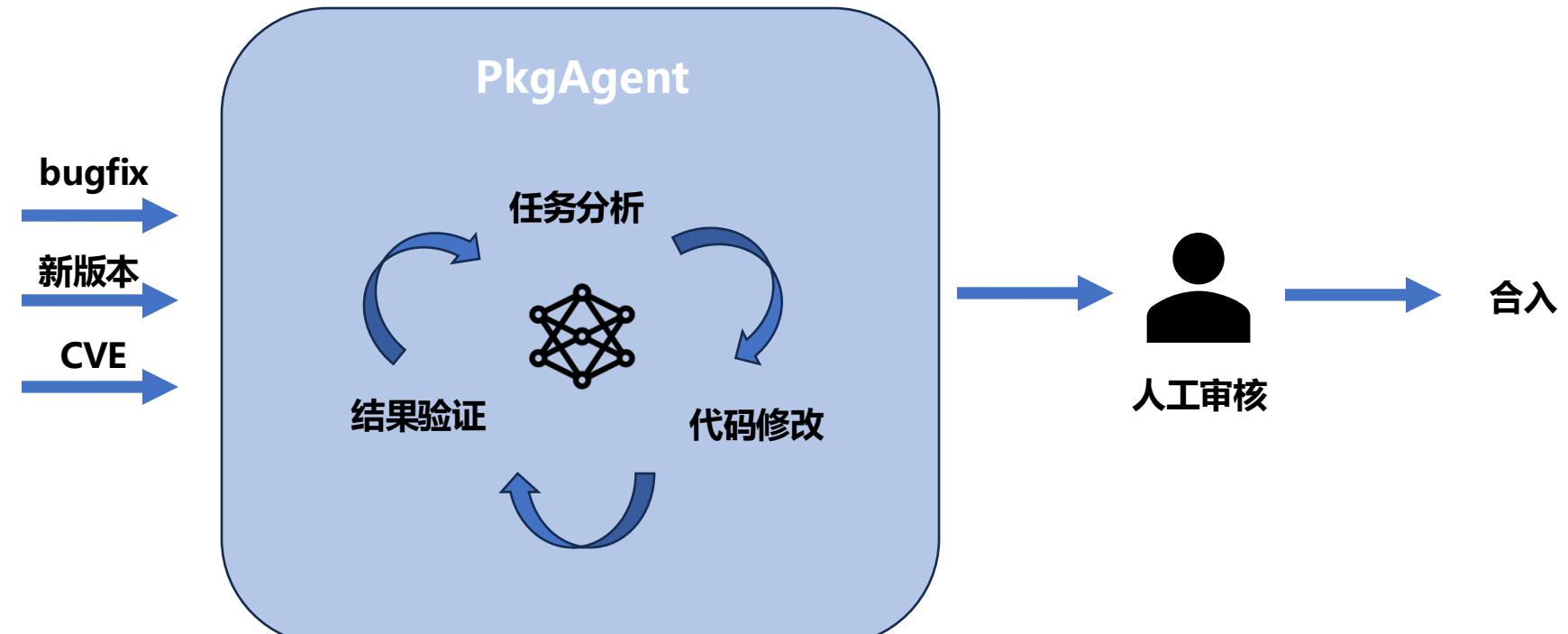
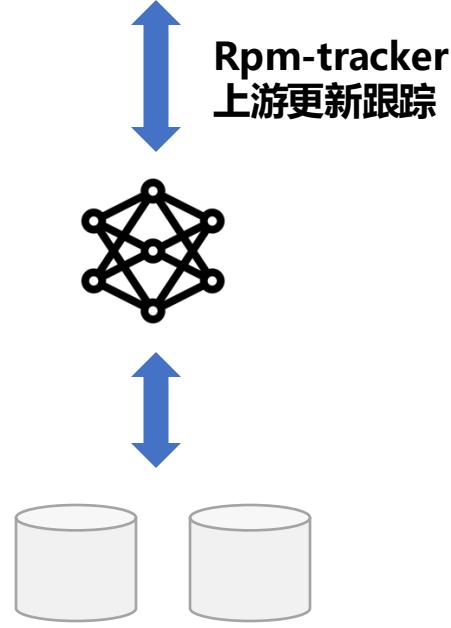
集成OpenCloudOS
官方知识库

Chat、命令、代码生成

AI辅助，提升开发效率，提高研发质量

集成OpenCloudOS官方知识库，帮助运维人员快速获取所需信息

软件包自维护工作流重塑- PkgAgent



软件包自维护工作流重塑- PkgAgent

➤ 小成本撬动大节省，让maintainer专注于技术本身

	时效	人力成本	质量
人工处理	7day	6h	准确度依赖maintainer对软件包的了解度
Agent处理	0.5day	0.5h	大语言模型知识面更广，对代码理解能力强



8000+软件包

日均commit数500+

日均3.3个漏洞修复



节省人力 **346人/天**

时效提升 **93%**

•一键安装：

```
yum install ocai-agent
```

•Chat模式：在命令窗口中与OCAI进行问答

```
[root@VM-0-120-opencloudos ~]# 怎样拉取一个opencloudos9的minimal docker镜像到本地
- assistant:
思考过程：首先，用户的问题是：“怎样拉取一个opencloudos9的minimal docker镜像到本地”。资料部分为空，所以我需要基于我的知识来回答。我是OpenCloudOS系统的智能助手，通过ocai-agent命令行工具提供帮助。
要拉取 OpenCloudOS 9 minimal Docker 镜像到本地，请按以下步骤操作：
1. **确保 Docker 已安装**
运行 `docker --version` 检查 Docker 是否可用。若未安装，请先安装 Docker 引擎。
2. **拉取镜像**
使用以下命令拉取镜像（假设官方镜像名为 `opencloudos/opencloudos:9-minimal`）：
```
bash
docker pull opencloudos/opencloudos:9-minimal
```
- 如果标签不存在，尝试 `docker pull opencloudos/opencloudos:9` 或查阅 [OpenCloudOS 官方镜像列表](https://hub.docker.com/r/opencloudos/opencloudos) 确认正确标签。
3. **验证镜像**
拉取完成后，运行 `docker images` 查看本地镜像列表，确认 OpenCloudOS 9 minimal 镜像已下载。
如有网络问题，可配置 Docker 镜像加速器。
```

•CMD模式：根据用户指示生成可执行的shell命令

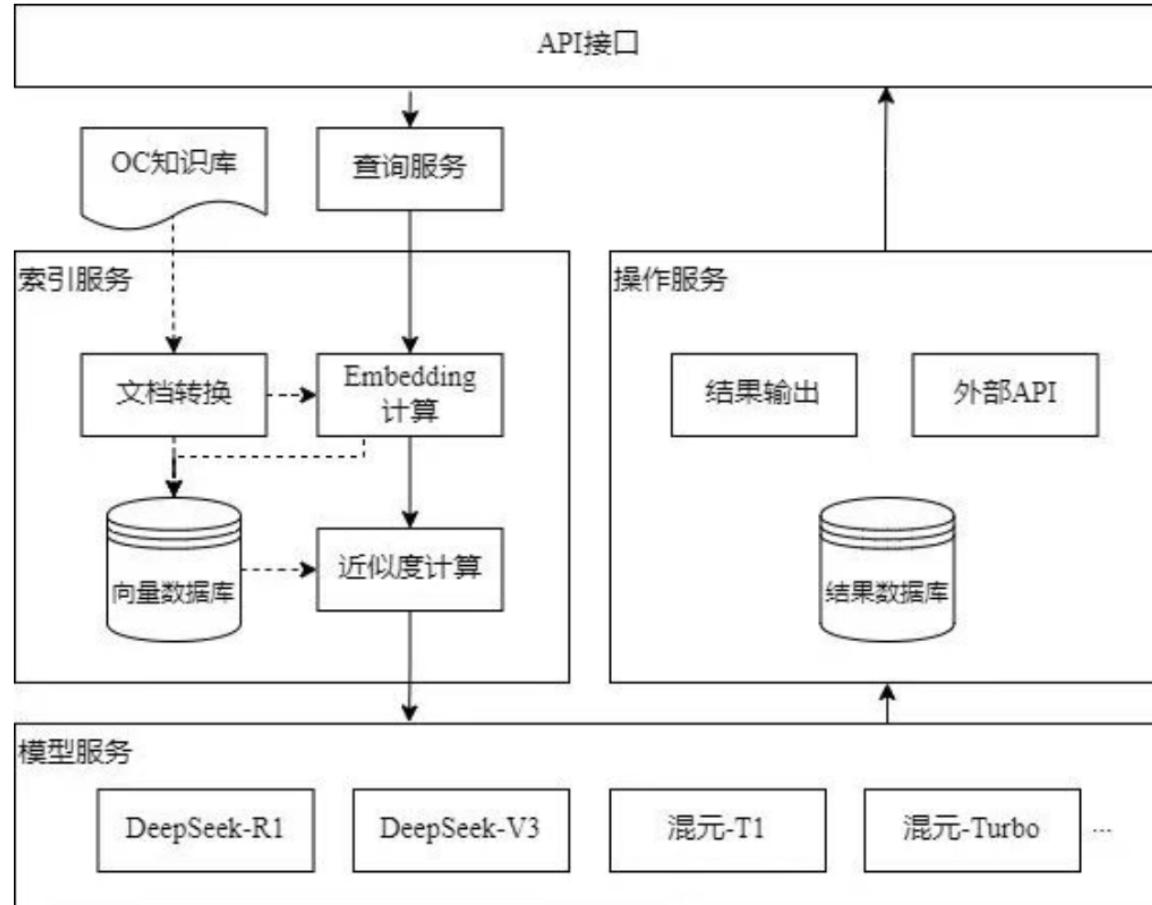
```
[root@VM-0-120-opencloudos ~]# ocai cmd "如何更新yum源"
? Choose a command: (Use arrow keys)
» sudo yum update
Edit command before execute
Exit

Total download size: 508 M
Is this ok [y/N]: n
Operation aborted.
ocai: command ends with non-zero return code: 1
? Would you like to resolve this error using AI chat? Yes
- assistant:
思考过程：首先，用户的问题是：执行 `sudo yum update` 后返回代码是 1，错误输出和标准输出都是 None。用户想知道为什么。
```

•Code模式：根据用户指示生成代码

```
[root@VM-0-120-opencloudos ~]# ocai code "用python实现快速排序"
def quicksort(arr):
    """
    快速排序算法实现
    """
    if len(arr) <= 1:
        return arr

    pivot = arr[len(arr) // 2] # 选择中间元素作为基准值
    left = [x for x in arr if x < pivot] # 小于基准值的元素
    middle = [x for x in arr if x == pivot] # 等于基准值的元素
    right = [x for x in arr if x > pivot] # 大于基准值的元素
```



多Agent编排

针对不同场景和数据源创建多个不同 Agents，如对话、命令生成、代码生成，并通过指定逻辑将多个Agents组合编排，从而使使用入口得以统一，多个Agents可以协同工作。

RAG

将OC知识内容向量化存储。用户提问将先进行向量检索和召回，再统一拼接成 prompt 提交给大模型进行总结和回答。

底座模型

OCAI智能助手在处理逻辑的各个阶段使用了 DeepSeek 和混元的各类大模型，包括但不限于 DeepSeek-R1、DeepSeek-V3、混元T1、混元Turbo 等。



THANK YOU!





OpenCloudOS
定制T恤*5



AMD定制遮阳帽*4



AMD定制马克杯*3

OpenCloudOS SGLang x AMD Meetup Shenzhen

2025.12.20 (周六)

13:30-18:00

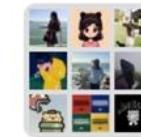
深圳市深南大道10000号

腾讯大厦2F多功能厅 (i)

主办方:



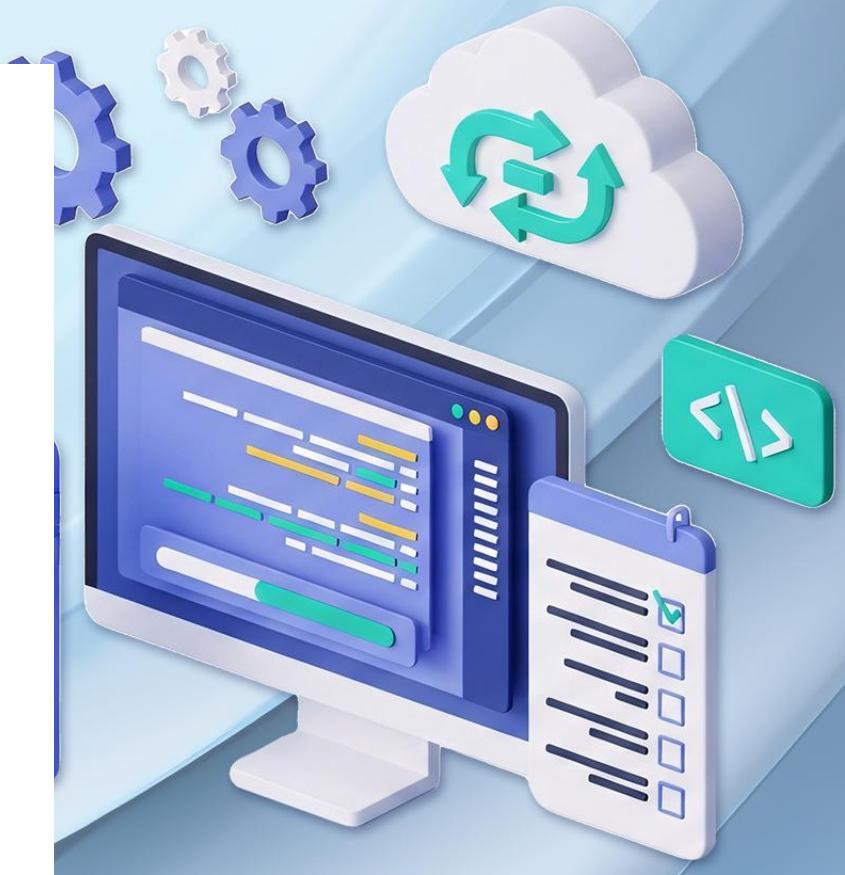
合作伙伴:



