



第8届 AI+ Development
Digital Summit

AI+研发数字峰会

拥抱AI 重塑研发

11月14-15日 | 深圳





2026 AI+ PRODUCT INNOVATION SUMMIT

01.16-17 · ShangHai

AI+产品创新峰会



Track 1: AI 产品战略与创新设计

从0到1的AI原生产品构建

论坛1: AI时代的用户洞察与需求发现

论坛2: AI原生产品战略与商业模式重构

论坛3: Agentic AI产品创新与交互设计

2-hour Speech: 回归本质



用户洞察的第一性

——2小时思维与方法论工作坊

在数字爆炸、AI迅速发展的时代，
仍然考验“看见”的“同理心”



Track 2: AI 产品开发与工程实践

从1到10的工程化落地实践

论坛1: 面向Agent智能体的产品开发

论坛2: 具身智能与AI硬件产品

论坛3: AI产品出海与本地化开发

Panel 1: 出海前瞻



“出海避坑地图”圆桌对话

——不止于翻译:

AI时代的出海新范式



Track 3: AI 产品运营与智能演化

从10到100的AI产品运营

论坛1: AI赋能产品运营与增长黑客

论坛2: AI产品的数据飞轮与智能演化

论坛3: 行业爆款AI产品案例拆解

Panel 2: 失败复盘



为什么很多AI产品“叫好不叫座”?

——从伪需求到真价值:

AI产品商业化落地的关键挑战

智能重构产品 数据驱动增长
Reinventing Products with Intelligence, Driven by Data

80+

业界大咖

汇聚产品大咖的真知灼见

40+

创新案例

开拓全新AI+产品视角

10+

分论坛

涵盖产品到商业增长的全链路

800+

听众规模

共同探索产品的智能重构



扫码查看会议详情

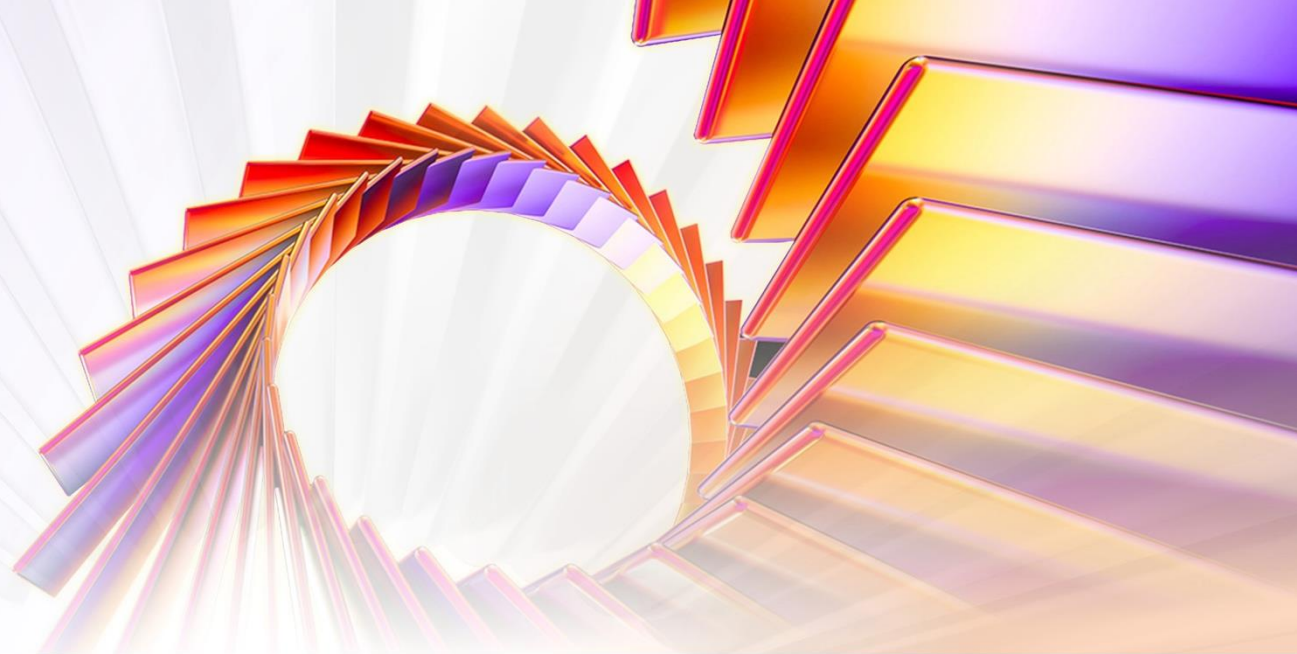


| 11月14-15日 | 深圳

第8届 AI+ Development
Digital Summit

AI+研发数字峰会

拥抱AI 重塑研发



PD分离架构：从大规模到小规模 普惠场景实践

车漾 | 阿里云



车漾

阿里云 高级技术专家

阿里巴巴云原生应用平台高级技术专家，从事 Kubernetes 和容器相关产品的开发，重点探索利用容器技术加速异构计算、深度学习、边缘计算等广泛场景方案的交付与落地，同时是对于开源社区的积极参与者。他是 CNCF 旗下开源项目 Fluid 的创始人之一，也是核心维护者。也是业界第一个 GPU 共享调度的主要作者和维护者。他还是 Alluxio 开源项目的管理委员会成员 (PMC Member)，Kubernetes, Docker 和 Kubeflow 等社区的积极贡献者。

目录

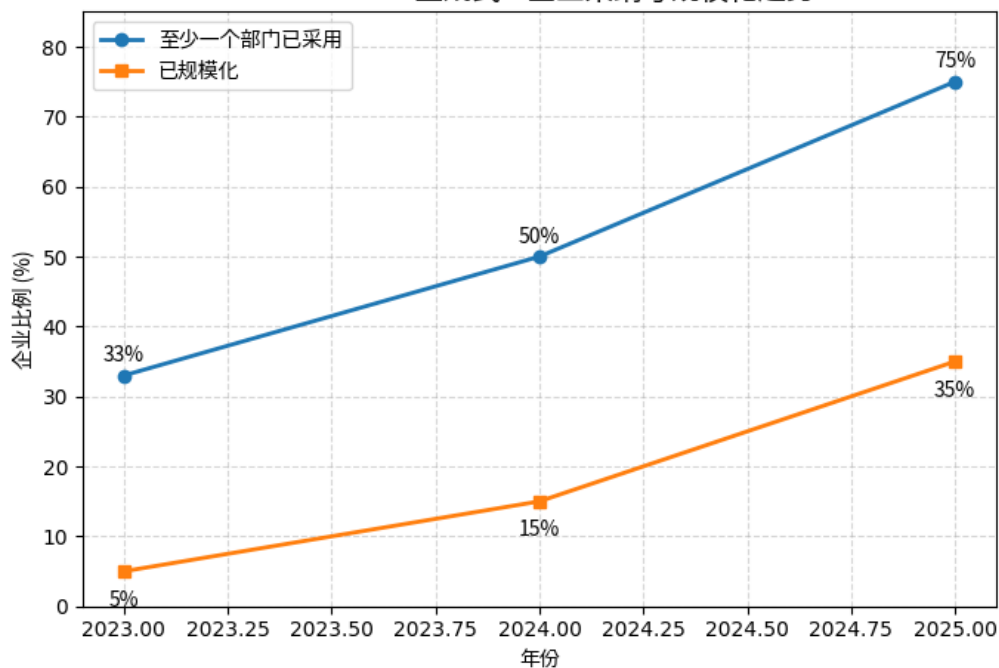
CONTENTS

- I. 大模型服务落地的趋势与挑战
- II. 微服务化管理LLM推理
- III. AI任务调度
- IV. AI负载可观测
- V. 集群稳定性
- VI. 未来工作

PART 01

大模型服务落地的趋势与挑战

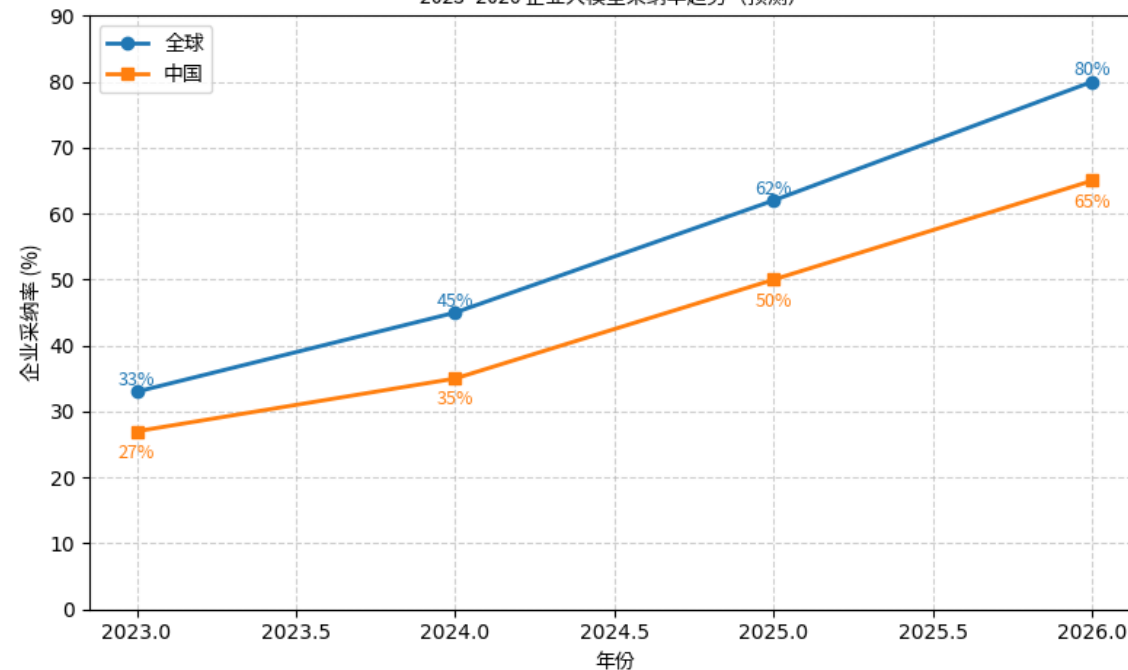
2023-2025 生成式AI企业采纳与规模化趋势



从 试验 到 规模化

1. 2023 年生成式 AI 的爆发试点期，大量企业进行 PoC。
2. 2024 年进入早期规模化，出现跨部门应用与多场景覆盖。
3. 2025 年将进入“生产系统化”阶段，**生成式 AI/LLM 被视为基础设施**。