群聊: 深圳 SGLang Meetup
开发者群



该二维码 7 天内 (12 月 26 日前) 有效, 重新进入将更新

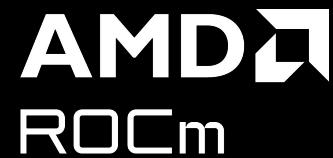




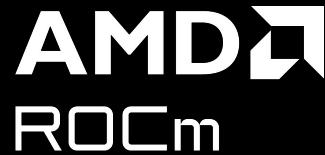
大规模分布式推理在AMD ROCm平台 上的最佳实践

演讲人: 何亚豪

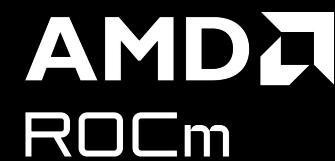




问题：在大模型MoE、尤其是大规模跨机场景下，系统瓶颈主要体现在哪里？ 算子？ 通信？

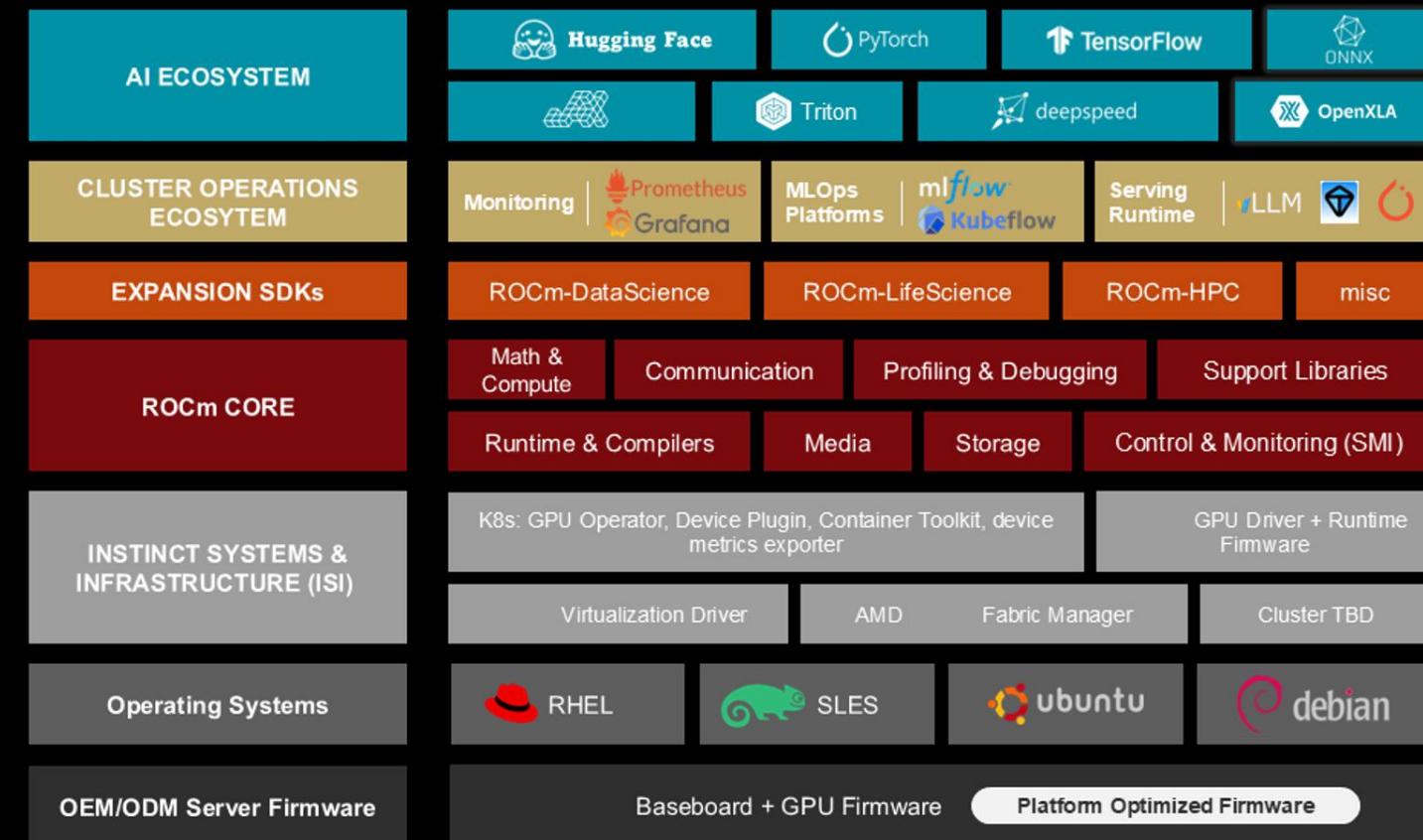


- 什么是ROCM平台
- ROCm 7 在计算、通信和调度层面，提供了哪些关键能力
- 这些能力是如何逐步集成进 SGLang社区，并支撑更大规模、更高效率的分布式推理or专家并行推理

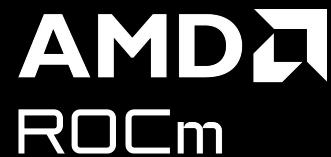


ROCM软件栈

 +



借助ROCM 生态， SGLang扩展了它在AMD平台的支持



ROCM7 - 生成式AI/大模型的全栈解决方案

AI 模型算法应用



开源社区的生态建设

算子通信库

编译器及其工具

AMD ROCm 软件栈

ROCM运行时

AMD
CDNA

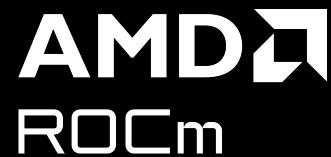


AMD
RDNA



实现从数据中心到消费端全覆盖的GPU平台





ROCM7 - 生成式AI/大模型的全栈解决方案

AI 模型算法应用



开源社区的生态建设

算子通信库

编译器及其工具

AMD ROCm 软件栈

ROCM运行时

AMD
CDNA

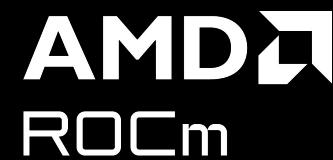


AMD
RDNA



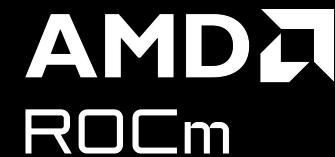
实现从数据中心到消费端全覆盖的GPU平台





ROCM7 核心库



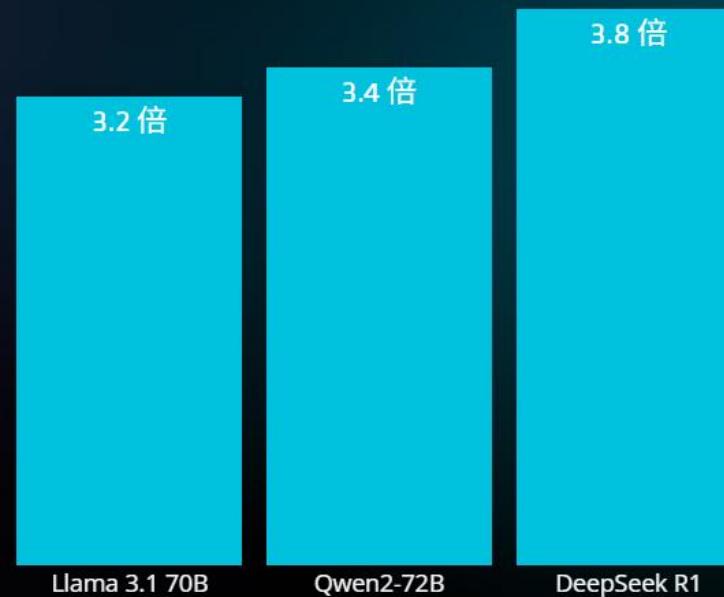


ROCM7 代际提升显著

代际性能提升

ROCM 7 对比 ROCM 6

3.5 倍 平均性能提升



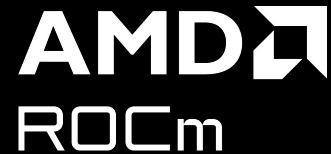
推理性能¹

3 倍 平均性能提升

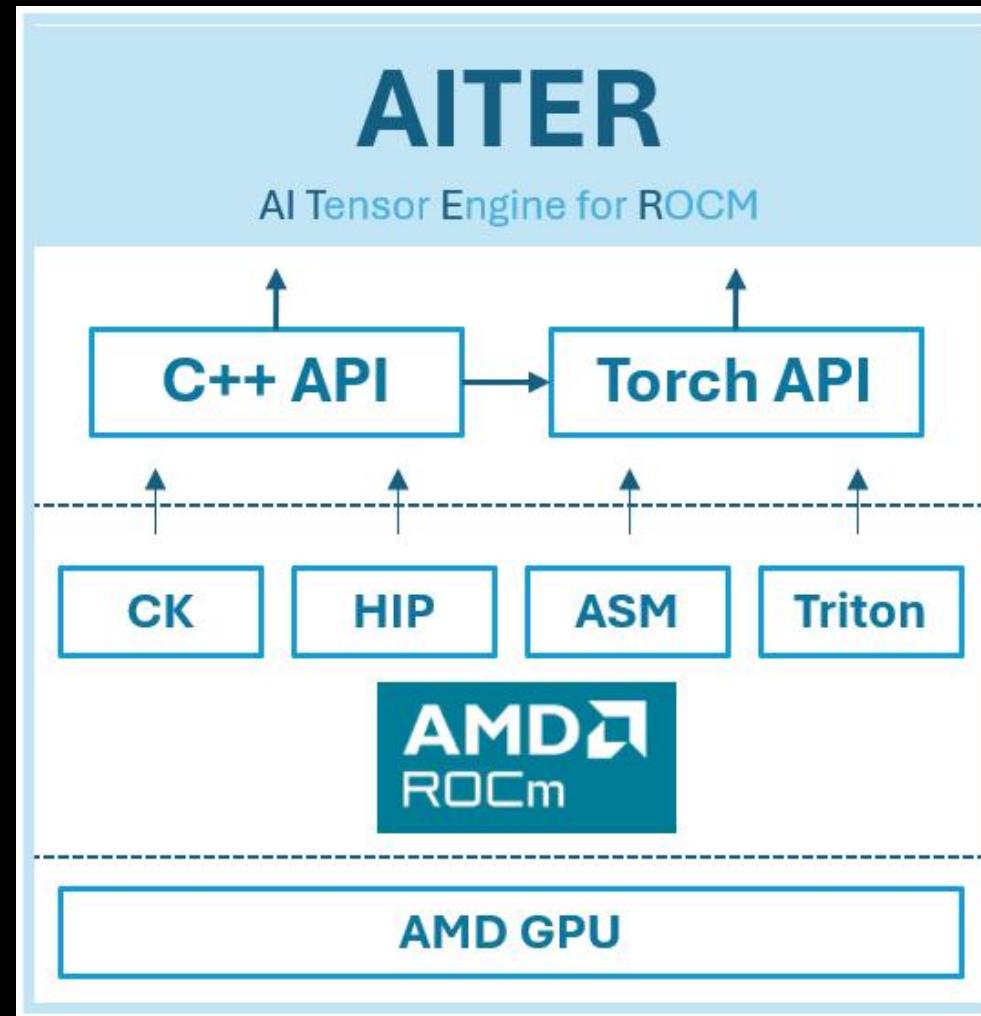


训练性能²

数据来源: AMD Advancing AI 2025



大规模专家并行在ROCM实践1：AITER 算子库



算子种类	具体实现
Prefill Attention	Fav3 FWD FP16/BF16 Fav3 BWD FP16/BF16 MLA FP16/BF16 FA FP8 FWD+BWD (Block-Scale) Chunked-prefill
Decode Attention	Paged Attention FP16/BF16 Paged Attention FP8 per-tensor quant Paged Attention FP8/INT8 with KV per-token quant KVCache Update & Rotary Batched Decoding MLA Decoding
Fused-Moe	Moe-Sorting kernel and tiling solution FP16/BF16 per-token Fused-Moe FP8/INT8 per-token Fused-Moe FP8 per-tensor Fused-Moe FP8/INT4 per-tensor Fused-Moe Fused-FFN
Low Precision Gemm	FP8 per-token/channel Gemm FP8 Block Scale Gemm INT8 weight-only Gemm
Normalization and Fusion	Layernorm+quant/shortcut RMSNorm+quant/shortcut
Custom Comm	AR/AG fused with normalization AR/AG quantized, Optimized hipgraph support

AMD 大规模专家并行在ROCM实践1：AITER 算子库赋能MoE模型推理加速

ROCM

- 示例1：集成AITER PTPC A8W8 GEMM
 - PR 链接: <https://github.com/sjl-project/sclang/pull/8555>
- 确保LLM FP8计算精度的同时，提升矩阵乘计算性能
 - 支持权重的预排 (preshuffle)
 - 取消对PyTorch版本的依赖
- 开启条件：
 - 根据对应的dynamic quant的权重quant config自动开启
- 示例2：集成AITER MLA FP8 Kvcache支持
 - PR 链接: <https://github.com/sjl-project/sclang/pull/13147>
 - AITER MLA 集成persistent模式，支持了FP8 Kvcache

Pull Request #8555: enable aiter gemm_a8w8_bpreshuffle for ptpc gemm

Merged by HaiShaw on Sep 6. 14 commits from Yuechguo:pr-main_enable_aiter_ptpc_gemm.

Motivation: enable aiter gemm_a8w8_bpreshuffle for gemm in per-token-per-channel quant computing.

Modifications:

- Modified the weight preloading part to support weight shuffle.
- enable aiter gemm_a8w8_bpreshuffle op as an alternative to torch._scaled_mm, it does not strictly depend on a specific version of PyTorch.

Accuracy Test:

```
USE_ROCM_AITER_ROPE_BACKEND=0 \ SGLANG_USE_AITER=1 \ CUDA_VISIBLE_DEVICES=0 \ python3 -m sclang.launch_server \ --model-path /pretrained_model/Qwen/Qwen3-32B-FP8-Dynamic \ --served-model-name Qwen3-32B-FP8-Dynamic \ --attention-backend aiter \ --host 127.0.0.1 \ --port 28000 \ --tp-size 1 \ --trust-remote-code \ --max-prefill-tokens 32768 \ --chunked-prefill-size 32768 \ --context-len 32768 \ --max-running-requests 512 \ --disable-radix-cache \ --page-size 64 \
```

TP=1:

Qwen3-32B-FP8-Dynamic	gsm8k score
origin (torch 2.7.1 + hipblas)	0.90 ± 0.01
aiter gemm_a8w8_bpreshuffle	0.91 ± 0.01

Pull Request #13147: Aiter fp8 kv cache

Merged by HaiShaw last week. 29 commits from kkHuang-amd:aiter-fp8-kv-cache.

Motivation: Support fp8 kv cache in aiter-backend of AMD. Aiter backend only support mla decode fp8 computation. Other attention function still do bf16 computation.

Modifications: Aiter backend and model runner.