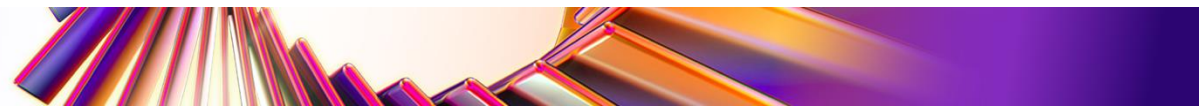
2023-2026 企业大模型采纳率趋势 (预测)



趋势解读：

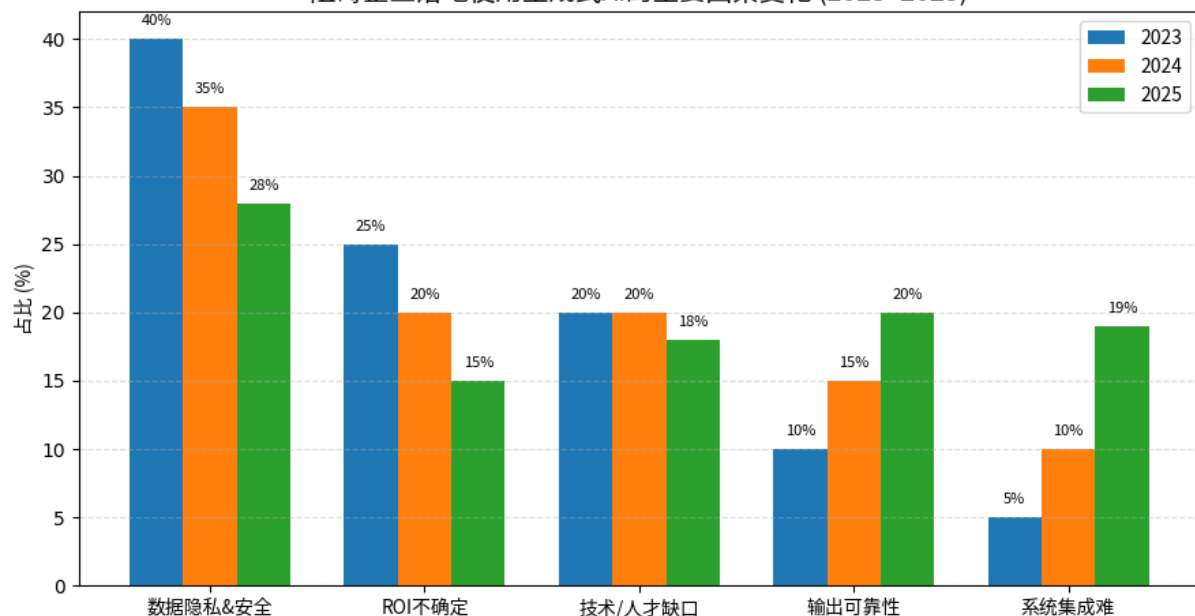
1. 全球：23年33%快速攀升，26年预计80%通过API、SaaS或自研使用大模型。
2. 中国：起步稍晚但增长快，预计26年中大型企业采纳率接近 65%。
3. 增速高点24-25年，得益于行业解决方案成熟化、成本下降及员工培训体系出现。

(图表均由 GPT-5-Chat-0807 分析麦肯锡、Gartner、赛迪、艾瑞等调研报告，以及厂商工程实践和白皮书，生成代码绘制)



企业落地Gen-AI的趋势

阻碍企业落地使用生成式AI的主要因素变化 (2023-2025)



推理层 (Inference)

推理延迟高 / 吞吐量低
输出不稳定 / 幻觉
上下文长度受限
多模态支持不足

训练层 (Training / Fine-tuning)

算力需求高 / 成本高
分布式训练复杂 / 易出错
微调周期长 / 过拟合风险
同步异常 / 权重丢失

部署层 (Serving)

GPU集群稳定性差
跨节点通信瓶颈
冷启动耗时长
驱动与框架版本兼容障碍

数据层 (Data)

高质量训练/标注数据不足
数据清洗/脱敏成本高
流式数据接入难
合规处理复杂

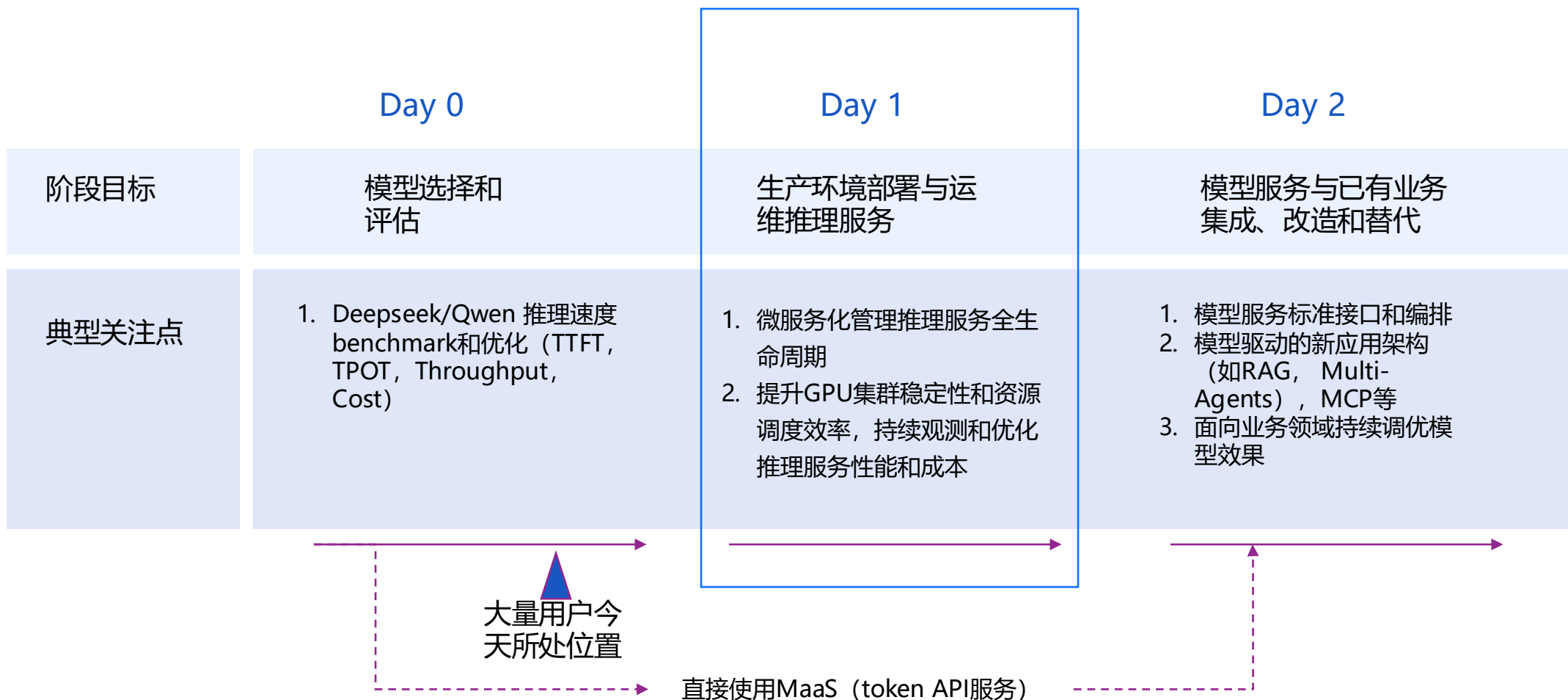
运维层 (MLOps / AIOps)

缺乏全链路监控告警
模型版本管理困难
缺少灰度发布/回滚机制
GPU能耗高 / 预算压力

(图表均由 GPT-5-Chat-0807 分析麦肯锡、Gartner、赛迪、艾瑞等调研报告, 以及厂商工程实践和白皮书, 生成代码绘制)



企业落地LLM服务的路线图





02

微服务化管理LLM推理

LLM推理负载 RoleBasedGroup

▶ ACK LLM Serving Stack – 在生产环境中部署LLM推理服务

↓ 85%

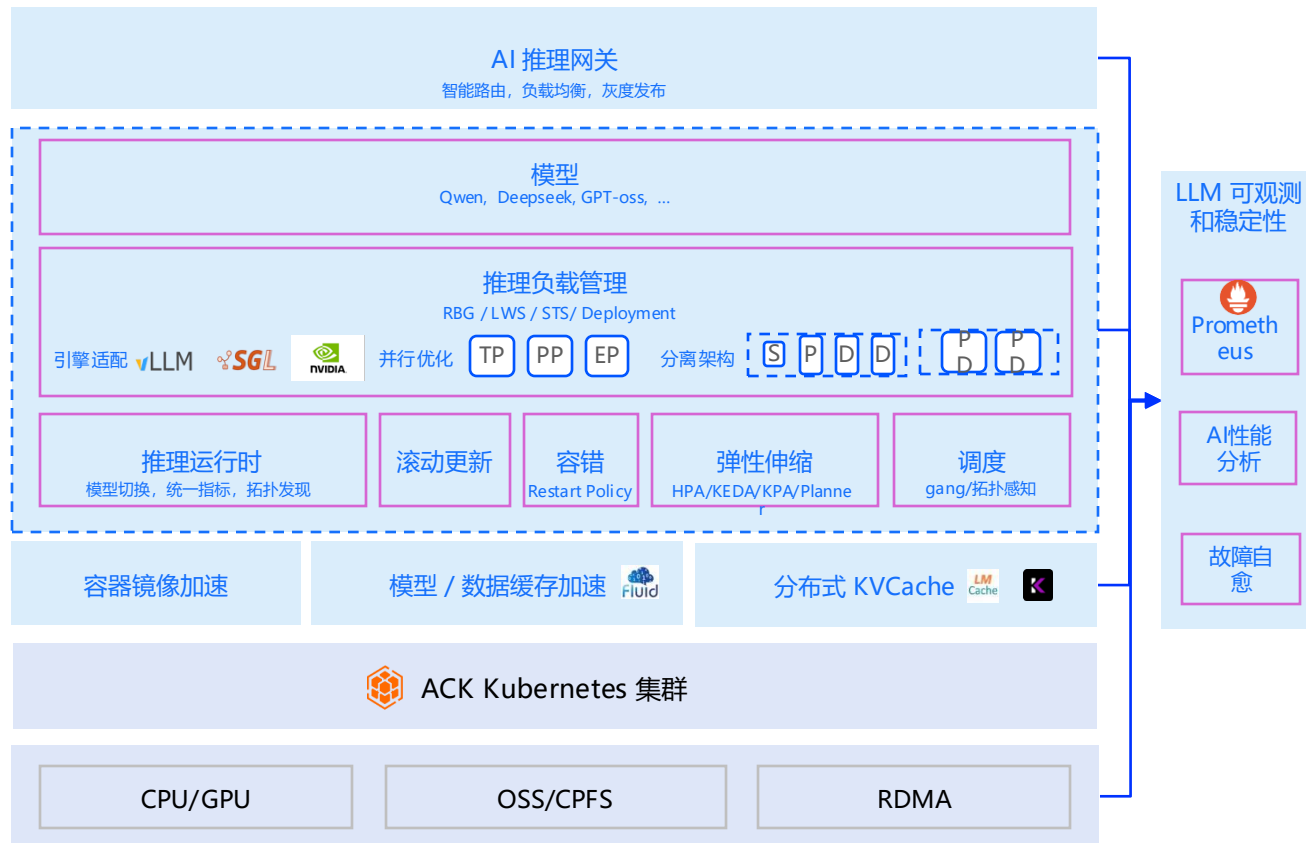
Cold start latency

↓ 60%

64并发, Avg TPOT

↑ 346%

卡均 GoodPut, TPOP < 30ms



LLM推理工作负载

RoleBasedGroup (RBG)

- K8s native API, 支持一个负载定义多种Role组合
- 支持各类分布式推理架构: PD 分离, PP/TP/EP并行
- 兼容 vLLM/SGLang+Mooncake、Dynamo、llm-d 等

监控与自动伸缩

- 统一监控指标: TTFT, TPOT, Throughput, Request Rate, KVCache usage, ISL, OSL
- 基于SLA, waiting requests, KVCache等, 按需扩缩容P和D

冷启动加速

- Fluid分布式缓存及模型预热能力, 加速模型加载85%以上

模型感知智能路由

Gateway API with Inference Extension

- 支持 LoRA Adapter、PrefixCache Aware、Waiting request num、Model Priority等推理请求路由策略
- 多模型版本的灰度发布能力

基于开源 PD 分离方案, Qwen3-32B 在 8 卡 GPU 机器测试, Goodput 提升 3.46 倍 (单卡吞吐 370.06 Token/s 提升到 1283.27 Token/s), 平均 TPOT 下降 60% (66.81ms 下降到27ms)

LLM推理部署方式

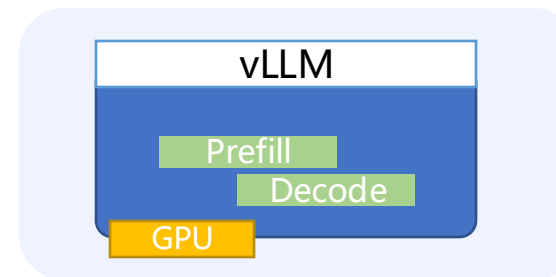
➤ Prefill (预填充) – 理解问题阶段

从用户输出的Prompt到生成第一个Token的过程

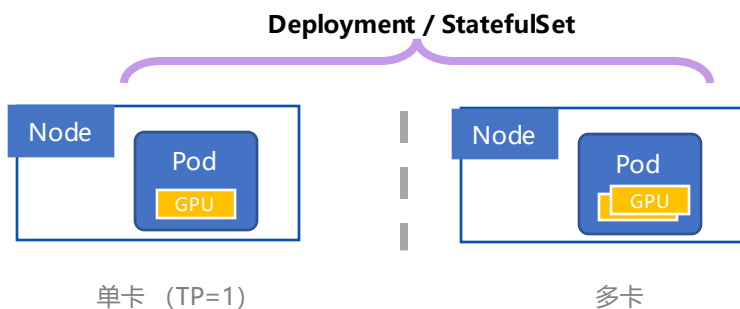
➤ Decode (解码) – 逐字回答阶段

生成第二个Token到最后一个Token的过程

Prefill-Decode一体
(单实例)

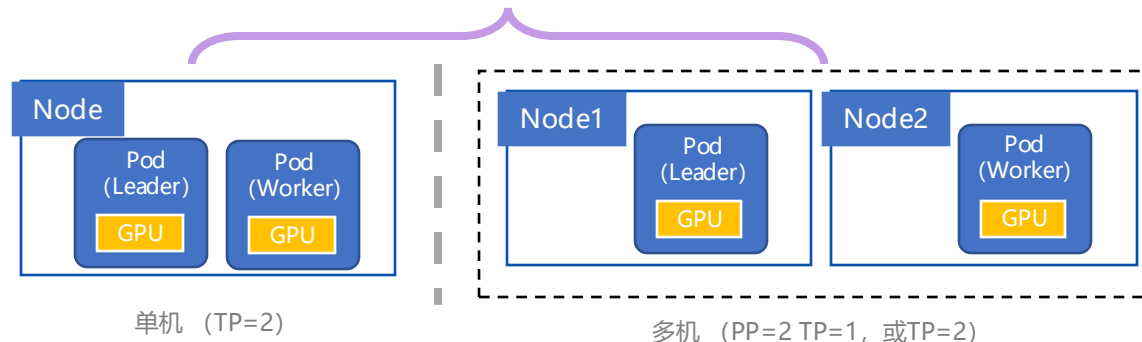


单实例 单Pod



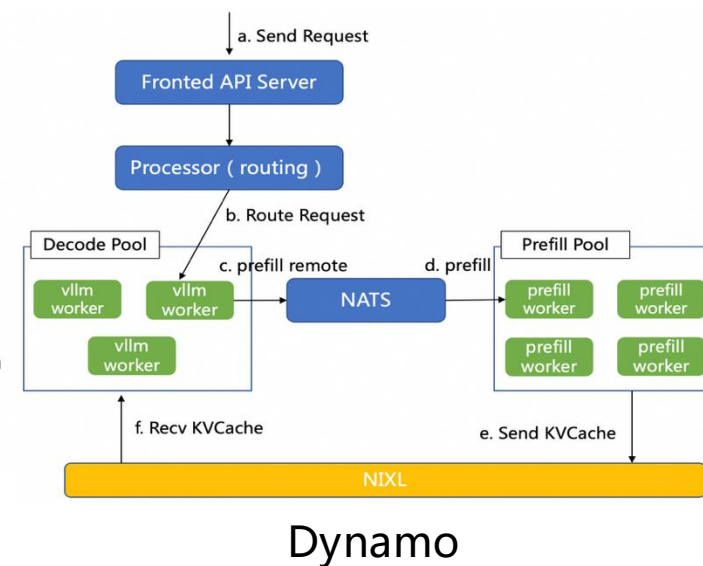
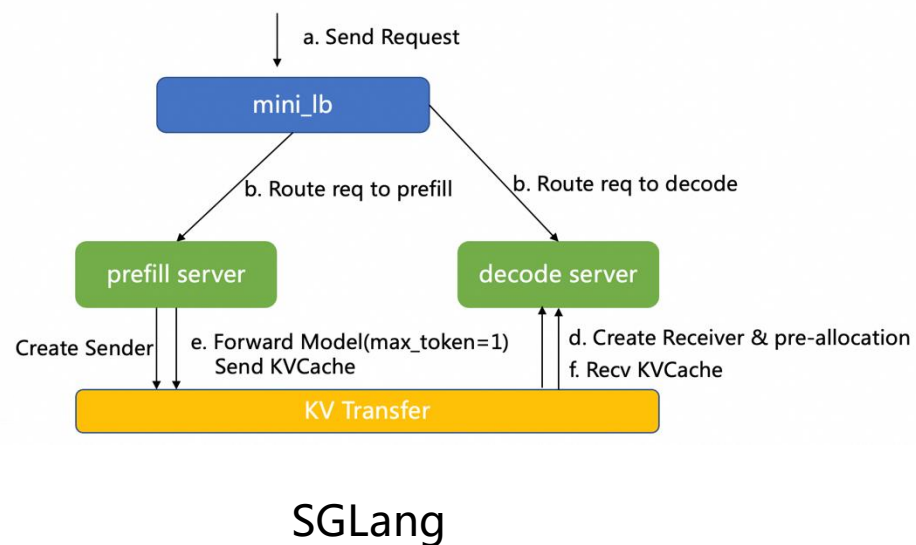
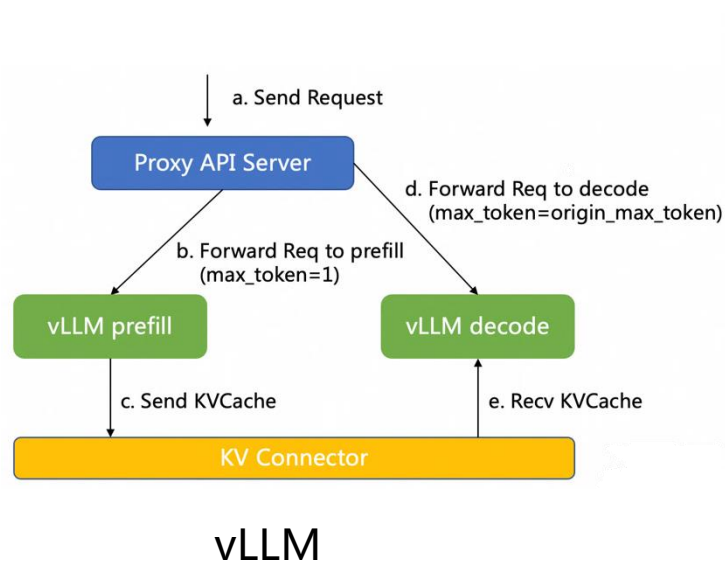
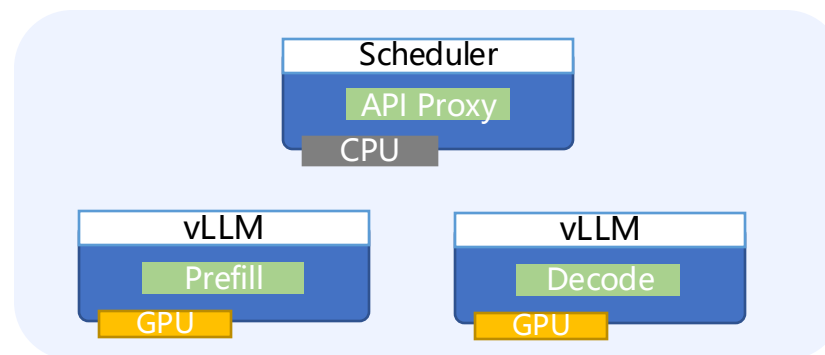
单实例 多Pod