Next actions:

- Support other attention function also do fp8 computation

Accuracy Tests:

```
sclang# python3 benchmark/gsm8k/bench_sclang.py --num-questions 2000 --parallel 2000 --port 8000 Downloading from https://raw.githubusercontent.com/openai/grade-school-math/master/grade_school_math/data/test.jsonl to /tmp/test.jsonl [01:05<00:00, 732kB [00:00, 39.1MB/s] 100%]
```

1319/1319
[01:05<00:00, 20.081it/s] Accuracy: 0.945 Invalid: 0.000 Latency: 65.804 s Output throughput: 1944.756 token/s

AMD 大规模专家并行在ROCM实践1： ALTER 算子库赋能MoE模型推理加速

ROCM

- 更多示例

- 为FusedMOE算子集成权重预排 preshuffle

- PR 链接: <https://github.com/sjl-project/sgl-lang/pull/12908>
- 显著增强 Deepseek-V3/R1 , Kimi K2等模型的性能

- 共享专家融合

- PR 链接: <https://github.com/sjl-project/sgl-lang/pull/12201>
- 实现更均衡、更高效的专家利用，DeepSeek V3的吞吐可以增加4.9%，TTFT延迟降低6.1%

- MLA 优化

- PR 链接: <https://github.com/sjl-project/sgl-lang/pull/13705>
- 为FP8 output projection新增了fused_flatten_fp8_group_quant
- 为shared expert算子提供了新的Triton融合

- BMM优化

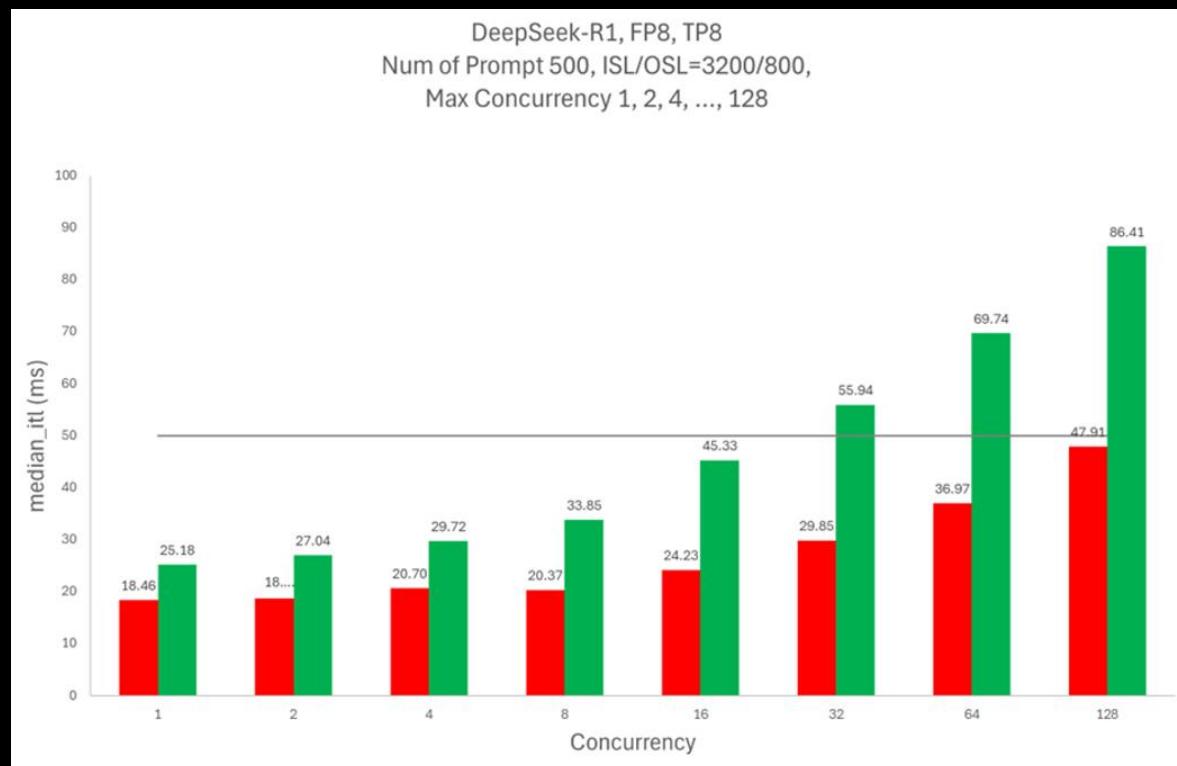
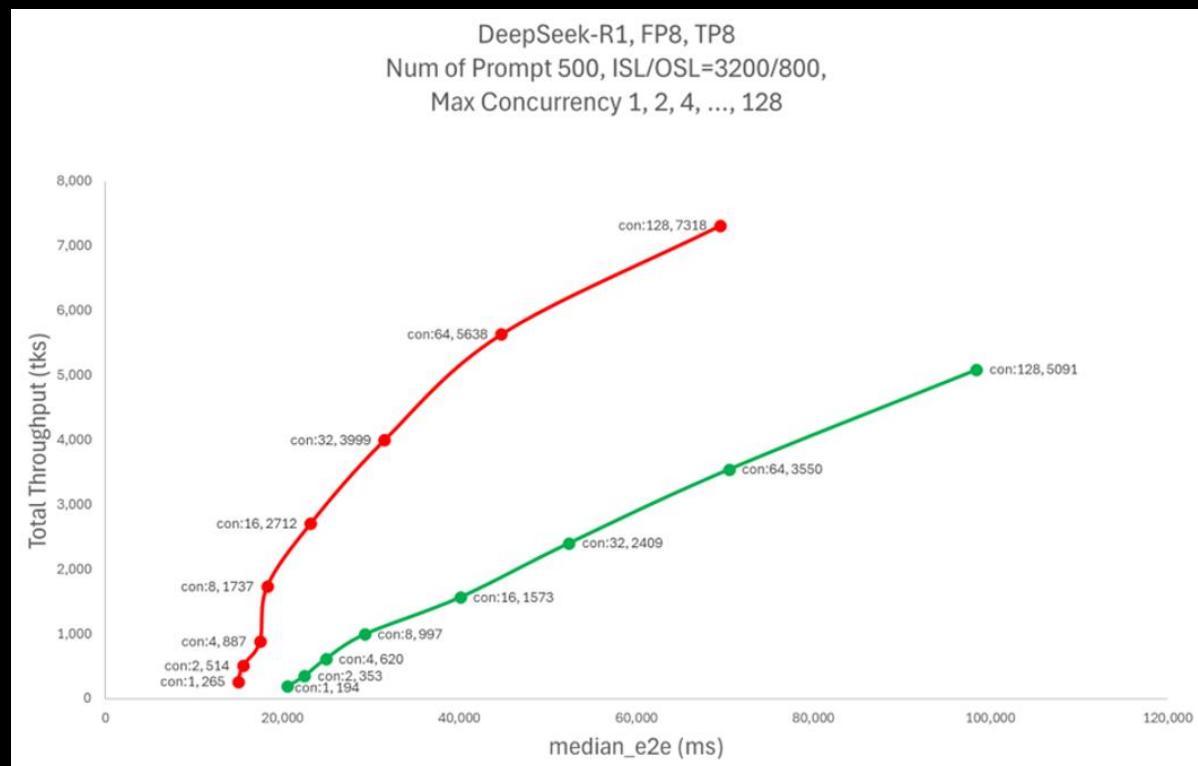
- PR链接: <https://github.com/sjl-project/sgl-lang/pull/13617>
- 使能batch_gemm_a8w8 和set_mla_kv_buffer

- RMSnorm量化融合

- PR链接: <https://github.com/sjl-project/sgl-lang/pull/12689>
- 使能RMSnorm和FP8量化

AMD 大规模专家并行在ROCm实践1：AITER 算子库赋能MoE模型推理加速

ROCm

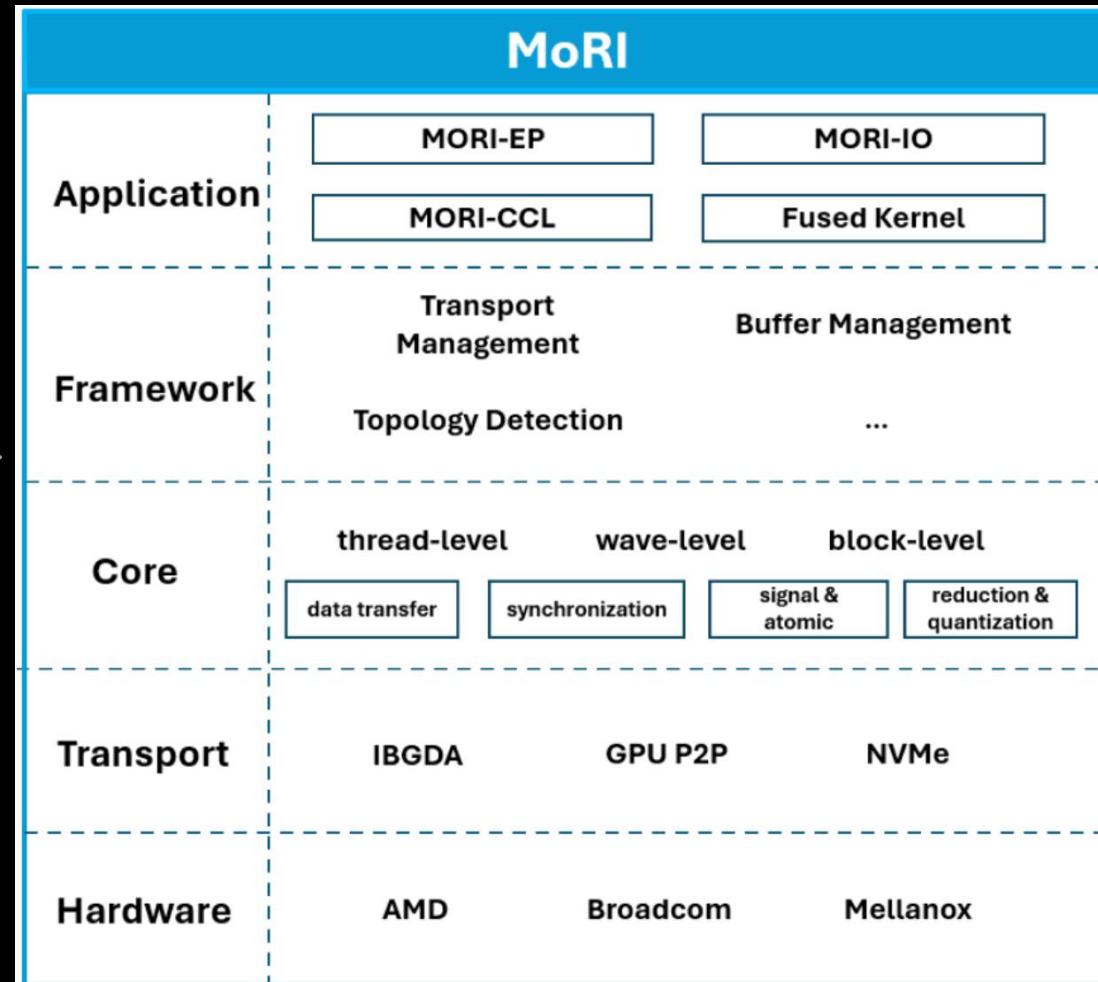


数据来源：<https://rocm.blogs.amd.com/artificial-intelligence/DeepSeekR1-Part2/README.html>

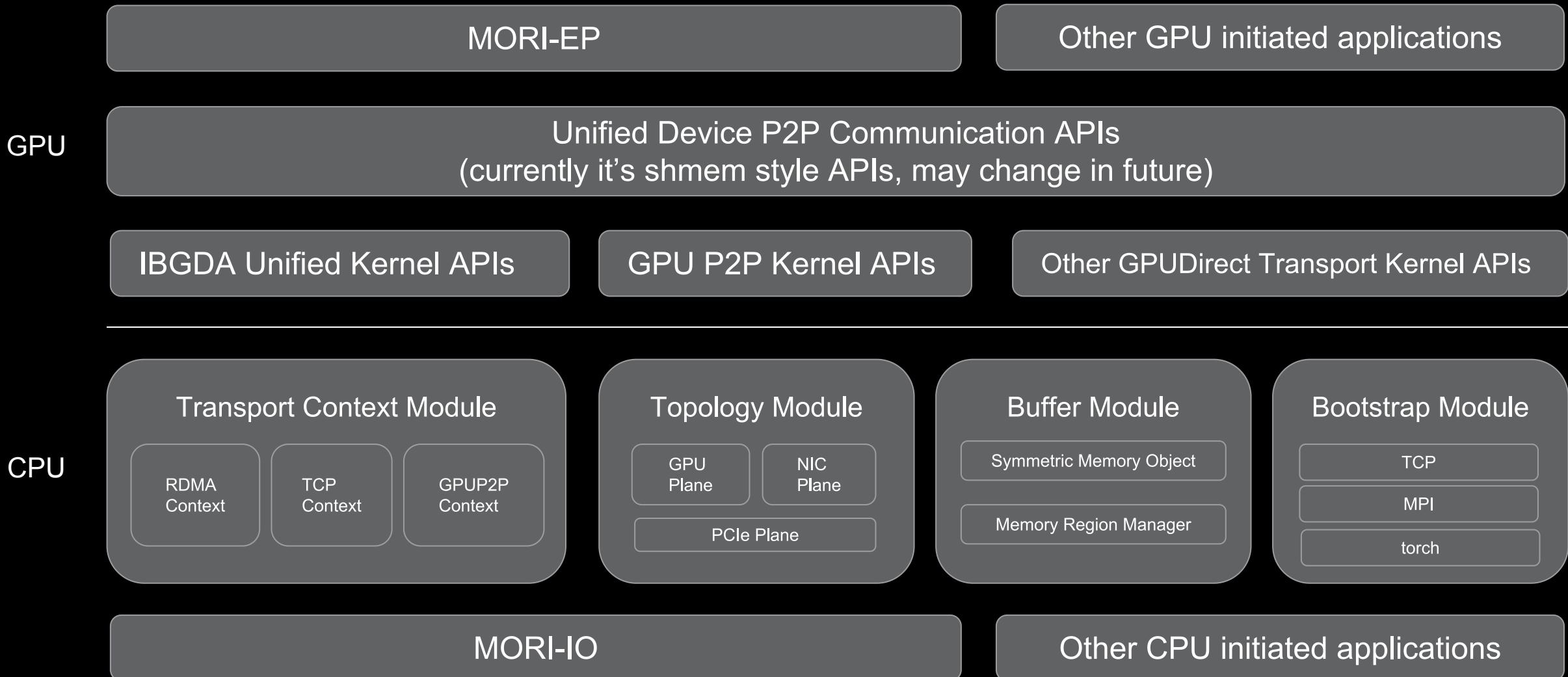
AMD 大规模专家并行在ROCM实践2：自研开源通信库MORI覆盖 ROCM 机内/跨机场景

- 多个开源通信库，有效支持PD分离等分布式推理方案

AMD开源通信库	代码链接
RCCL	https://github.com/NVIDIA/nccl
rocShmem	https://github.com/ROCm/rocshmem
MORI	
AMD DeepEP	https://github.com/ROCm/DeepEP