LeaderWorkerSet (LWS)



LLM推理部署方式

Prefill-Decode 分离
(单实例)



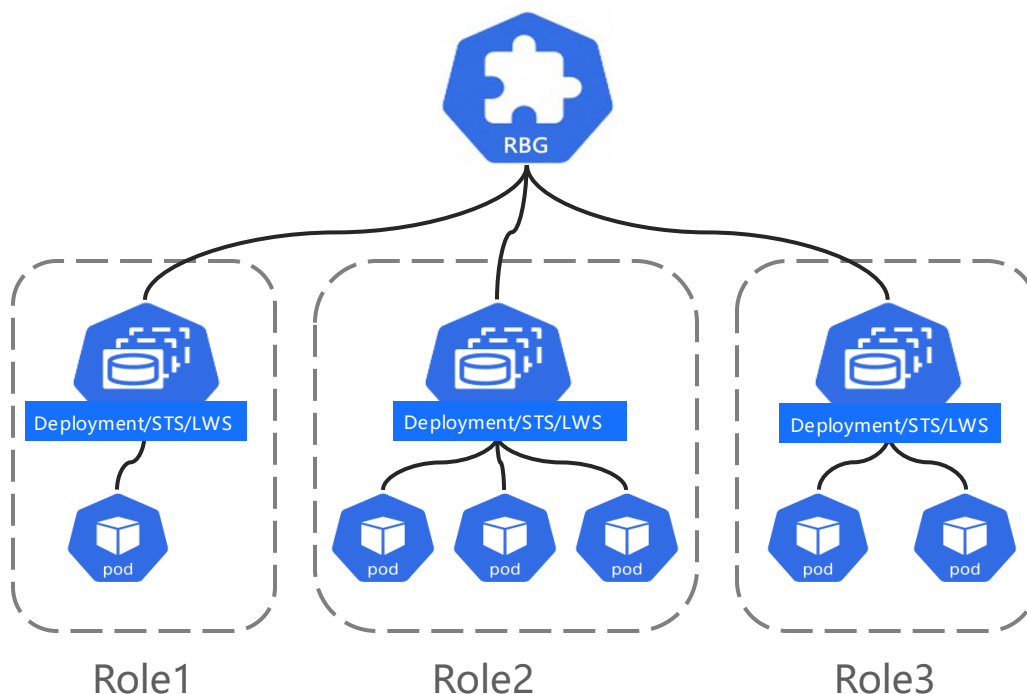
➤ 核心问题：传统微服务架构无法满足大模型推理的“高性能+强状态+快迭代”三重需求，需要的是HPC和微服务的结合。

- 架构不确定性：
 -  PD分离、AF分离、Mooncake等架构快速迭代
- 性能敏感：
 -  强依赖GPU/RDMA拓扑亲和性，TTFT/TPOT要求严苛
- 组件关系强依赖：
 -  Prefill-Decode需比例协同，启动顺序和版本严格同步
- 升级和弹性效率低：
 -  每天5%时间消耗在升级扩缩容，GPU闲置
- 资源利用率低：
 -  潮汐特征明显，峰值与谷值差距大



▶ RoleBasedGroup (RBG)

- RBG - 多个Role的集合, Role间相互协作, 共同构成一个推理应用
- 每个Role可以为单/多实例, 每个实例可以为单Pod或多Pod (TP, EP并行), Role可以指定使用STS或LWS部署
- 用于Kubernetes集群中, 部署和管理LLM推理服务的生命周期



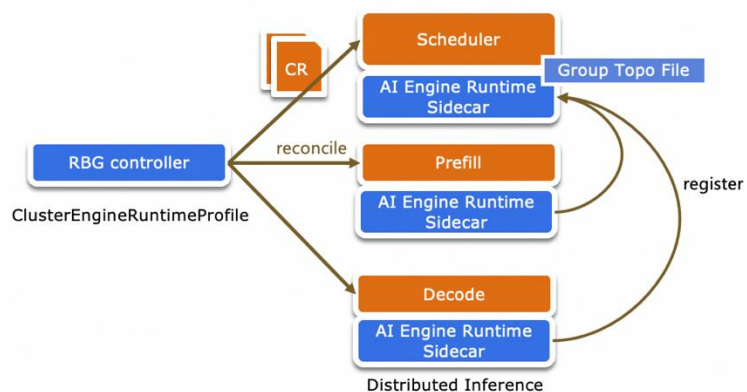
```
1  apiVersion: workloads.x-k8s.io/v1alpha1
2  kind: RoleBasedGroup
3  metadata:
4    name: pd-disagg
5  spec:
6    roles:
7    - name: scheduler
8      replicas: 1
9      template:
10       spec:
11         containers:
12         - name: scheduler
13           ...
14
15    - name: prefill
16      replicas: 3
17      template:
18       spec:
19         containers:
20         - name: vllm
21           ...
22
23    - name: decode
24      replicas: 2
25      template:
26       spec:
27         containers:
28         - name: vllm
29           ...
```

架构设计 – RoleBasedGroup (RBG)

Deploy and Manage LLM Inference – RoleBasedGroup (RBG)

多角色负载编排

- 基于角色的启动控制
- Role粒度动态扩缩容
- GangScheduling: all-or-nothing, 避免资源浪费和调度死锁。

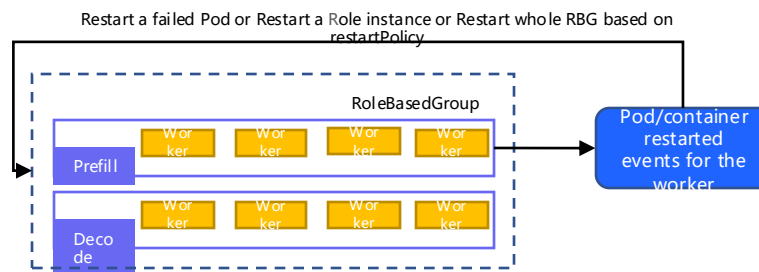


使用RBG编排部署PD分离架构的LLM推理服务

统一LLM推理运行时

- 🌐 自动服务发现和拓扑信息注入
- 🔑 多级错误处理
- 🛡️ 统一可观测指标和监控大盘

```
roles:
- name: sglang
  restartPolicy: None | Recreate RBG On Pod Restart | Recreate Role Instance On Pod Restart
```



自定义restartPolicy: Recreate Entire Role, Restart Entire Group, or Manual Control.

```
apiVersion: workloads.x-k8s.io/v1alpha1
kind: RoleBasedGroup
metadata:
  name: sglang
spec:
  roles:
  - name: worker
    replicas: 2
    template:
      spec:
        containers:
        - name: sglang
          image: lmsysorg/sglang: latest
          command:
          - python
          - -m
          - sglang.launch_server
          - --model-path
          - meta-llama/Meta-Llama-3.1-8B-
          Instruct
          - --dist-init-addr
          - $(GROUP_NAME)-$(ROLE_NAME)-
          0:5000
          - --nnodes
          - 2
          - --node-rank
          - $(ROLE_INDEX)
          - --tp
          - "2" # Size of Tensor Parallelism
          - --trust-remote-code
          - --host
          - "0.0.0.0"
          - --port
          - "80000"
```

► 核心能力- RoleBasedGroup (RBG)

► RBG是SGLang社区开源的一种面向多角色协同工作场景设计的工作负载，旨在解决**多角色场景**下的生命周期管理难题，包括多角色的**创建、调度、升级、故障自愈、服务发现**等能力。



项目地址: <https://github.com/sgl-project/rbg>



模版抽象 减少冗余定义

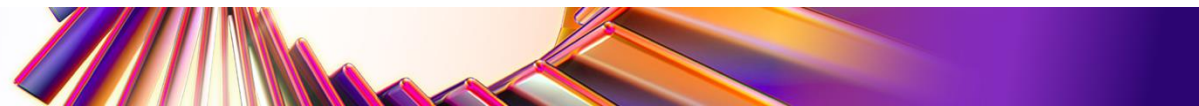
```
apiVersion: workloads.x-k8s.io/v1alpha1
kind: RoleBasedGroup
metadata:
  name: sglang-pd-demo
spec:
  roleTemplate:
    - name: engine
      template:
        spec:
          containers:
            - name: router
              command:
                - ...
              env:
                - name: DEMO_KEY
                  value: "demo_value"
              volumeMounts:
                - name: model
                  mountPath: /model
          volumes:
            - name: model
              ...
```

自定义角色依赖与负载类型

```
roles:
- name: router
  replicas: 1
  dependencies: ["prefill", "decode"]
  workload:
    apiVersion: apps/v1
    kind: statefulset #Custom Defined
  template:
    spec:
      containers:
        - name: router
          command:
            - ...
          env:
            image: router:latest
```

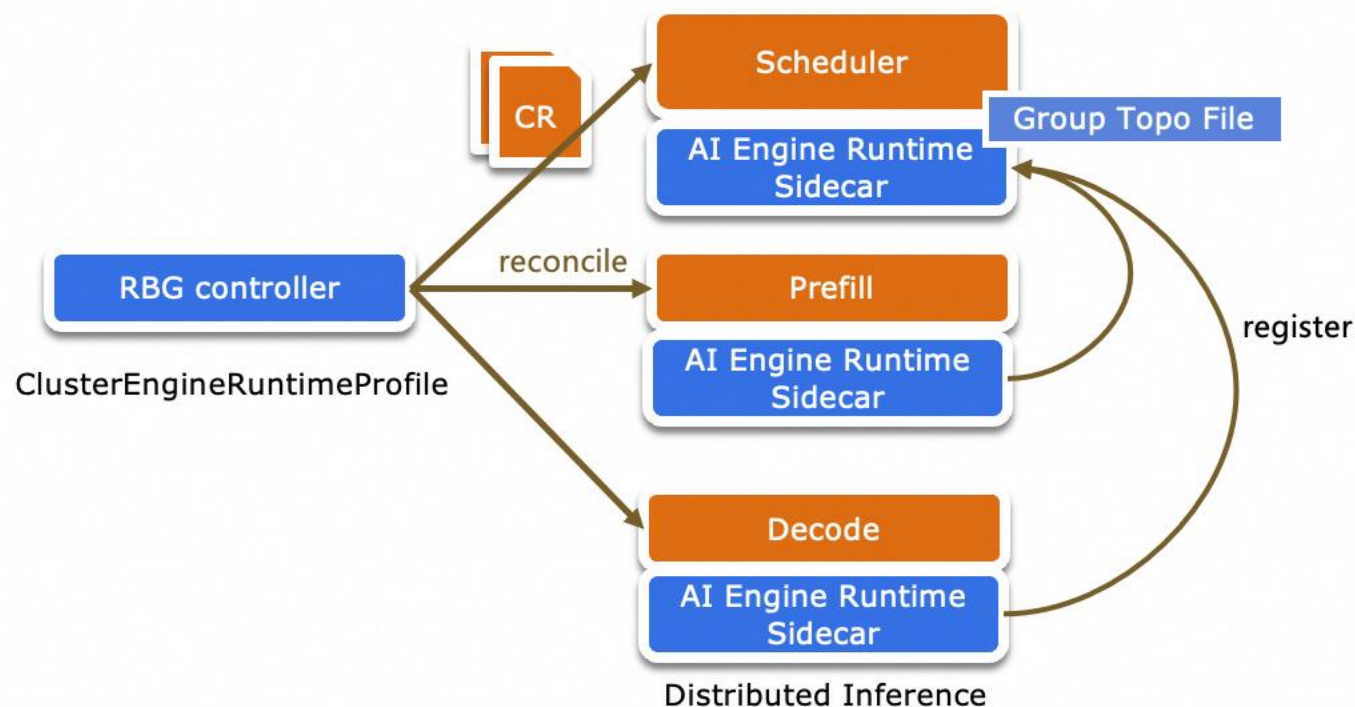
EngineRuntime灵活注入

```
- name: prefill
  replicas: 2
  templateRef: engine
  templatePatch:
    spec:
      containers:
        - name: engine
          command:
            - ... --prefill
  engineRuntimes:
    - profileName: patio-runtime
- name: decode
  replicas: 2
  templateRef: engine
  templatePatch:
    spec:
      containers:
        - name: engine
          command:
            - ... --decode
  engineRuntimes:
    - profileName: patio-runtime
```