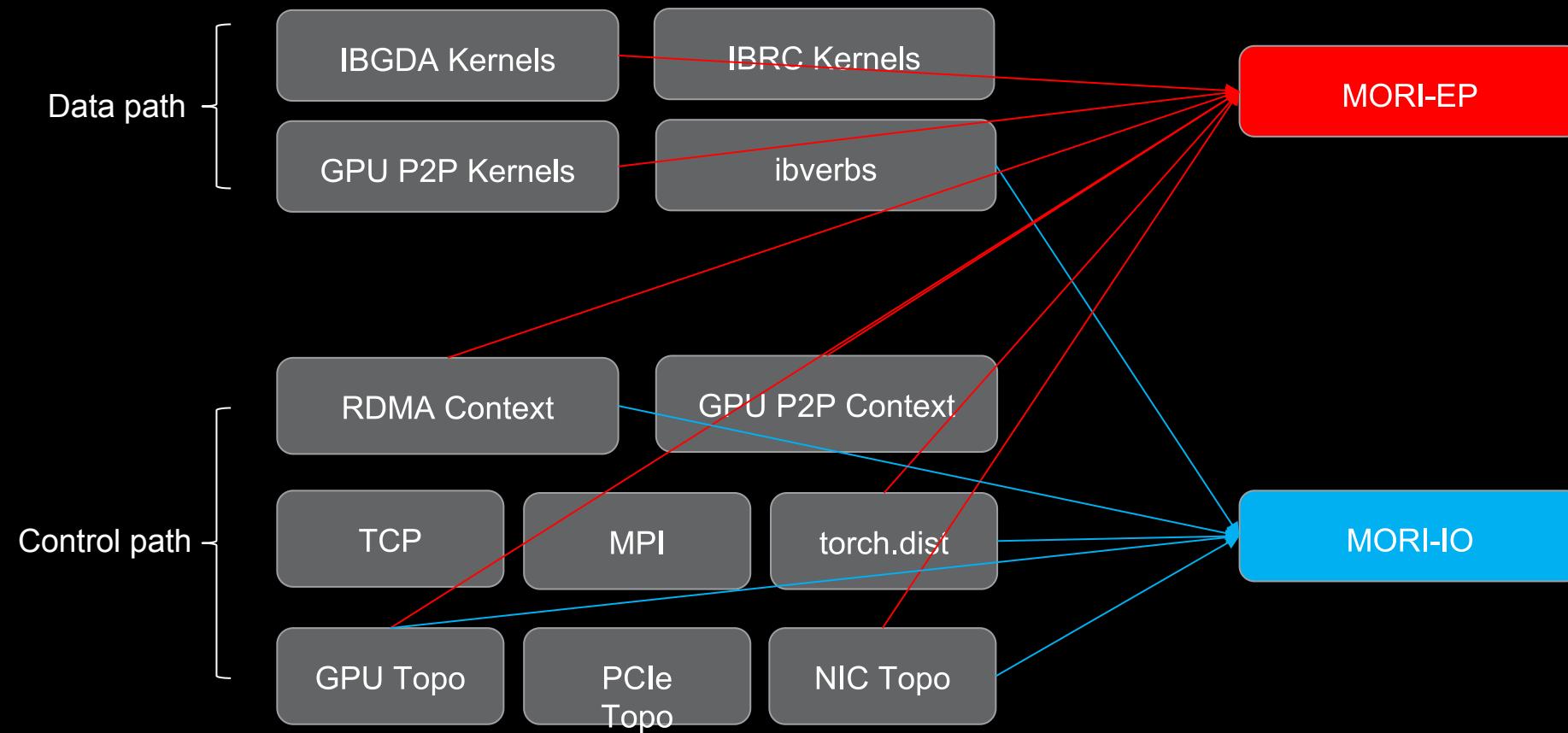


AMD 大规模专家并行在ROCM实践2：自研开源通信库MORI ROCM 覆盖机内/跨机场景



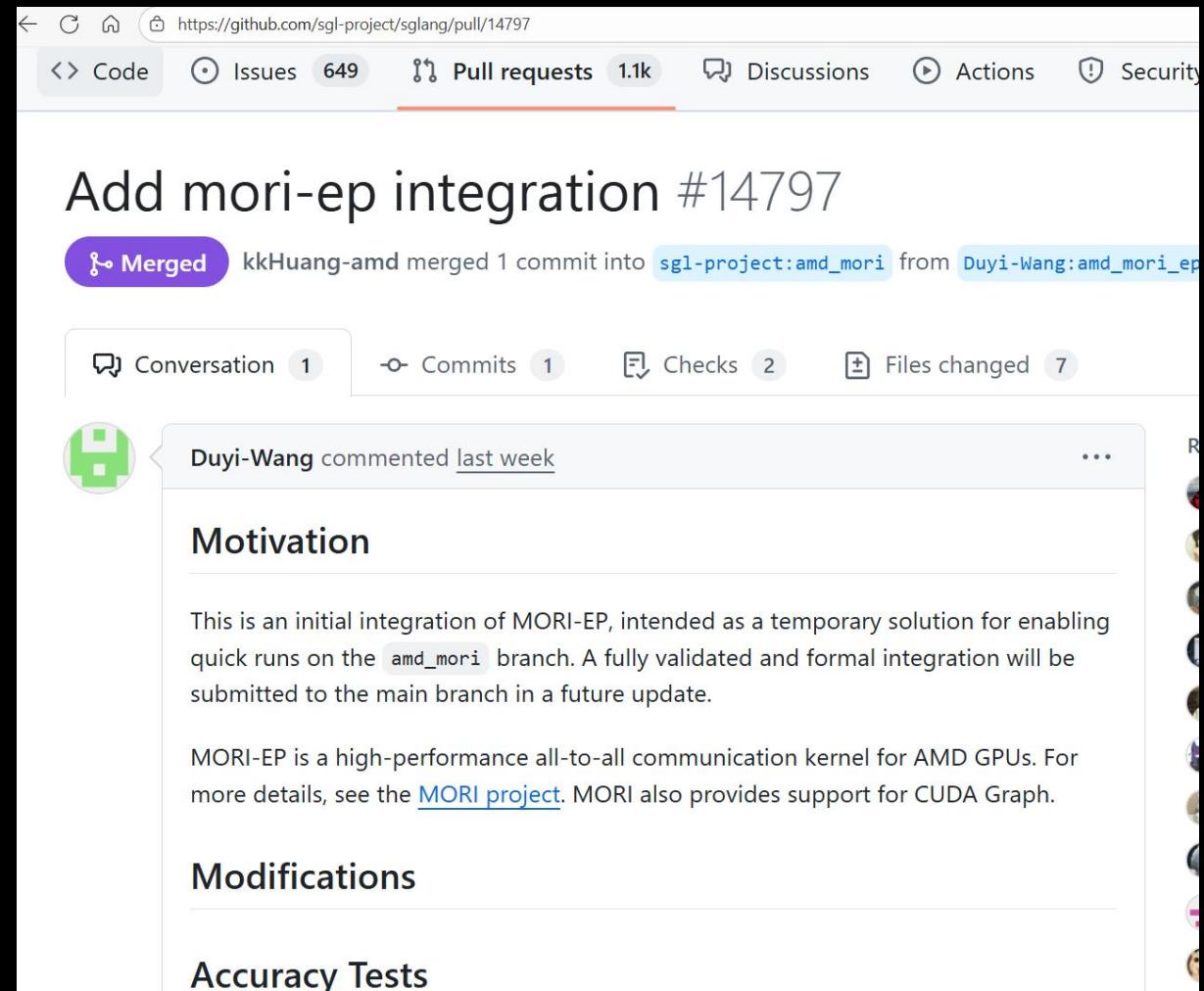
AMD 大规模专家并行在ROCM实践2：自研开源通信库MORI

ROCM 覆盖机内/跨机场景



AMD 大规模专家并行在ROCM实践2：自研开源通信库MORI ROCM 覆盖机内/跨机场景

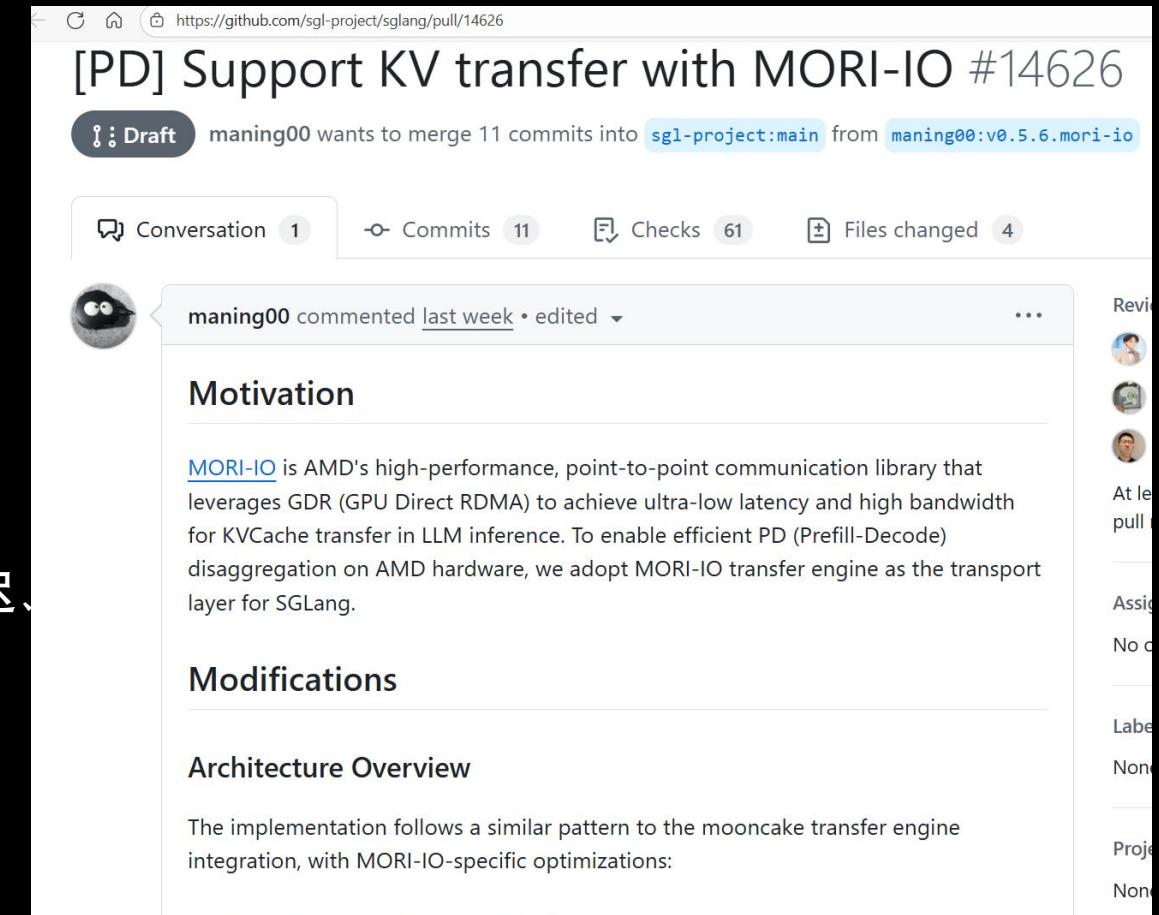
- 开源贡献1：集成MORI-EP
 - PR 链接: <https://github.com/sjl-project/sgl-lang/pull/14797>
- 实现细节：
 - 遵循DeepEP的集成方式
 - 实现MoRI-EP token dispatcher
 - 新增a2a_backend.is_mori()



AMD 大规模专家并行在ROCM实践2：自研开源通信库MORI ROCM 覆盖机内/跨机场景

- 开源贡献2：集成MORI-IO
 - PR 链接: <https://github.com/sjl-project/sglang/pull/14626>

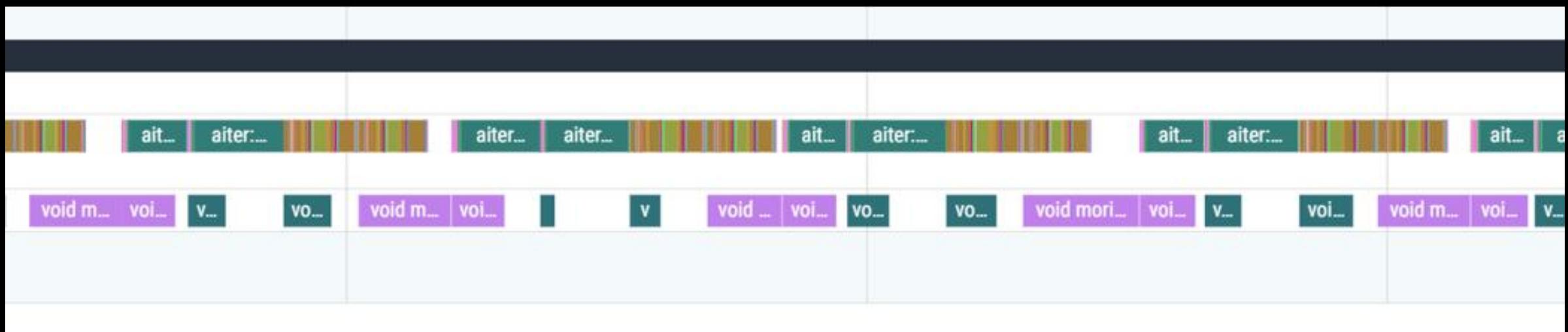
- MORI-IO 优势
 - MORI-IO 是 AMD 面向 GPU 场景打造的 点对点高性能通信库
 - 基于 GPU Direct RDMA (GDR) , 实现超低延迟、超高带宽的数据传输
 - 非常适合 KV Cache 这种 高频、大容量、对延迟极其敏感 的数据交换场景

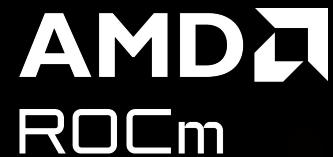


AMD 大规模专家并行在ROCM实践3：计算通信交叠

ROCM

- TBO实现
- 多流效果展示
- 更多的profiling分析正在进行





更多开发者资源

AMD AI Academy

Build real-world AI skills with free, hands-on learning paths from general AI to Enterprise AI



AMD ROCm™ AI Developer Hub

Access resources for accelerating AI on AMD GPUs



AMD Developer Cloud

Start your AI journey with complimentary credits!



AMD Developer Discord

Join spaces to discuss, share feedback, and receive tips and tricks from developers alike.

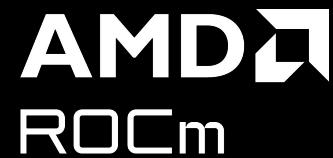


AI@AMD X
Follow us on X



AMD Developer
Follow us on LinkedIn





国内开发者资源

微信公众号



AMD B站



AMD 知乎



AMDG

Unified FP8

超越混合精度的稳定高效强化学习方案

Moving Beyond Mixed Precision for Stable and Accelerated
MoE RL

InfiXAI



蚂蚁集团 AQ



SGLang RL



NVIDIA



我们实现了在 RL 中完全使用 FP8 进行采样 (rollout) 和训练 (training)。实验表明，MoE 模型规模越大，BF16 训练 FP8 采样的训推差异越明显。训推过程统一使用 FP8 有效消除了量化误差导致的不一致性，提升了 RL 训练的速度和稳定性。

在此基础上，我们进一步向大家分享兼顾稳定性与性能的新进展——在 RL 中实现全流程的 FP8 训练与采样。
[Qwen3-4B](#) 与 [Qwen3-30B-A3B](#) 模型的 FP8 RL 训练已在 `slime` 中全面支持，开箱即用。

本次工作由 InfiXAI 团队、蚂蚁集团 AQ 团队、SGLang RL 团队、miles 团队及 slime 团队联合完成。特别感谢 DataCrunch 为本工作提供的算力赞助，以及 NVIDIA 在 Transformer Engine (TE) 方面给予的技术支持。

大模型训练的显存与算力瓶颈

显存墙限制规模扩展

当前主流 BF16 训练方案占用显存巨大，直接制约了 MoE 等超大模型的规模扩展能力。

FP8 的理论突破

相比 BF16，FP8 格式可将显存占用减少 50%，并提供 2 倍的理论算力峰值。

硬件就绪

H100 GPU 的 FP8 Tensor Core 提供高达 1979 TFLOPS 算力，是 BF16 单元的两倍。

50%

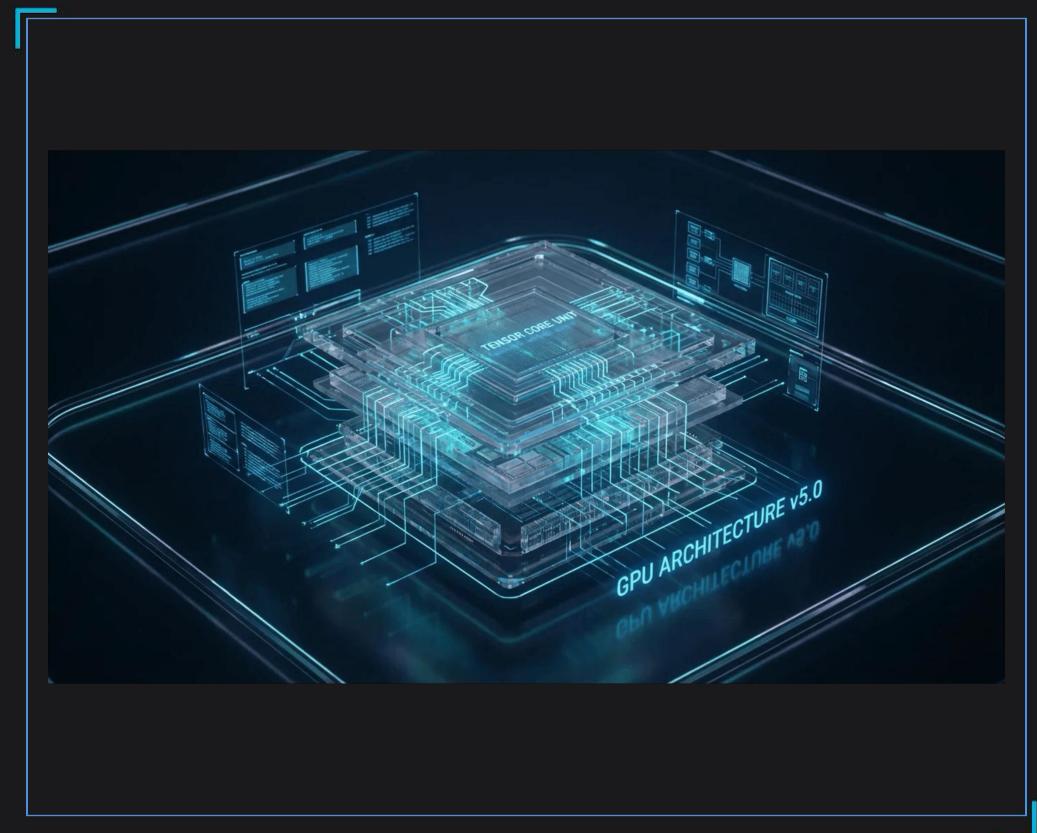
显存节省

2x

算力提升

1979

TFLOPS (H100)



FP8 浮点格式与硬件支持

E4M3

4位指数 + 3位尾数

特点：动态范围较小，但精度相对较高。适用于前向传播。

E5M2

5位指数 + 2位尾数

特点：动态范围更大，精度较低。适用于梯度计算。

	E4M3	E5M2
Exponent bias	7	15
Infinities	N/A	S 11111 00 ₂
NaN	S 1111 111 ₂	S 11111 {01, 10, 11} ₂
Zeros	S 0000 000 ₂	S 00000 00 ₂
Max normal	S 1111 110 ₂ = ± 2 ⁸ × 1.75 = ± 448	S 11110 11 ₂ = ± 2 ¹⁵ × 1.75 = ± 57,344
Min normal	S 0001 000 ₂ = ± 2 ⁻⁶	S 00001 00 ₂ = ± 2 ⁻¹⁴
Max subnorm	S 0000 111 ₂ = ± 2 ⁻⁶ × 0.875	S 00000 11 ₂ = ± 2 ⁻¹⁴ × 0.75
Min subnorm	S 0000 001 ₂ = ± 2 ⁻⁹	S 00000 01 ₂ = ± 2 ⁻¹⁶

Arch	FP64	F16	INT8	INT4	FP8	MXFP
Volta	✗	✓ FP16	✗	✗	✗	✗
Turing	✗	✓ FP16	✓	✓	✗	✗
Ampere	✓	✓ FP16/BF16	✓	✓	✗	✗
Hopper	✓	✓ FP16/BF16	✓	✗	✓ (累加精度只支持FP22)	✗
Blackwell	✓	✓ FP16/BF16	✓	✗	✓	✓ MXFP(8/6/4) NVFP4
Blackwell Ultra	✓ (减少算力)	✓ FP16/BF16	✓ (减少算力)	✗	✓	✓ MXFP(8/6/4) NVFP4

分层量化策略实现精度与效率的平衡

SCALING FACTOR $S = \max|X| / V_{\max}$ QUANTIZATION $Q(x) = \text{round}(x / S)$

激活值

1x128

采用逐块(1x128)量化。激活值中常出现显著的离群值(Outliers)，较细的量化粒度能将影响限制在局部，保留整体精度。

权重

128x128

采用逐块(128x128)量化。权重分布通常平稳但对误差敏感。Block-wise量化在保证精度的同时能更好地配合硬件优化。

梯度

1x128

采用E4M3格式的逐块(1x128)量化。虽然梯度动态范围大，但**DeepSeek-V3**等研究证明细粒度E4M3量化能兼顾精度与动态范围。

Scale 精度选择

FP32 vs E8M0

我们最终采用**FP32 Scale**而非E8M0。

1. 数值稳定性: FP32能精细捕捉动态范围，确保Loss曲线贴近BF16。
2. 硬件现状: Hopper架构无针对E8M0 Scale的加速单元，强行使用无收益。

理论优势与实际应用的鸿沟

01. 显存与计算效率未达预期

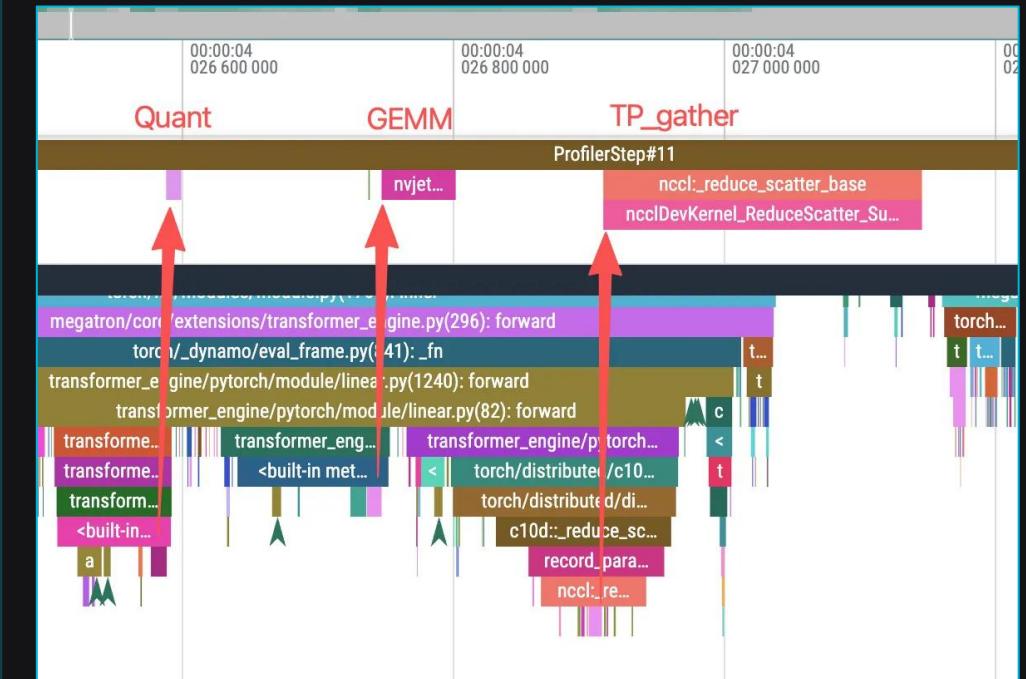
冗余权重副本: TE为加速反向传播, 额外保存转置后的量化权重。

高精度激活值: Attention层仍保留FP32激活值用于梯度计算。

小批量瓶颈: Batch size较小时, 量化开销导致CPU Bound, 性能甚至劣于BF16。

```
Code Blame 579 lines (501 loc) · 24.3 KB
480
481 ... void quantize_transpose_square_blockwise(const SimpleTensor& input, SimpleTensor& scale_inv,
482     SimpleTensor& scale_inv_t, SimpleTensor& output,
483     SimpleTensor& output_t, const float epsilon,
484     const bool return_transpose, const bool pow_2_scale,
485     const SimpleTensor& noop_tensor, cudaStream_t stream) {
```

TransformerEngine 在Quant的时候会保存一份转置的weight, 导致了显存的节省小于预期



CPU Bound for FP8 Training: GPU kernel并不密集, 很多时候 GPU 已经完成了上一个工作, 但由于 CPU bound 导致下一个 kernel launch 还没发出来。

理论优势与实际应用的鸿沟

01. 显存与计算效率未达预期

冗余权重副本: TE为加速反向传播, 额外保存转置后的量化权重。

高精度激活值: Attention层仍保留FP32激活值用于梯度计算。

小批量瓶颈: Batch size较小时, 量化开销导致CPU Bound, 性能甚至劣于BF16。

02. 精度对齐困难

固有误差: 单次FP8 GEMM引入约0.0007的误差。

逐层累积: 深层Transformer中误差放大。

RL敏感性: 梯度由对数概率差决定, 微小误差可能导致策略更新方向偏差。

Kernel (M, K, N)	cuBLAS(TE)	DeepGEMM
128,128,128	0.00068	0.00036
256,128,256	0.00068	0.00037
320,128,336	0.000684	0.00037
320,64,336	0.00067	0.00024
320,256,336	0.00068	0.00048
1024,4096,1024	0.000681	0.00065
2048,2048,512	0.00068	0.00063
1024,1024,1024	0.000683	0.0006

最初我们猜测 TransformerEngine 里闭源的 cuBLAS GEMM 实现不如开源广泛使用的 DeepGEMM 准确, 因此我们设计实验比较了两种 FP8 GEMM 实现 (cuBLAS 和 DeepGEMM) 与 BF16 的精度差异。从结果来看, cuBLAS和DeepGEMM同样实现了split accumulator, 精度均能够达到0.0007的「特征值」。

理论优势与实际应用的鸿沟

01. 显存与计算效率未达预期

冗余权重副本: TE为加速反向传播, 额外保存转置后的量化权重。

高精度激活值: Attention层仍保留FP32激活值用于梯度计算。

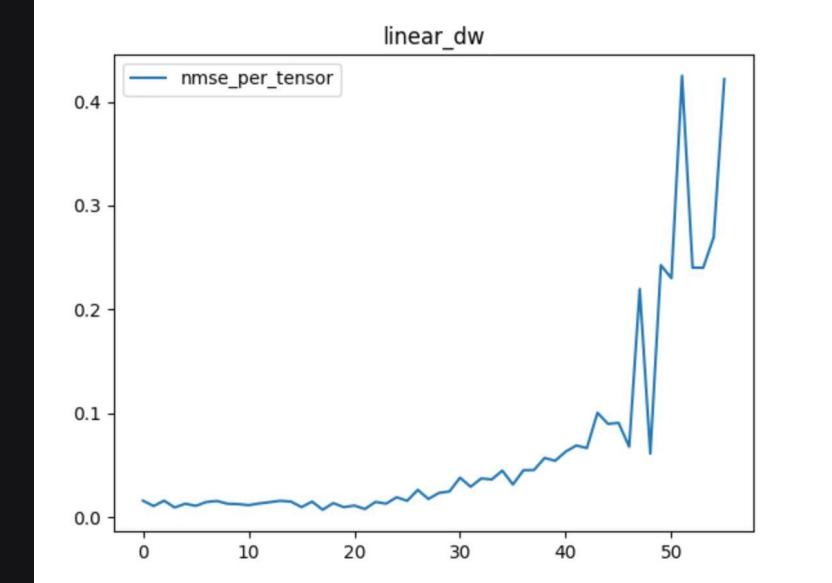
小批量量瓶颈: Batch size较小时, 量化开销导致CPU Bound, 性能甚至劣于BF16。

02. 精度对齐困难

固有误差: 单次FP8 GEMM引入约0.0007的误差。

逐层累积: 深层Transformer中误差放大。

RL敏感性: 梯度由对数概率差决定, 微小误差可能导致策略更新方向偏差。



然而, 对于一个28层模型, 假设我们选取其中一个Linear层, 对比BF16与FP8的diff, 可以发现从forward到backward中activation的差异会越来越大

理论优势与实际应用的鸿沟

01. 显存与计算效率未达预期

冗余权重副本: TE为加速反向传播, 额外保存转置后的量化权重。

高精度激活值: Attention层仍保留FP32激活值用于梯度计算。

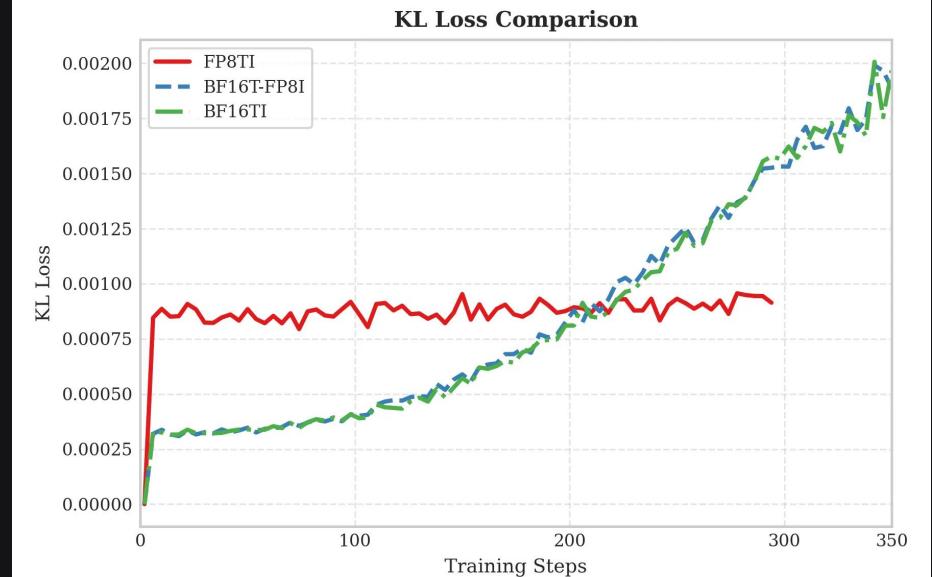
小批量瓶颈: Batch size较小时, 量化开销导致CPU Bound, 性能甚至劣于BF16。

02. 精度对齐困难

固有误差: 单次FP8 GEMM引入约0.0007的误差。

逐层累积: 深层Transformer中误差放大。

RL敏感性: 梯度由对数概率差决定, 微小误差可能导致策略更新方向偏差。



在W8A8下, 对于Qwen3-4B 模型, 我们发现在第一个step上就会有比较大的KL Diff
(有意思的是, 当我们拿DeepSeek Distill系列模型来做startpoint时并没有这个问题)