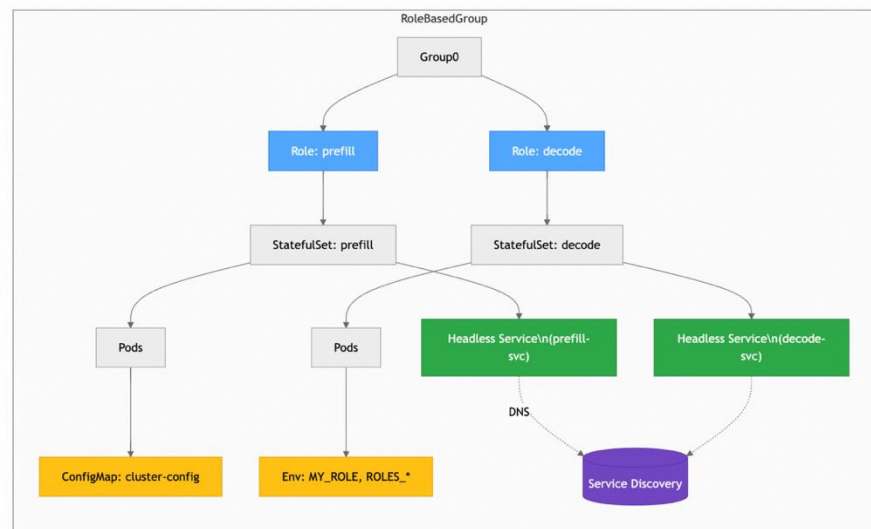


▶ RBG LLM推理运行时 (Engine Runtime)

- 统一监控指标和端点，兼容vLLM和SGLang等推理引擎。
- 支持LoRA模型的动态加载与卸载。
- 将RBG全局拓扑信息转换为特定引擎格式（例如vLLM、SGLang、Dynamo）。



▶ Role拓扑信息自动注入



Inject Configmap
into Pod,
Dynamic change

```
# /etc/rbg/config.yaml
group:
  name: nginx-cluster-0
  roles:
    - leader
    - worker
  size: 2
  roles:
    leader:
      instances:
        - address: leader-0.nginx-cluster-0-leader
          size: 1
    worker:
      instances:
        - address: worker-0.nginx-cluster-0-worker
          ports:
            http: 80
          size: 1
```

全局信息

Inject ENV into Pod,
Describe current
Pod

```
env:
- name: GROUP_NAME
  value: dynamo
- name: ROLE_INDEX
  value: 0
- name: ROLE_NAME
  value: prefill
```

自身信息

```
apiVersion: workloads.x-k8s.io/v1alpha1
kind: RoleBasedGroup
metadata:
```

```
  name: sglang
```

```
spec:
```

```
  roles:
```

```
    - name: worker
```

```
      replicas: 2
```

```
  template:
```

```
    spec:
```

```
      containers:
```

```
        - name: sglang
```

```
          image: lmsysorg/sglang: latest
```

```
          command:
```

```
            - python
```

```
            - -m
```

```
            - sglang.launch_server
```

```
            - --model-path
```

```
            - meta-llama/Meta-Llama-3.1-8B-Instruct
```

```
            - --dist-init-addr
```

```
            - $(GROUP_NAME)-$(ROLE_NAME)-0:5000
```

```
            - --nnodes
```

```
            - 2
```

```
            - --node-rank
```

```
            - $(ROLE_INDEX)
```

```
            - --tp
```

```
            - "2" # Size of Tensor Parallelism
```

```
            - --trust-remote-code
```

```
            - --host
```

```
            - "0.0.0.0"
```

```
            - --port
```

```
            - " 80000"
```



Engine Runtime Profile

- 兼容不同推理引擎: SGLang, vLLM, Mooncake, Dynamo等
- 屏蔽配置和启动参数差异

```
apiVersion: workloads.x-k8s.io/v1alpha1
kind: ClusterEngineRuntimeProfile
metadata:
  name: mooncake-runtime
spec:
  volumes:
    - emptyDir: {}
      name: patio-group-config
  containers:
    - name: patio-runtime
      image: registry-cn-hangzhou.ack.aliyuncs.com/dev/patio-runtime:v0.2.0
      env:
        - name: TOPO_TYPE
          value: Mooncake
        - name: TOPO_SERVER_ROLE
          value: scheduler
```

Mooncake Runtime 公共配置

```
apiVersion: workloads.x-k8s.io/v1alpha1
kind: RoleBasedGroup
metadata:
  name: sglang
spec:
  roles:
    - name: scheduler
      engineRuntimes:
        - profileName: mooncake-runtime
      template:
        ...
        volumeMounts:
          - name: patio-group-config
            mountPath: /etc/patio
        ...
```

挂载任务拓扑文件给 scheduler

```
...
- name: prefill
  engineRuntimes:
    - profileName: mooncake-runtime
  containers:
    - name: patio-runtime
      args:
        - --instance-info={"data":{"worker_role":"prefill-only"}}
  template:
    ...
```

注册Mooncake P worker

```
...
- name: decode
  engineRuntimes:
    - profileName: mooncake-runtime
  containers:
    - name: patio-runtime
      args:
        - --instance-info={"data":{"worker_role":"decode-only"}}
  template:
    ...
```

注册Mooncake D worker

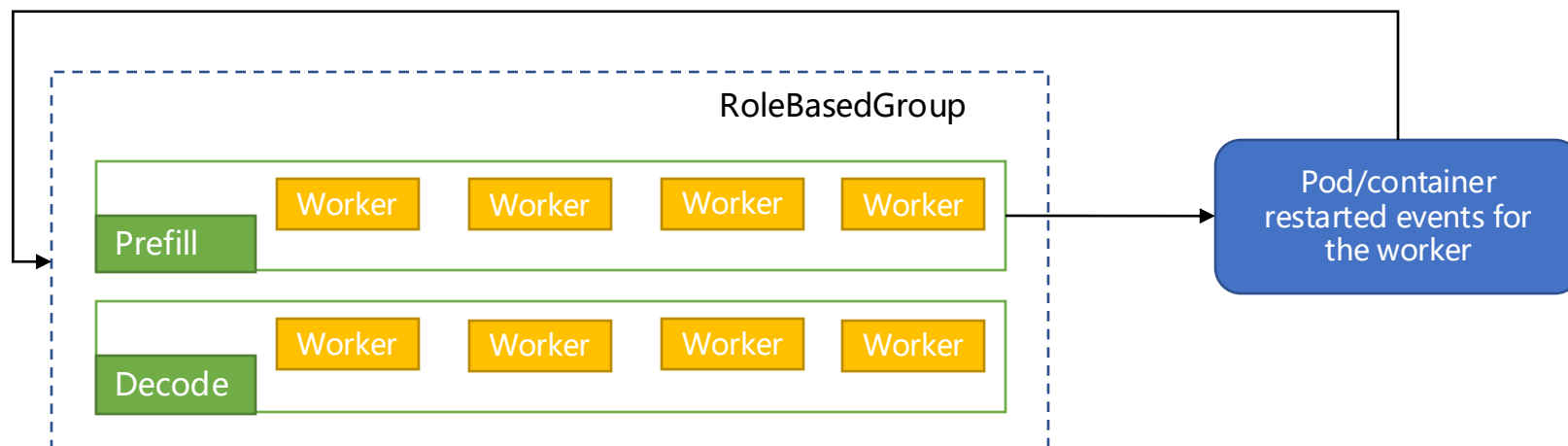


▶ RBG 错误处理 (RestartPolicy)

Configurable **restartPolicy**: Recreate Entire Role, Restart Entire Group, or Manual Control.

```
roles:  
- name: sglang  
  restartPolicy: None | RecreateRBGOnPodRestart | RecreateRoleInstanceOnPodRestart
```

Restart a failed Pod or Restart a Role instance or Restart whole RBG based on restartPolicy



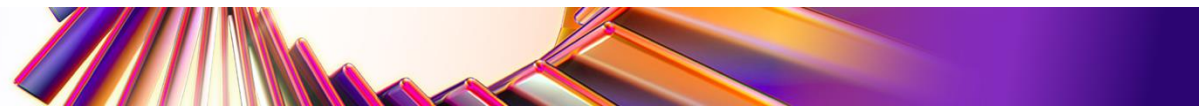
▶ RBG 滚动更新 (RolloutStrategy)

- 滚动更新对于需要zero downtime的在线服务至关重要。对于 LLM 推理服务而言，这一点同样关键。
- RBG 支持两种不同的配置参数：maxUnavailable 和 maxSurge。

```
rolloutStrategy:  
  type: RollingUpdate  
  rollingUpdateConfiguration:  
    maxUnavailable: 2  
    maxSurge: 2  
  replicas: 4
```