理论优势与实际应用的鸿沟

01. 显存与计算效率未达预期

冗余权重副本: TE为加速反向传播, 额外保存转置后的量化权重。

高精度激活值: Attention层仍保留FP32激活值用于梯度计算。

小批量量瓶颈: Batch size较小时, 量化开销导致CPU Bound, 性能甚至劣于BF16。

02. 精度对齐困难

固有误差: 单次FP8 GEMM引入约0.0007的误差。

逐层累积: 深层Transformer中误差放大。

RL敏感性: 梯度由对数概率差决定, 微小误差可能导致策略更新方向偏差。

03. 框架兼容性问题

版本依赖: TransformerEngine迭代快, 版本间数值结果可能不一致。

结构支持: 对MLA等非标准组件的支持尚不成熟。

功能冲突: Optimizer CPU offload与FP8 param gather暂不兼容。

前置工作：先用slime来做FP8的SFT吧

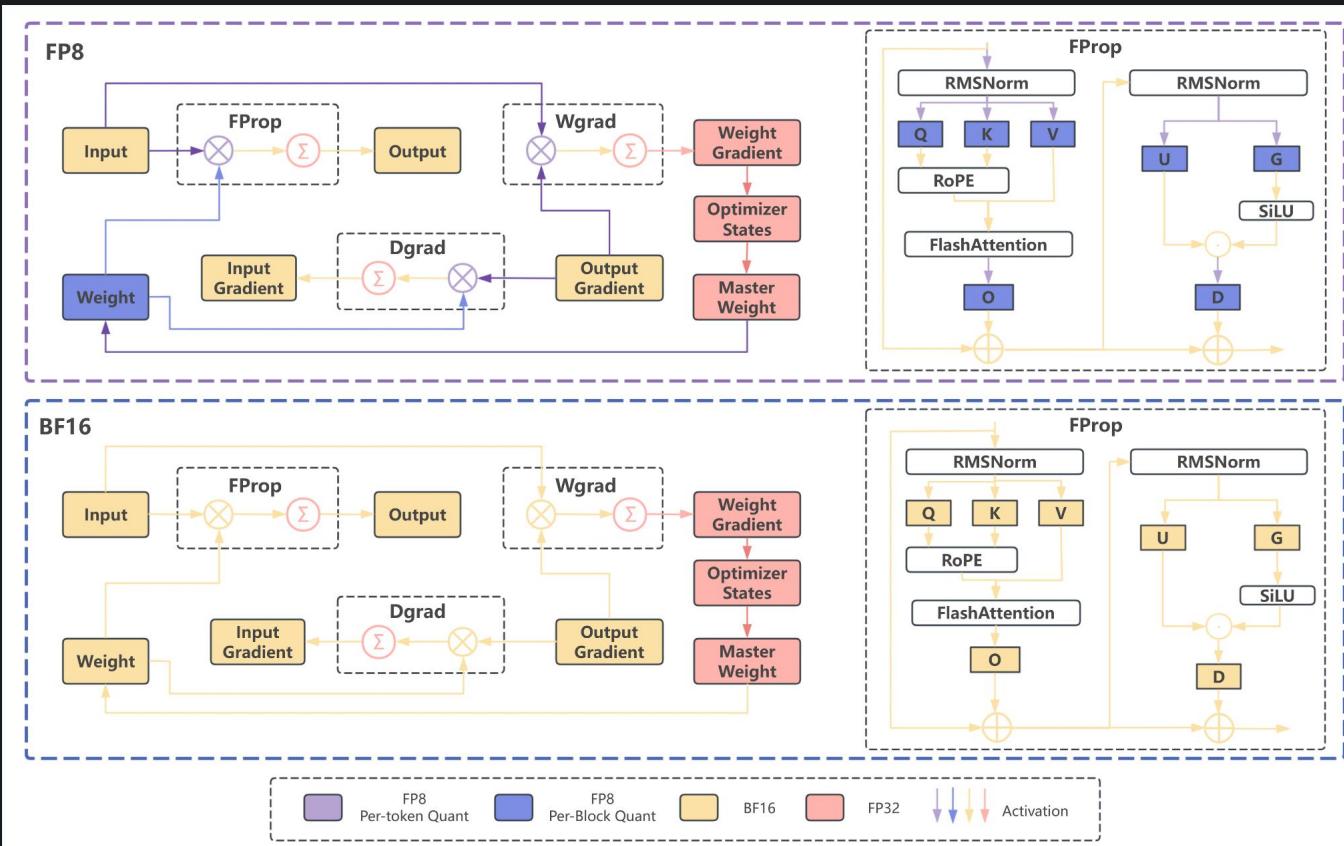


Table 4: A detailed comparison of training quantization methods (BF16, FP8 w. FP32 scale, FP8) on the performance of Qwen2.5-Math-7B and Qwen2.5-Math-1.5B. The models were evaluated on AIME 25, AIME24, GPQA, and LiveCodeBench v5 using checkpoints from Stage 1 and Stage 2 of training.

Base Model	Quantization Method	AIME 25	AIME24	GPQA	LiveCodeBench v5
Qwen2.5-Math-7B	BF16	44.16	56.87	45.14	32.22
	FP8 w. FP32 scale	44.06	56.67	47.98	32.18
	FP8	44.89	57.81	47.10	31.34
Qwen2.5-Math-1.5B	BF16	50.00	59.48	48.36	35.22
	FP8 w. FP32 scale	46.46	57.92	45.39	35.87
	FP8	49.79	59.69	46.78	36.21
Qwen2.5-Math-1.5B	BF16	15.41	18.33	24.68	10.71
	FP8 w. FP32 scale	15.73	18.65	25.38	10.14
	FP8	17.50	16.88	23.17	9.84
Qwen2.5-Math-1.5B	BF16	17.92	21.35	24.48	12.16
	FP8 w. FP32 scale	20.62	22.81	27.78	12.69
	FP8	20.73	21.77	25.13	12.96

Table 2: Performance comparison on reasoning benchmarks. All scores are percentages (%).

Model	AIME 25	AIME 24	GPQA	LiveCodeBench v5
Deepseek-Distill-Qwen-1.5B	21.35	26.87	32.26	18.50
Qwen2.5-1.5B-base (w. InfiAlign)	14.58	10.52	28.98	12.99
InfiR2-1.5B-FP8	18.45	17.39	29.48	17.10
Deepseek-Distill-Qwen-7B	43.00	49.00	48.20	37.60
Qwen2.5-7B-base (w. InfiAlign)	33.75	43.02	48.11	39.48
InfiR2-7B-FP8	40.62	55.73	45.33	40.31

关键发现：误差源于量化自身

实验设计 (Experimental Setup)

MODE 1: Baseline

BF16 权重 + BF16 输入

使用标准 BF16 GEMM 算子

MODE 2: FP8 Real Quant

FP8 权重 + FP8 输入

使用真实 FP8 GEMM 算子 (cuBLAS)

MODE 3: FP8 Fake Quant

BF16 权重 + BF16 输入 (模拟量化)

使用 BF16 GEMM 算子

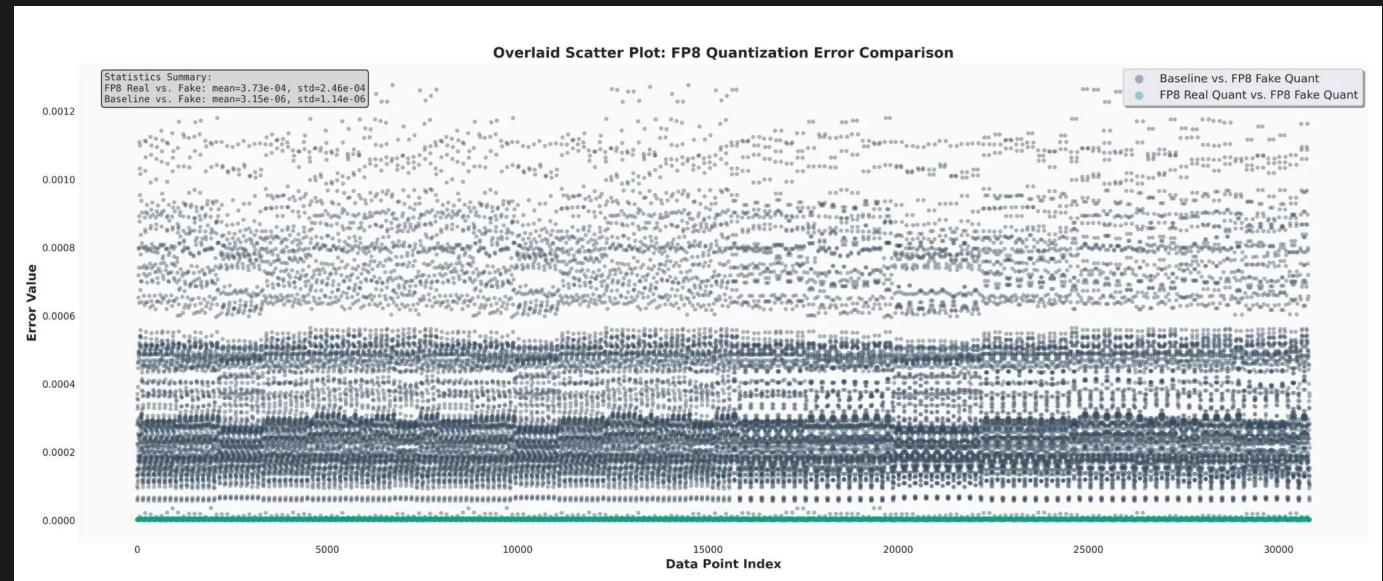
误差分析 (Error Analysis)

Fake Quant vs. Baseline

ERROR: HIGH

Real Quant vs. Fake Quant

ERROR: LOW



真实计算与模拟量化几乎一致。证明 cuBLAS FP8 GEMM 算子精度极高，可信賴。

CONCLUSION

误差源于 **量化 (Quantization)** 本身，而非 **算子实现 (Operator)**。

TransformerEngine 的 FP8 GEMM 算子精度可靠，可直接投入生产环境。

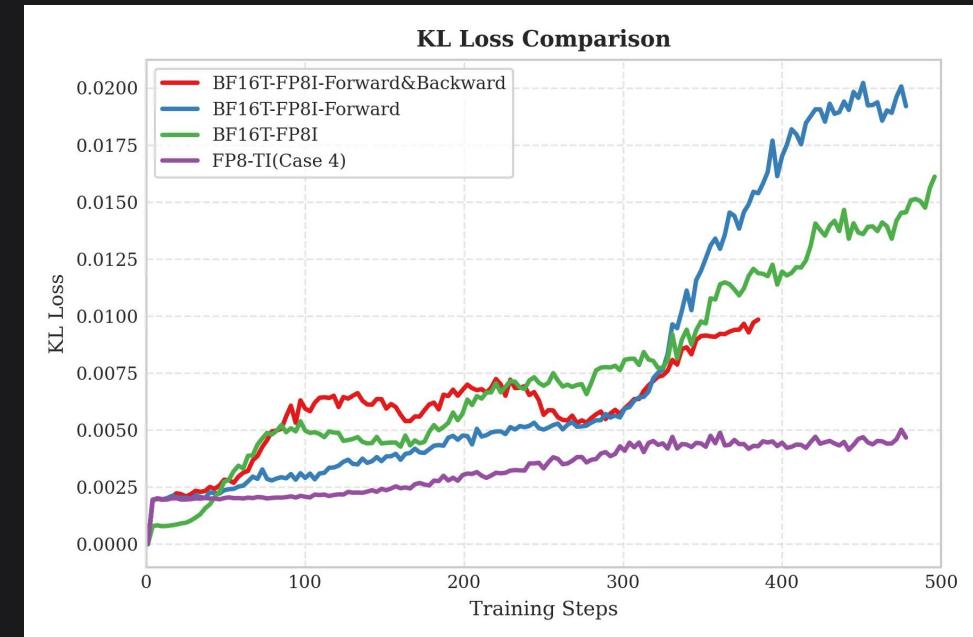
RL 训练中的训推不一致性问题

实验方案设置

Case 1 (Baseline)
BF16 训练 / FP8 推理
混合精度模式
Case 2
BF16 训练 (Forward Quant) / FP8 推理
Case 3
BF16 训练 (Fwd+Bwd Quant) / FP8 推理
Case 4 (Unified)
FP8 训练 / FP8 推理
统一精度模式

关键指标分析: KL Loss

KL Loss 对比



上图展示了四种情况下 KL loss 的变化。

可以看到, Case 2、Case 3 和 Case 4 (FP8-TI) 在 step 1 的 KL loss 基本一致, 并且都显著高于 Case 1.

Forward 过程的量化误差是主要原因。

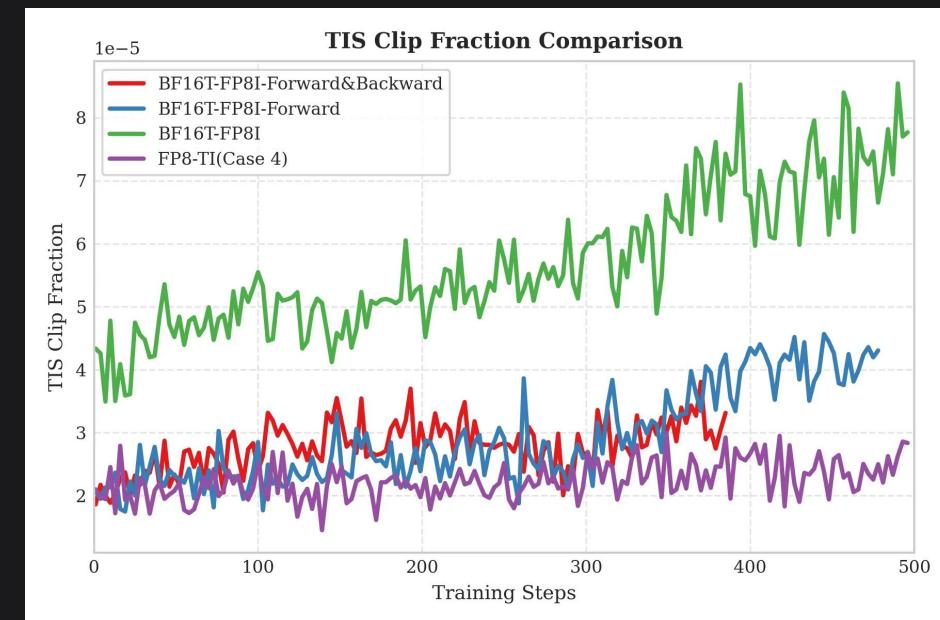
RL 训练中的训推不一致性问题

实验方案设置



关键指标分析: TIS-clipfrac

TIS-clipfrac 对比 (越低越稳定)

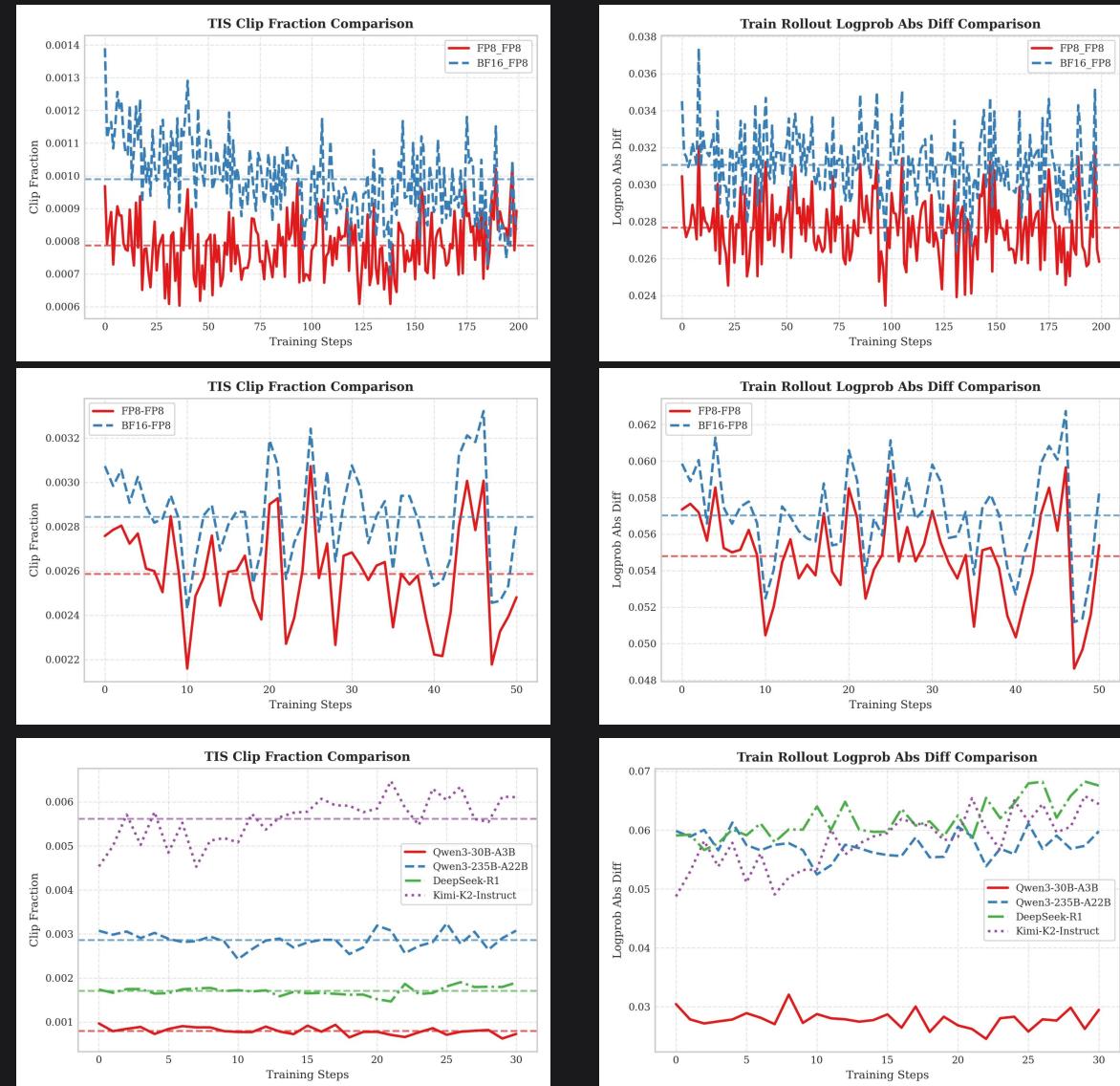


结论:

混合精度模式 (Case 1) 导致高 TIS-clipfrac, 表明严重的训推不一致。

统一精度方案 (Case 4) 与模拟量化方案表现一致, 显著降低了不一致性。

MoE 模型验证：30B 与 235B



Qwen3-30B-A3B

2x H20 GPU

TIS-clipfrac 显著降低

FP8 训推方案的裁剪比例远低于混合精度方案，训练更稳定。

训推概率差异收窄

Train_rollout_logprob_abs_diff 波动范围明显减小。

Qwen3-235B-

16x H20 GPU

A22B 优势持续扩展

在更大规模模型上，FP8 训推方案的优越性依然显著。

可扩展性验证

方案在不同规模 MoE 模型上表现一致，证明了其普适性。

其他 MoE

越大的MoE，越容易出现Diff，因此越大的模型越需要关注训推一致性

总结与展望：稳定性与效率的双赢

主要贡献

ROOT CAUSE ANALYSIS

系统分析了 FP8 训练挑战，定位量化误差为训推不一致的根本原因。

UNIFIED FP8 SCHEME

验证 FP8 统一训推方案，有效消除了混合精度导致的一致性。

SCALABILITY

在 30B 和 235B MoE 模型上验证了方案的可扩展性。

已支持模型

QWEN3 SERIES

- Qwen3-4B
- Qwen3-30B-A3B
- Qwen3-235B-A22B
- and other Megatron-Core models

FRAMEWORK SUPPORT

已在 `slime` 框架中全面支持，开箱即用。

未来展望

INCONSISTENCY RESEARCH

深入研究训推不一致性的根本机理，探索更优解决方案。

QUANTIZATION STRATEGY

深化量化策略研究，探索误差更小的量化方法。

CO-DESIGN ACCELERATION

通过硬件-框架-算法协同设计，隐藏量化耗时，实现真正加速。

ACKNOWLEDGMENTS

InfiXAI Team

| 蚂蚁集团 AQ Team

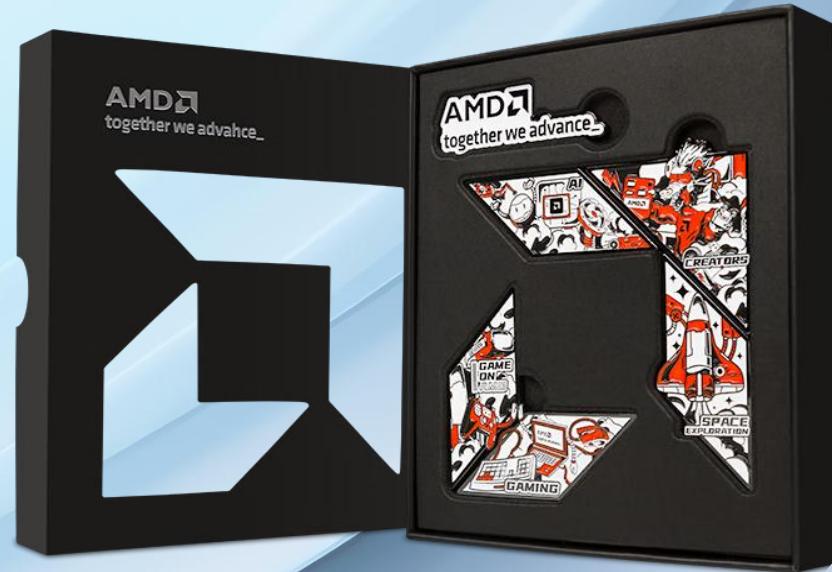
| SGLang RL Team

| slime Team

| NVIDIA



腾讯长鹅毛绒抱枕*5



AMD 3D立体磁贴*3



AMD Radeon RX 9070 XT 16G 显卡





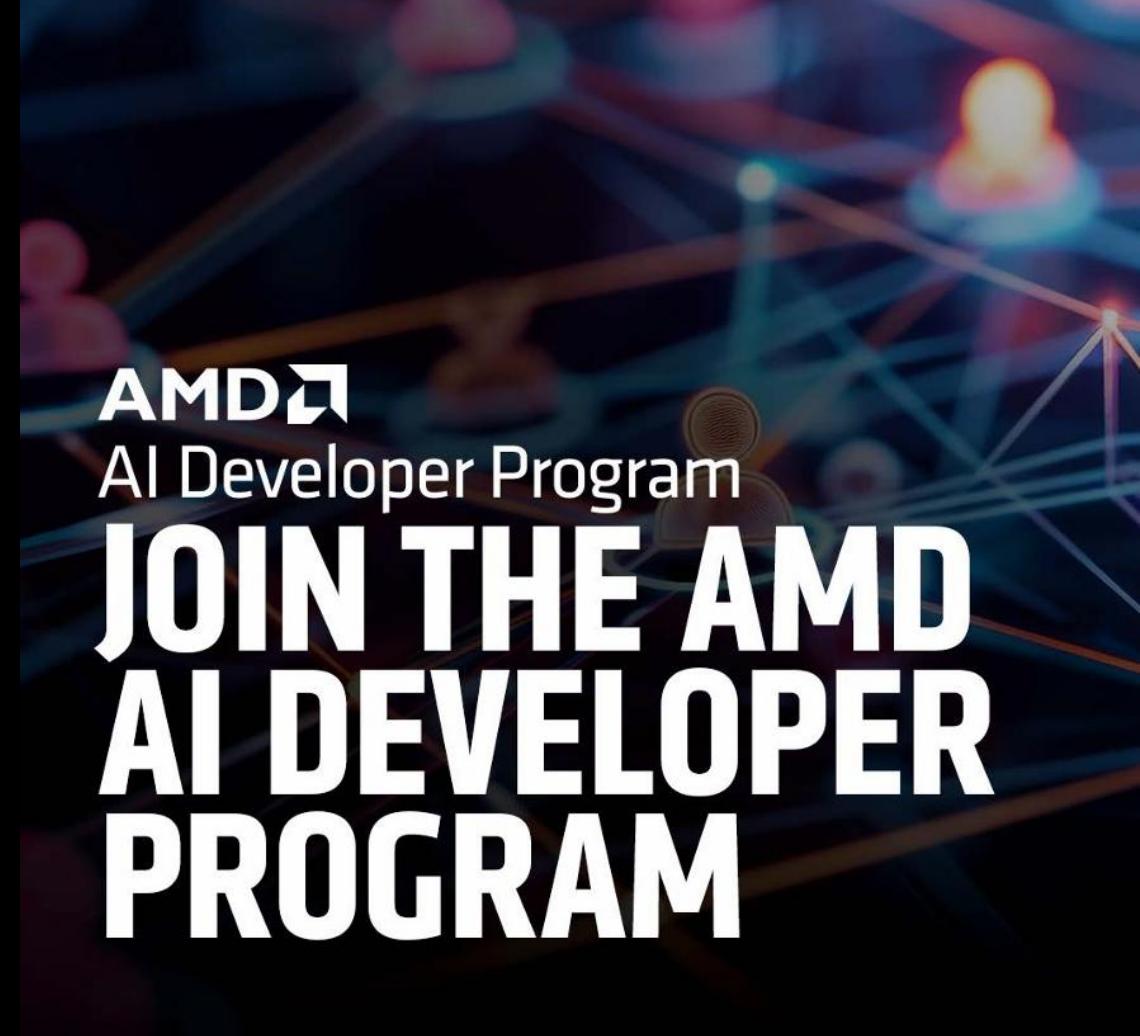
Developer Program Share

AMD
together we advance_

AMD AI开发者项目



<https://account.amd.com/en/forms/registration/ai-dev-program.html>



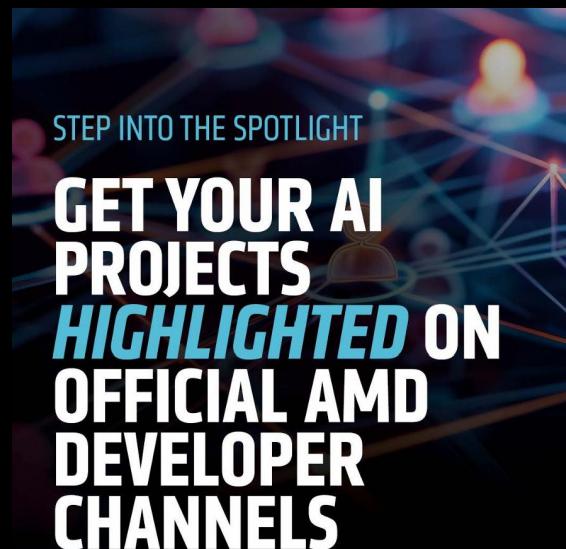
四大亮点

\$100 Dev Cloud算力券

月度抽奖赢不停

个人项目官方高亮

AI开发者交流群



AMD
AI Developer Program

AMD
AI Developer Program

AMD
AI Developer Program

AMD
AI Developer Program

AI开发者资源包

Resources	Non-Member Access	AMD AI Developer Program Access
Technical Materials		
Intro AMD Developer Cloud Credits		✓
Technical Blogs	Read Only	Can Author/Contribute
Documentation	✓	✓
Training		
AI Academy	Limited Access	Exclusive Courses
DeepLearning.AI Pro membership		1 Month
Developer Videos and Webinars	✓	✓
Ask the Experts Office Hours		✓
Community		
Developer Newsletter	Can opt in	Personalized Content
Developer Forum	General Developer Forum Access	Member-Only Forum
Developer Events	✓	Early Invites
Recognition		
Recognized at AMD Events		✓
Showcased on AMD social channels		Limited

开发者云算力券
联名AI blog
获取AI文档

AI Academy专属课程
Deeplearning.AI限免
开发者视频及线上资源
AI专家 office hour

AI开发者专享简讯
AI开发者交流群
AI开发者活动早邀

AI开发者活动认证
AI开发者社媒互动

Join AMD AI Developer Program

AMD AI Developer Program Sign Up Form

Thank you for your interest in the AMD AI Developer Program. Complete the form to join and start accessing your member benefits.

By joining this program, you will receive periodic program emails notifying you about member benefits, upcoming events, new technical resources, and sweepstakes announcements.

First Name *

Bingqing

Last Name *

Guo

E-mail *

bingqing.guo@amd.com

Company Name

AMD

AMD AI开发者项目



<https://account.amd.com/en/forms/registration/ai-dev-program.html>

AMD
AI Developer Program

JOIN
TODAY



更多AI开发者资源

开发者中心



AMD BiliBili 开发者专区



AMD 知乎开发者专栏



AMD



How to Develop and Optimize Triton Kernels

Agenda

- Why Triton
- Overview of GPU fundamentals
- Introduce the features of Triton
- Go through the Triton sample Code, vector-add kernel, Softmax kernel

A Tale of Two Racers

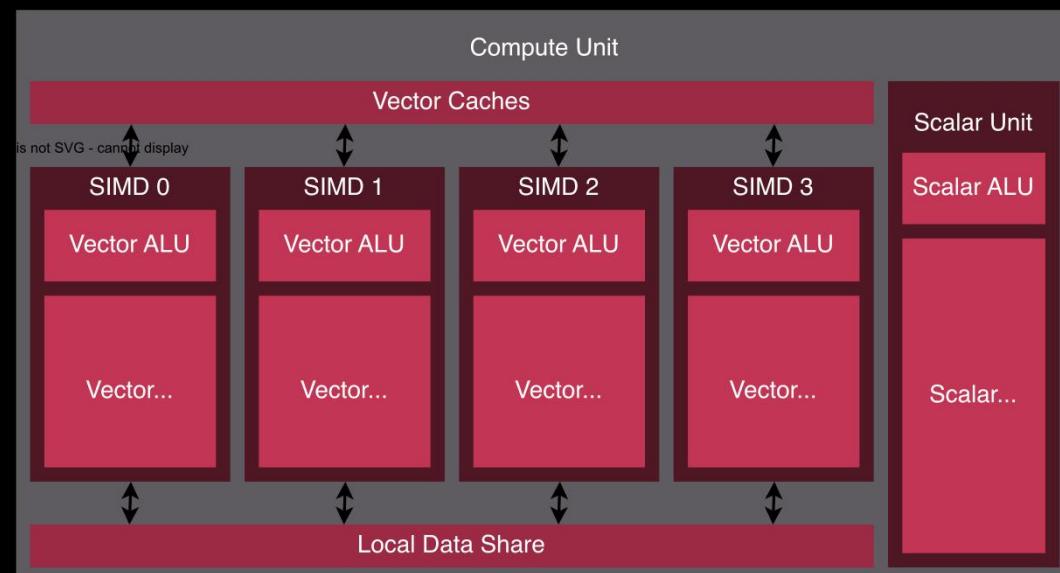
1000 horsepower !