	Partition	Replicas	R-0	R-1	R-2	R-3	R-4	R-5	Note
Stage1	0	4	✓	✓	✓	✓			Before rolling update
Stage2	4	6	✗	✗	✗	✗	⌚	⌚	Rolling update started
Stage3	2	6	✗	✗	⌚	⌚	⌚	⌚	Partition changes from 4 to 2
Stage4	2	6	✗	✗	⌚	⌚	✓	⌚	Since the last Replica is not ready, Partition will not change
Stage5	0	6	⌚	⌚	⌚	⌚	✓	✓	Partition changes from 2 to 0
Stage6	0	6	⌚	⌚	⌚	✓	✓	✓	
Stage7	0	5	⌚	✓	⌚	✓	✓		Reclaim a Replica for the accommodation of unready ones
Stage8	0	4	✓	⌚	✓	✓			Release another Replica
Stage9	0	4	✓	✓	✓	✓			Rolling update completed

- ✓ Replica has been updated
- ✗ Replica hasn't been updated
- ⌚ Replica is in rolling update



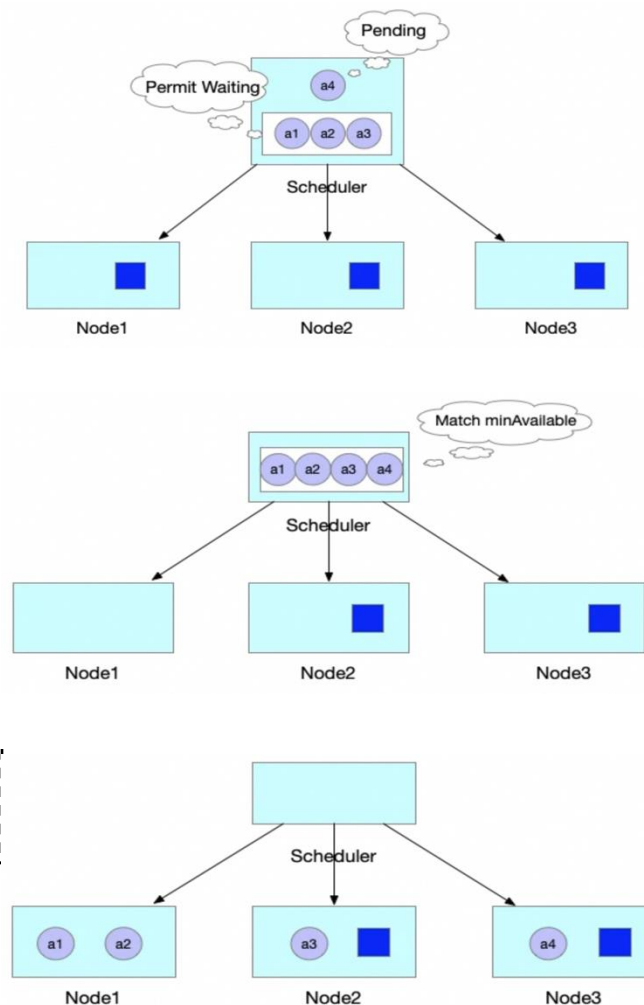
► RBG Gang调度

- Gang 调度是深度学习工作负载的关键特性，用于实现all-or-nothing的调度能力。Gang 调度可避免资源浪费和调度死锁。

```
apiVersion: workloads.x-k8s.io/v1alpha1
kind: RoleBasedGroup
metadata:
  name: nginx
spec:
  podGroupPolicy:
    kubeScheduling:
      scheduleTimeoutSeconds: 120
```

```
# PodGroup CRD spec
apiVersion: scheduling.sigs.k8s.io/v1alpha1
kind: PodGroup
metadata:
  name: nginx
spec:
  minMember: 3
  scheduleTimeoutSeconds: 10
---
# Add label "pod-group.scheduling.sigs.k8s.io/name" for pod
labels:
  pod-group.scheduling.sigs.k8s.io/name: nginx
```

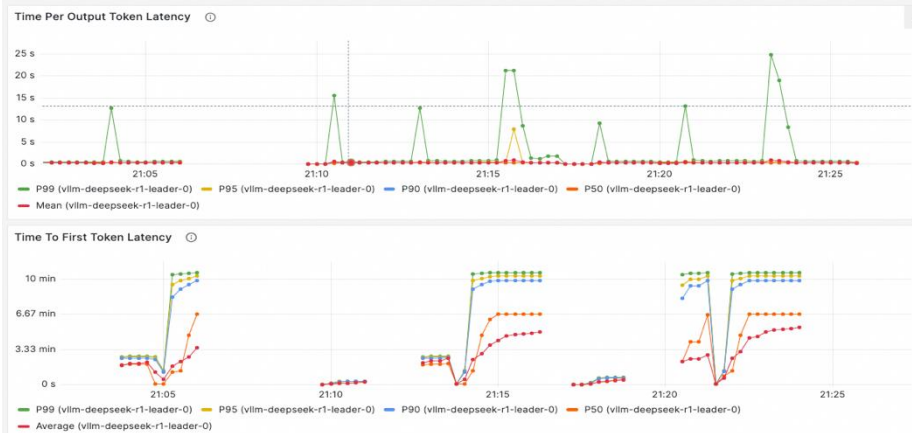
MinMember 等于RBG中的pods总数；
也可以在Role级别定义每个Role的PodGroup



支持 K8s NameSpace、推理服务、模型等 3 层级的大盘状态追踪

Datasource prom-ca421d04493f844058d8f8dbbd363d0df8 namespace default workload_name vllm-deepseek-r1 model_name /mnt/model/deepseek-r1

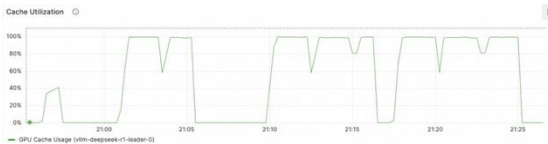
TTFT 和 TPOT 等 SLO 指标监控



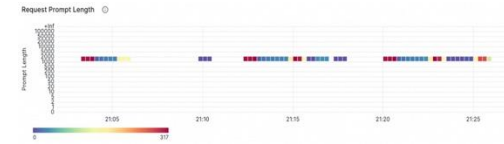
推理服务级别吞吐和请求速率监控



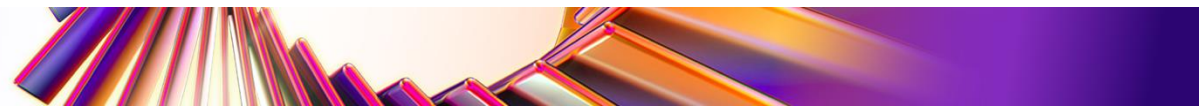
Pod-level分析性能瓶颈



分析业务请求特征 (输入长度和输出长度等)



容器支持实时监控 GPU 使用情况和 LLM 推理性能指标，结合弹性策略动态调整实例，提升资源利用率，保障服务稳定性，并为性能优化提供数据支撑。



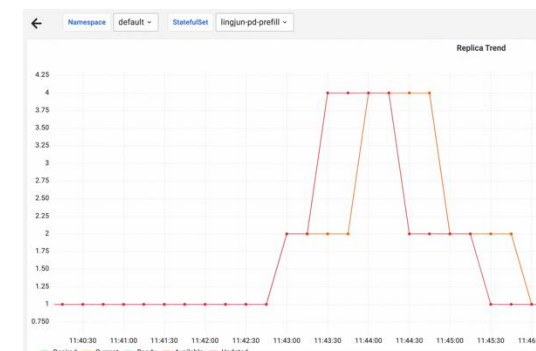
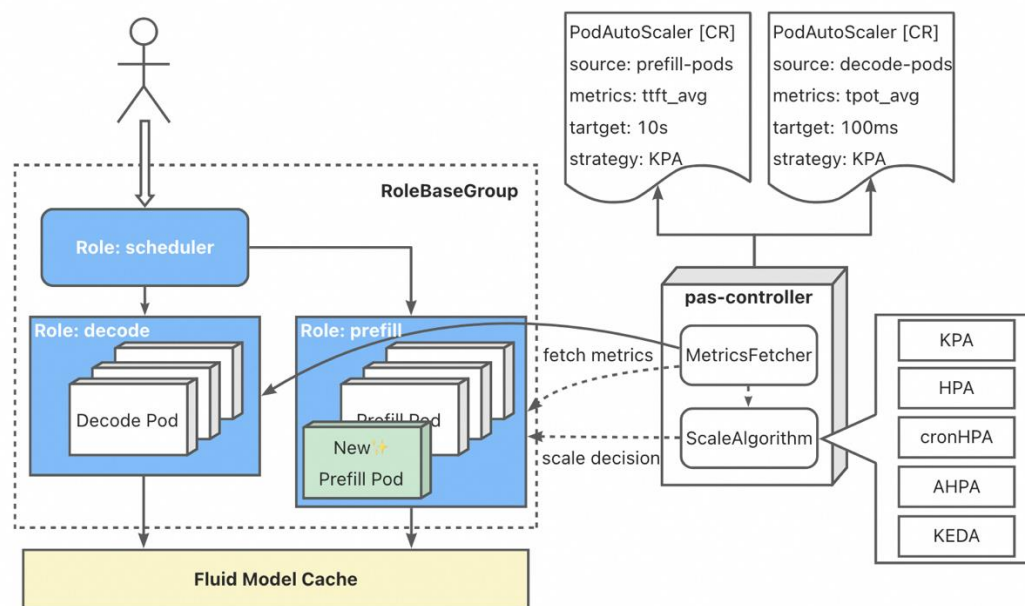
▶ RBG 实例弹性伸缩

- 为 Role 级别实例扩缩容提供 RBGScalingAdapter, 支持 HPA、KEDA 或 KPA 等各类弹性伸缩方案, 自动调整每种 Role 的副本数。
- 基于 SLO 指标, PD 分离下 scheduler/planner 统计指标等, 动态调整 Prefill 和 Decode worker 数, 自动优化推理性能

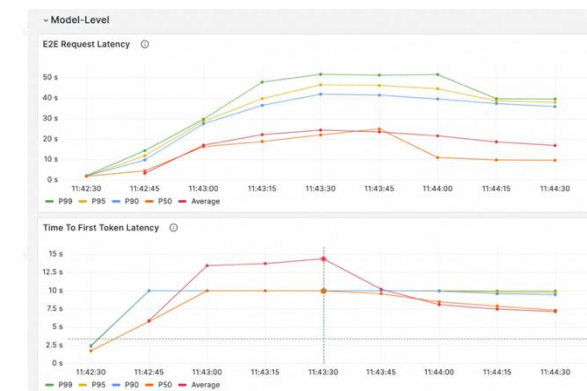
- 突发流量, 触发自动扩缩副本数

推荐扩缩容指标

- waiting request num
- running request num
- KV cache usage percentage
- TTFT
- TPOT



- 对应 SLO (TTFT, E2E Request Latency) 变化



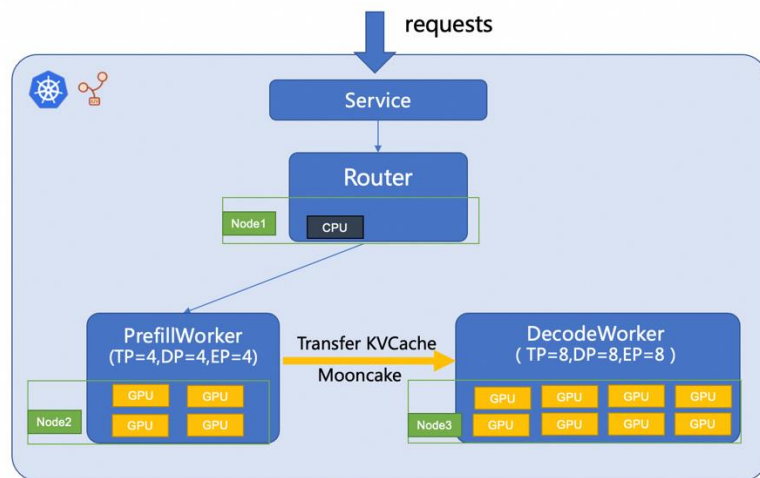
▶▶ 示例：RBG部署PD分离+EP并行推理，优化Qwen3-235B性能

Sample : Qwen3-235B Performance Optimization

```
apiVersion: workloads.x-k8s.io/v1alpha1
kind: RoleBasedGroup
metadata:
  name: qwen3-235b-sgl-pd
spec:
  roles:
  - name: scheduler
    replicas: 1
    dependencies: [ "decode", "prefill" ]
    template:
      spec:
        containers:
        - name: scheduler
          image: lmsysorg/sglang:v0.4.10.post2-cu126
          command:
            - sh
            - -c
            - >>
              python3 -m sglang.srt.disaggregation.mini_lb
              --prefill http://qwen3-235b-sgl-pd-prefill-0.qwen3-235b-sgl-pd-prefill:8000
              --decode http://qwen3-235b-sgl-pd-decode-0.qwen3-235b-sgl-pd-decode:8000
              --prefill-bootstrap-ports 34000 --host 0.0.0.0 --port 8000
```

```
- name: prefill
  replicas: 1
  template:
    spec:
      containers:
      - name: sglang-prefill
        image: lmsysorg/sglang:v0.4.10.post2-cu126
        command:
          - >>
            python3 -m sglang.launch_server --model-path /models/Qwen3-235B-A22B-FP8
            --port 8000 --host $(POD_IP) --cuda-graph-max-bs 128 --chunked-prefill-size 16000
            --ep-size 4 --dp-size 4 --tp-size 4 --disable-radix-cache |
            --attention-backend fa3 --enable-dp-attention --moe-a2a-backend deepseek
            --deepseek-mode normal --enable-dp-lm-head --enable-two-batch-overlap
            --disaggregation-mode prefill --disaggregation-bootstrap-port 34000
          - containerPort: 8000
        name: http
      - containerPort: 34000
        name: bootstrap
```

```
- name: decode
  replicas: 1
  template:
    spec:
      containers:
      - name: sglang-decode
        image: lmsysorg/sglang:v0.4.10.post2-cu126
        command:
          - >>
            python3 -m sglang.launch_server --model-path /models/Qwen3-235B-A22B-FP8
            --port 8000 --host $(POD_IP) --cuda-graph-max-bs 128 --enable-dp-attention
            --ep-size 8 --dp-size 8 --tp-size 8 --attention-backend flashinfer --enable-dp-lm-head
            --moe-a2a-backend deepseek --deepseek-mode low_latency
            --load-balance-method minimum_tokens --disaggregation-mode decode --disable-radix-cache
          ports:
            - containerPort: 8000
              name: http
```



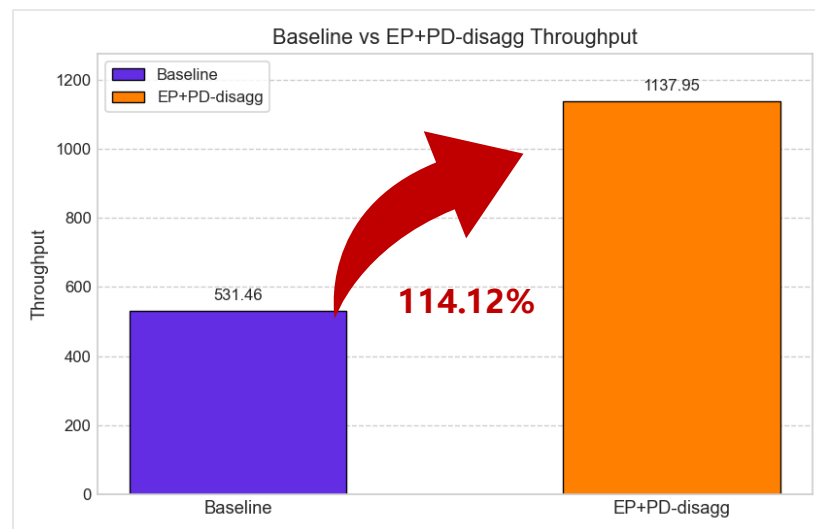
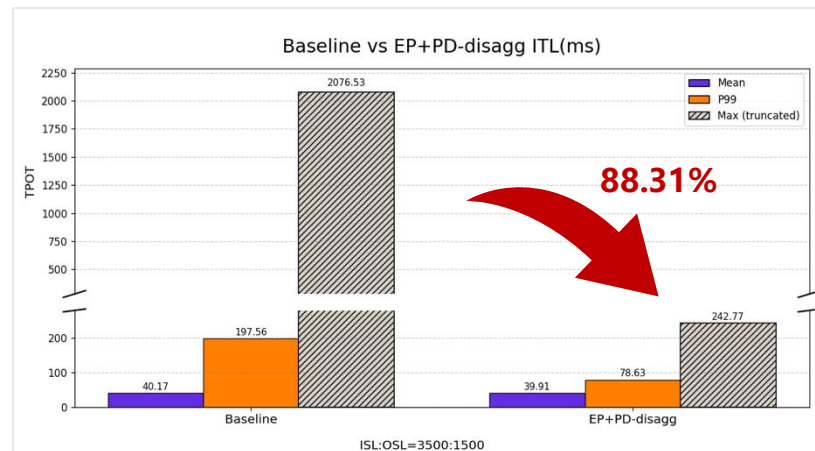
Max ITL 下降88.31%, 卡均吞吐提升114.12%

测试环境

- SGLang, Qwen3-235B FP8, 4 ~ 12 * 141G
- ISL:OSL = 3500:1500
- Baseline : TP=4

优化方案 (阿里云基础软件团队提供)

- PD分离: 基于 Mooncake + RDMA 实现
- 并行策略 DP + TP + EP
- DP 负载均衡优化, Scheduler 策略优化, Decoding 算子优化等



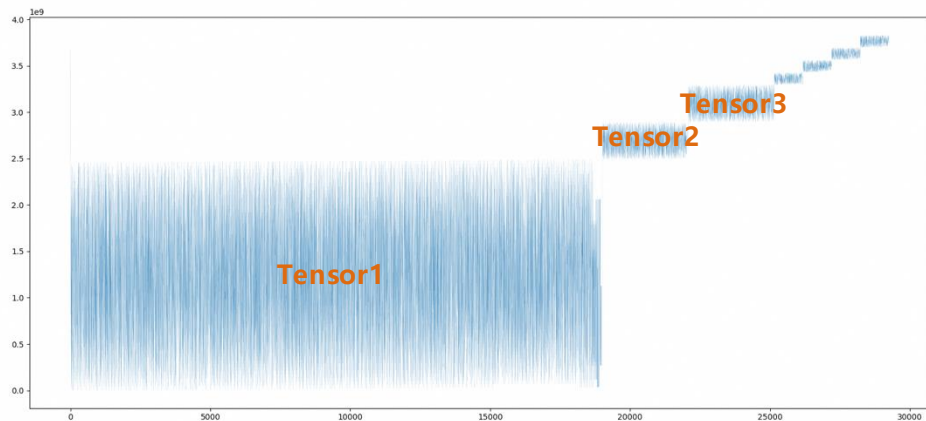
▶ RBG 使用Fluid加速模型加载

根据AI应用典型I/O特征加速数据/模型访问

现存问题

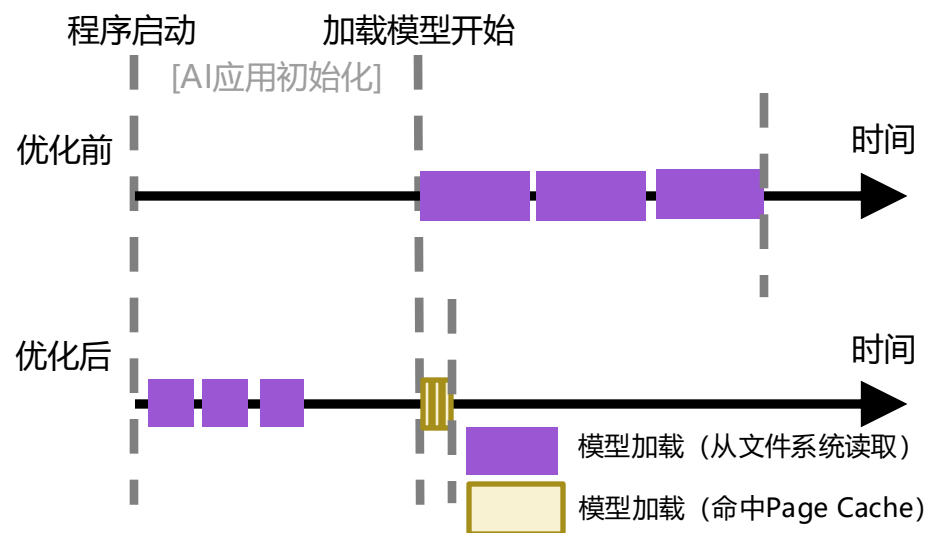
- LLM模型参数大多以Safetensors文件格式分发
- Safetensors文件的加载过程涉及大量跳变的随机读，受存算分离架构下文件系统实现影响，I/O效率低下。
- (机器节点带宽利用率不到20%)