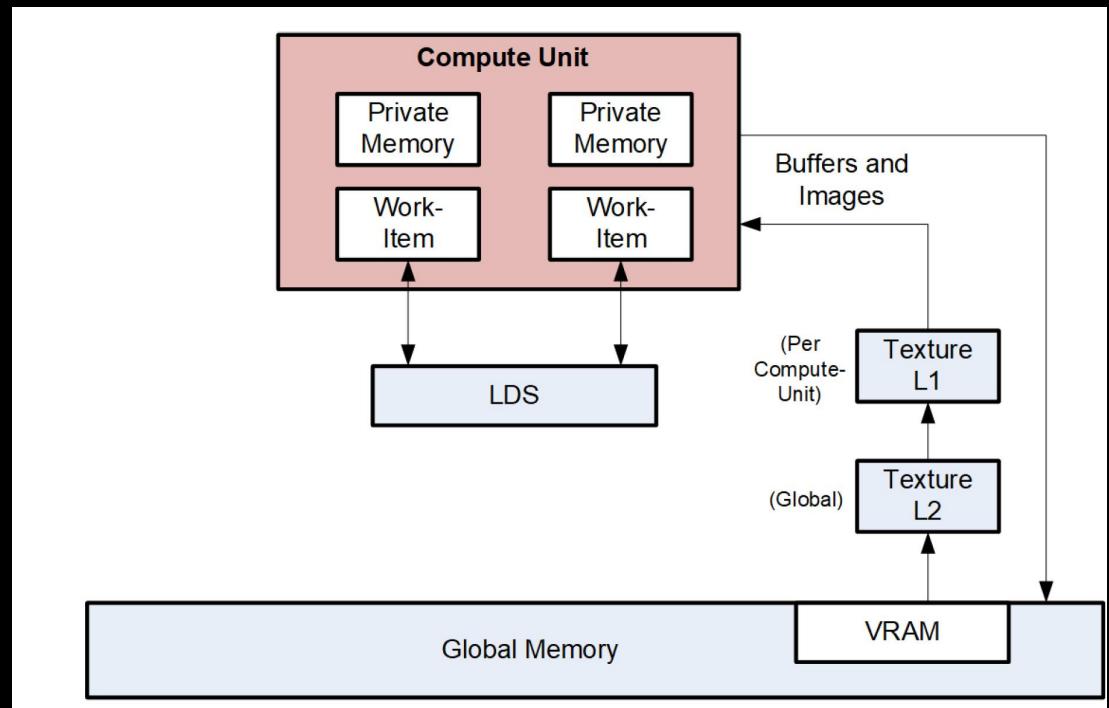
easy to drive, streamlined in design, still powerful

AMD GPU Architecture

- AMD GPU Architecture
 - 8 accelerated compute dies (XCDs) with totally 304 compute units (CUs)
 - 4 SIMDs per CU, 16 lanes per SIMD, 64 threads per wavefront
 - Memory Hierarchy: VGPRs/SGPRs, LDS and L1 Cache, L2 Cache, Global Memory

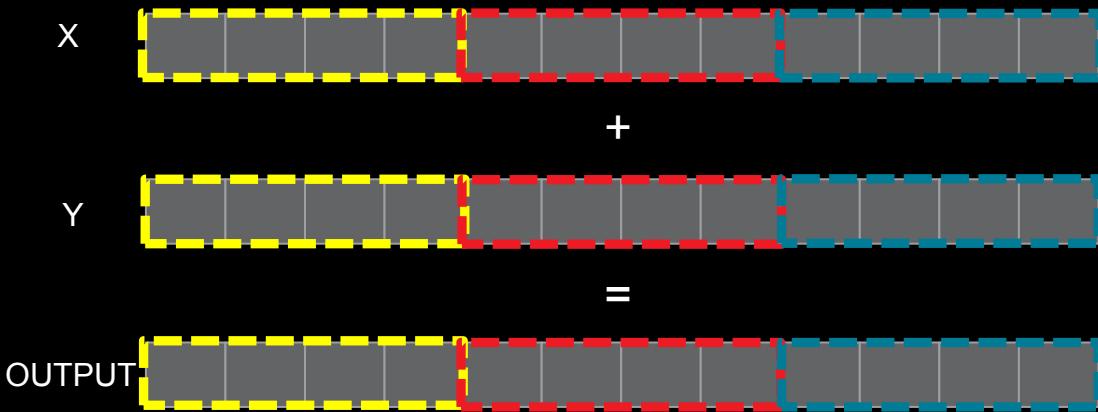


Memory Hierarchy



Triton Programming Model

- **Block** in Triton
 - Organizes the computation on data blocks
 - One program per block
 - Block size – powers of two
 - Each program instance acts on $[pid * \text{BLOCK_SIZE} : (pid + 1) * \text{BLOCK_SIZE}]$ in below example

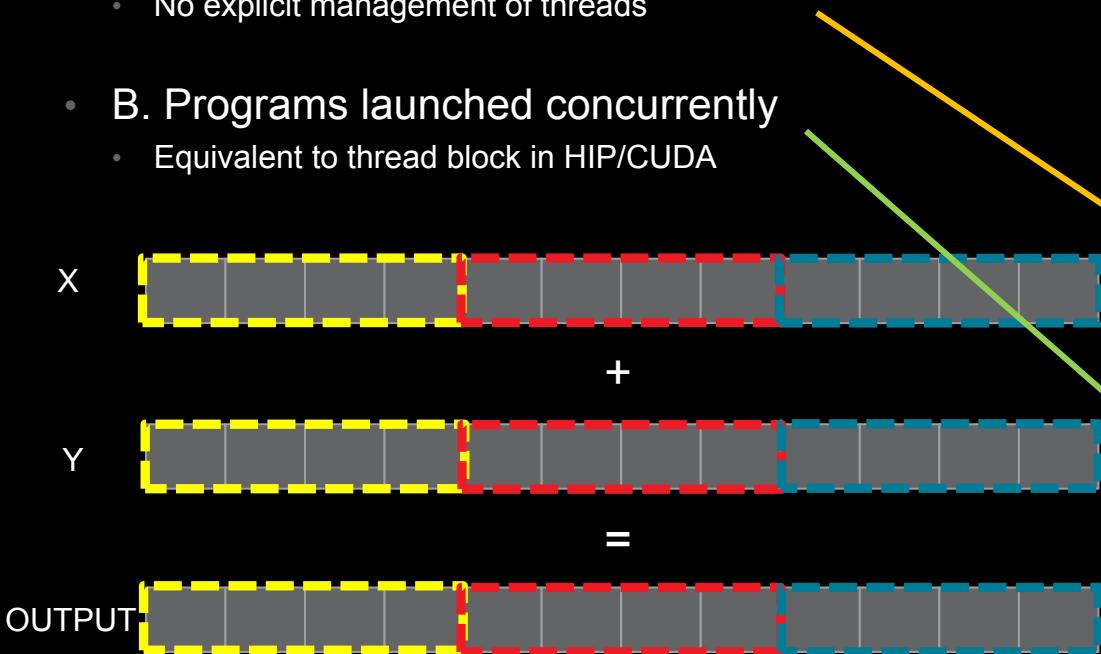


Split X, Y, OUTPUT into 3 blocks for 3 programs

```
70 @triton.jit
71 def add_kernel(x_ptr, # *Pointer* to first input vector.
72             y_ptr, # *Pointer* to second input vector.
73             output_ptr, # *Pointer* to output vector.
74             n_elements, # Size of the vector.
75             BLOCK_SIZE: tl.constexpr, # Number of elements each program should process.
76             # NOTE: `constexpr` so it can be used as a shape value.
77             ):
78     # There are multiple 'programs' processing different data. We identify which program
79     # we are here:
80     pid = tl.program_id(axis=0) # We use a 1D launch grid so axis is 0.
81     block_start = pid * BLOCK_SIZE
82     offsets = block_start + tl.arange(0, BLOCK_SIZE)
83     mask = offsets < n_elements
84     x = tl.load(x_ptr + offsets, mask=mask)
85     y = tl.load(y_ptr + offsets, mask=mask)
86     output = x + y
87     tl.store(output_ptr + offsets, output, mask=mask)
88
89 ...
90 # Divide the input data into blocks of BLOCK_SIZE elements. 1D grid is used here.
91 grid = lambda meta: (triton.cdiv(n_elements, meta['BLOCK_SIZE']), )
92 add_kernel[grid](x, y, output, n_elements, BLOCK_SIZE=1024)
```

Triton Programming Model

- **Two-level Parallelism** mechanism
 - A. Parallelism primitives inside program
 - Vectorized Operations (loading, computing, storing)
 - No explicit management of threads
 - B. Programs launched concurrently
 - Equivalent to thread block in HIP/CUDA

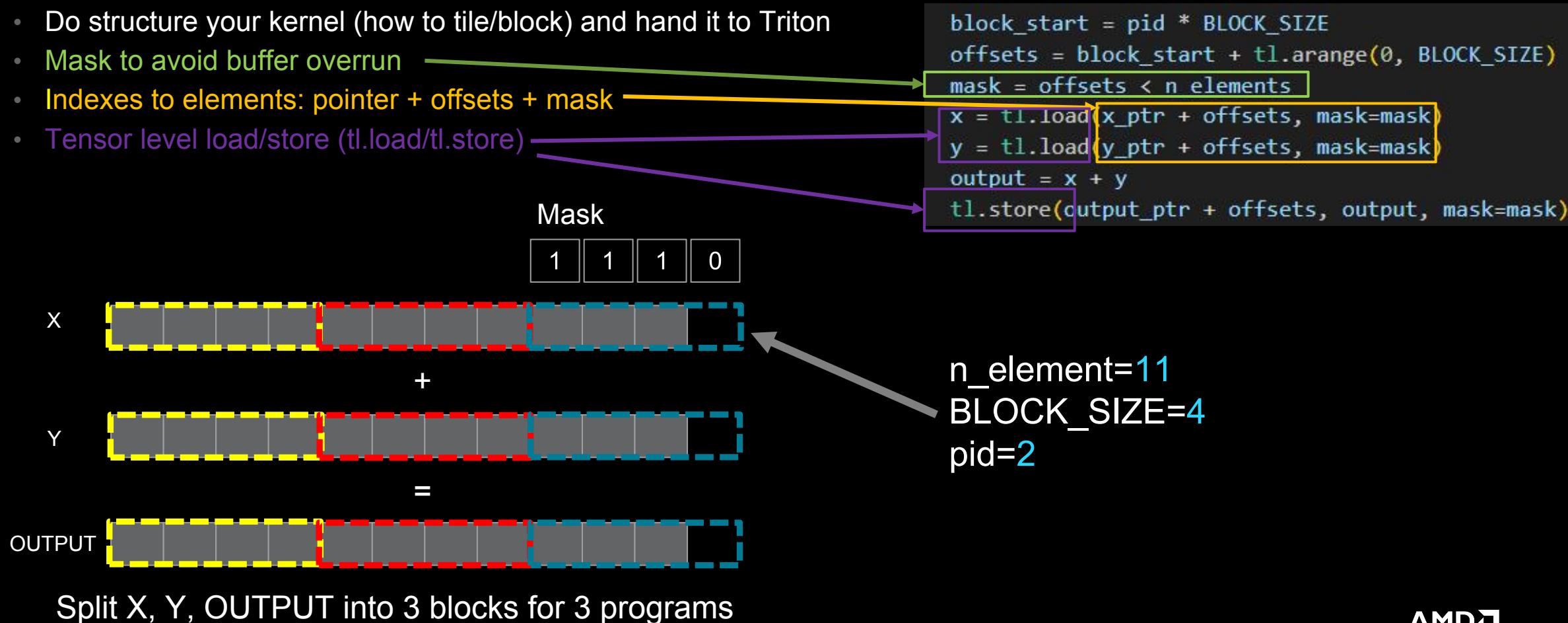


Split X, Y, OUTPUT into 3 blocks for 3 programs

```
70  @triton.jit
71  def add_kernel(x_ptr, # *Pointer* to first input vector.
72  #           y_ptr, # *Pointer* to second input vector.
73  #           output_ptr, # *Pointer* to output vector.
74  n_elements, # Size of the vector.
75  BLOCK_SIZE: tl.constexpr, # Number of elements each program should process.
76  # NOTE: `constexpr` so it can be used as a shape value.
77  ):
78  # There are multiple 'programs' processing different data. We identify which program
79  # we are here:
80  pid = tl.program_id(axis=0) # We use a 1D launch grid so axis is 0.
81  block_start = pid * BLOCK_SIZE
82  offsets = block_start + tl.arange(0, BLOCK_SIZE)
83  mask = offsets < n_elements
84  x = tl.load(x_ptr + offsets, mask=mask)
85  y = tl.load(y_ptr + offsets, mask=mask)
86  output = x + y
87  tl.store(output_ptr + offsets, output, mask=mask)
88
89 ...
90  # Divide the input data into blocks of BLOCK_SIZE elements. 1D grid is used here.
91  grid = lambda meta: (triton.cdiv(n_elements, meta['BLOCK_SIZE']), )
92  add_kernel[grid](x, y, output, n_elements, BLOCK_SIZE=1024)
```

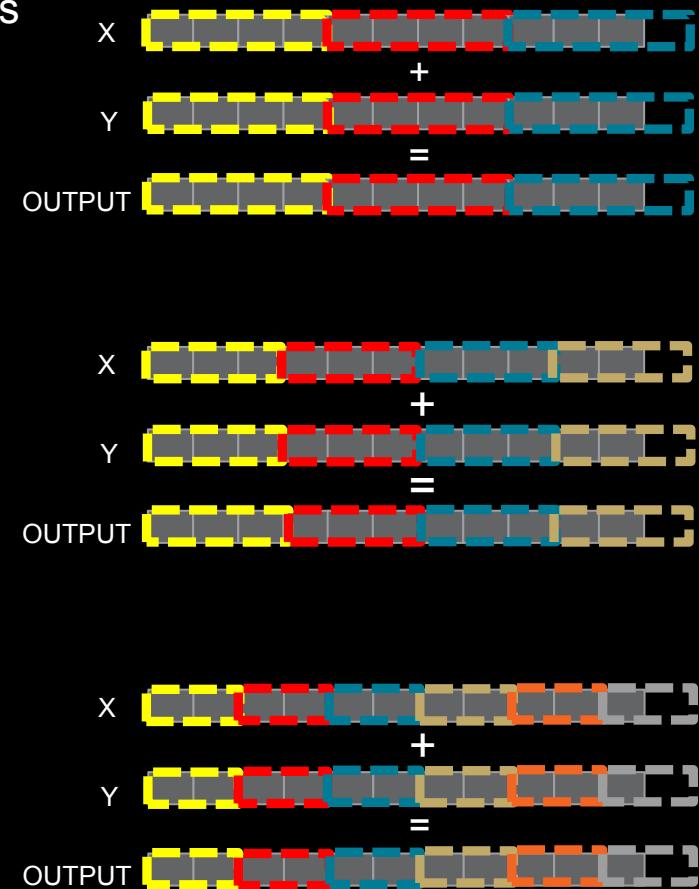
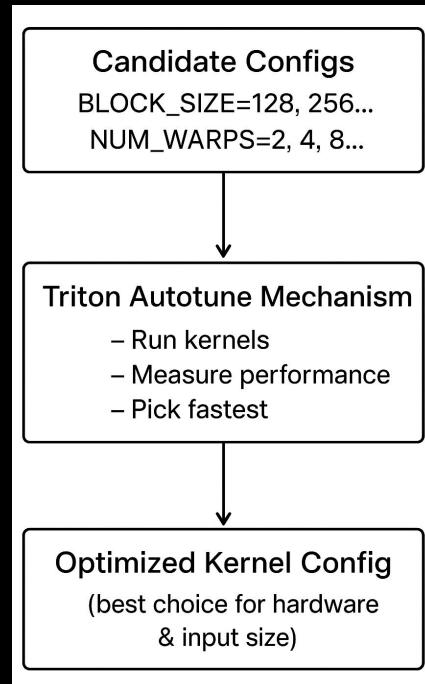
Triton Programming Model

- Memory hierarchy awareness
 - No explicit data transfer management between global/shared/register memory
 - Do structure your kernel (how to tile/block) and hand it to Triton
 - Mask to avoid buffer overrun
 - Indexes to elements: pointer + offsets + mask
 - Tensor level load/store (tl.load/tl.store)



Triton Programming Model

- Meta-parameters **Autotune**
 - **BLOCK_SIZE / NUM_WARPS**
 - Finding the best meta parameters
 - `triton.autotune` decorator



```
@triton.autotune(  
    configs=[  
        triton.Config({"BLOCK_SIZE": 128}, num_warps=2),  
        triton.Config({"BLOCK_SIZE": 256}, num_warps=4),  
        triton.Config({"BLOCK_SIZE": 512}, num_warps=4),  
        triton.Config({"BLOCK_SIZE": 1024}, num_warps=8),  
    ],  
    key=["n_elements"], # re-autotune when problem size changes  
)  
@triton.jit  
def add_kernel(x_ptr, # *Pointer* to first input vector.  
               y_ptr, # *Pointer* to second input vector.  
               ):
```

AMD