模型参数文件(.safetensors) 读取过程read offset变化



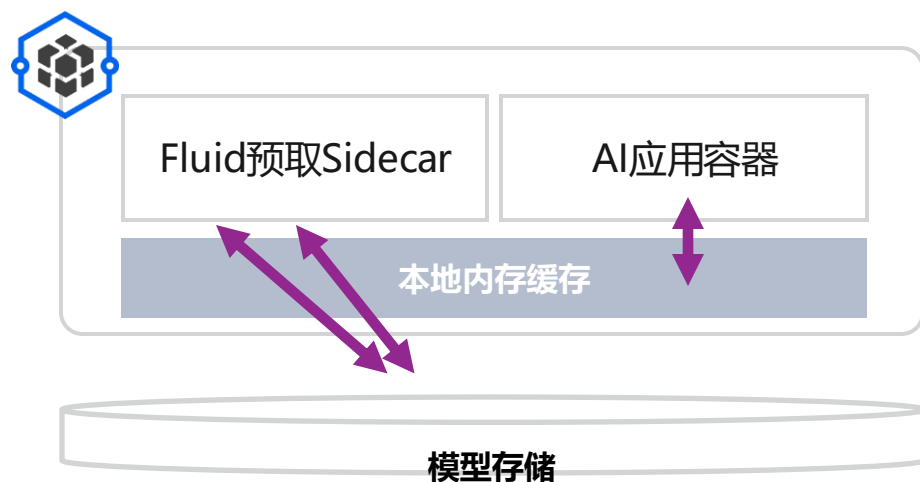
Fluid解决方案

- 将模型参数文件提前预读到随机读友好的本地内存缓存 (Page Cache)
- 多线程并行顺序预读，节点带宽利用率提升至80%以上



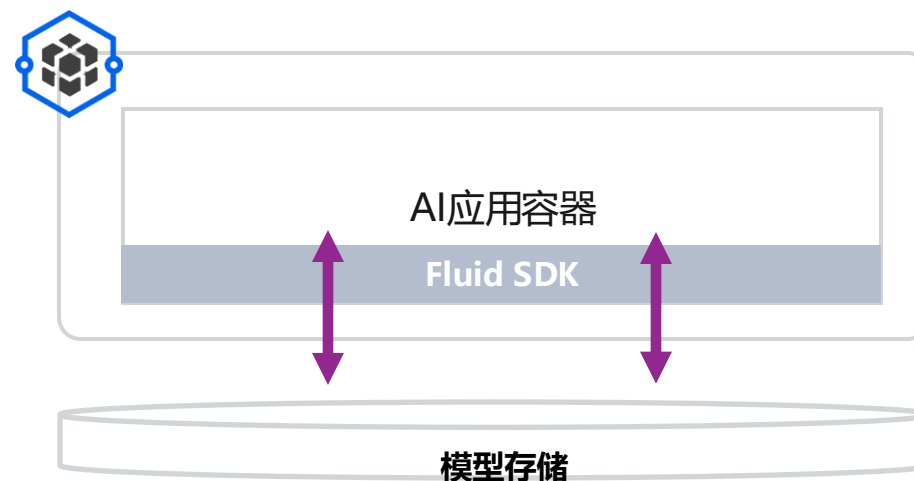
使用方法一：Sidecar预取

- 由Fluid自动对AI应用Pod注入Sidecar容器，由Sidecar发起多线程预取
- 优势：对应用无侵入，适用简单场景

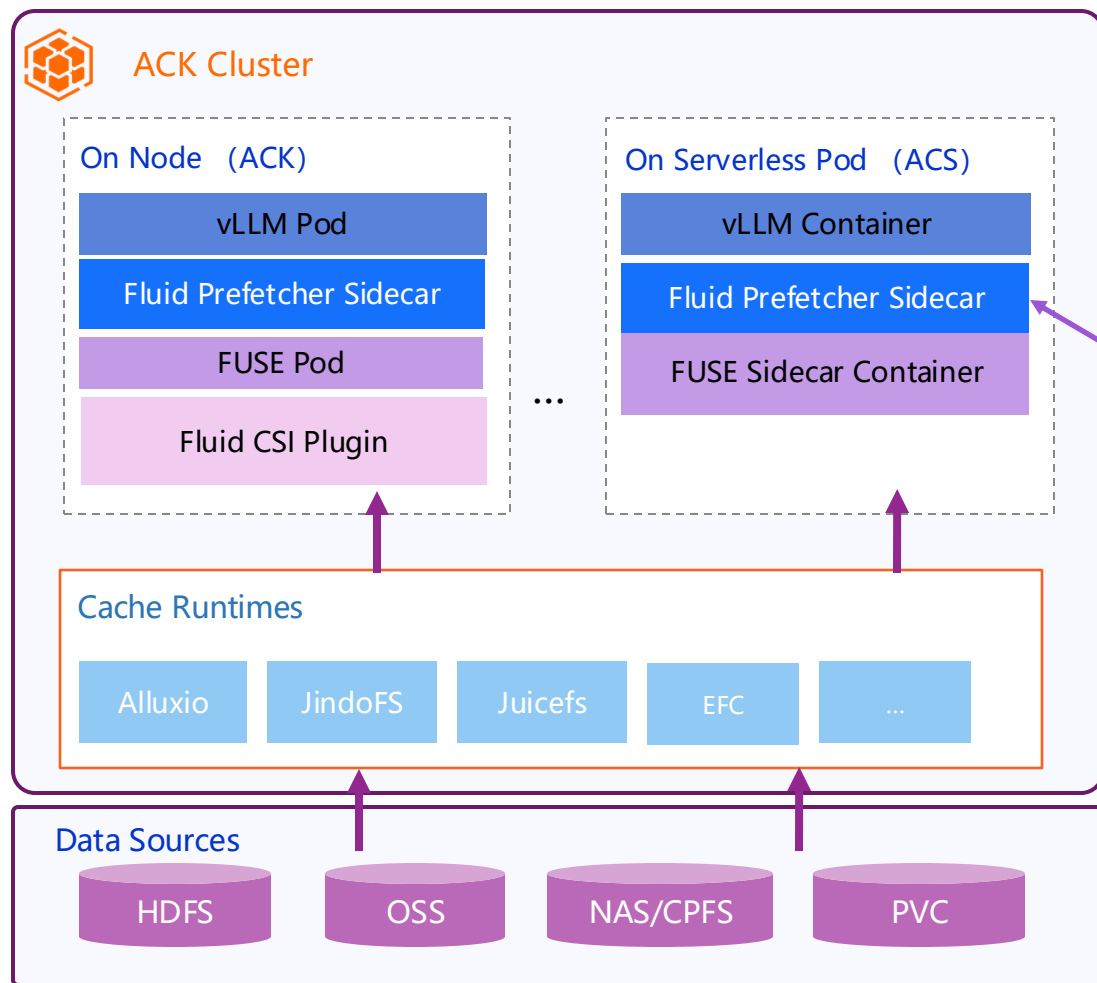


使用方法二：应用容器调用Fluid SDK

- 改造AI应用，根据业务需求适时调用Fluid SDK发起多线程预取
- 优势：支持运行过程中模型汰换等灵活场景



▶ RBG 使用Fluid加速模型加载



```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: sglang-deepseekl-r1-baseline
    name: sglang-deepseekl-r1-baseline
spec:
  template:
    metadata:
      labels:
        app: sglang-deepseekl-r1-baseline
        alibabacloud.com/fluid-sidecar-target: "acs"
      annotations:
        # 开启Fluid预取功能
        file-prefetcher.fluid.io/inject: "true"
        # Fluid预取的文件列表, 支持glob语法, 默认为所有挂载的Fluid PVC中的所有文件。
        file-prefetcher.fluid.io/file-list: "pvc://jfs-llm-model/*.safetensors"
        # 指定预取容器的超时时间, 如果模型较大,
        file-prefetcher.fluid.io/prefetch-timeout-seconds: "2400"
    spec:
      containers:
        - command:
            - sh
            - -c
            - python -m sglang.launch_server --model-path=/mount/model/
          image: <SGLANG_IMAGE>
          imagePullPolicy: IfNotPresent
```



➤ Deepseek-R1 671B模型加载时间

方案	4副本并发 模型加载耗时	耗时减少百分比
Baseline (OSS直接Fuse挂载)	39min	-
Fluid分布式缓存	58s (预热耗时) + 856s (Tensor耗时) = 914s = 15min	减少60.9%
Fluid预取优化	242s (预取耗时) + 72s (Tensor耗时) = 314s = 5min	减少86.6%
Fluid分布式缓存+预取 优化	58s (预热耗时) + 101s (预取耗时) + 70s (Tensor耗时) = 229s = 4min	减少90.2%

对OSS的实际带宽占用
4副本 * 641GiB / 242s
= **10.59GiB/s**

OSS理论带宽(100Gbps)
利用率已达84%

分布式缓存优化

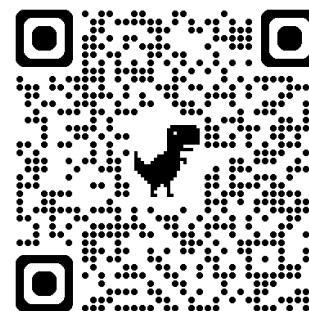
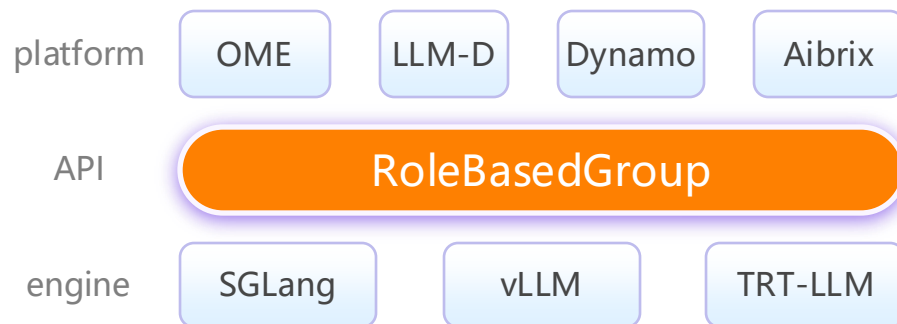
- 集群环境：
 - cn-beijing地域 (OSS理论带宽100Gbps)
 - 阿里云ACS集群
- 缓存配置：
 - 16 x ACS Pod, 每Pod 16c48g规格
 - JuiceFS分布式缓存
- PD分离推理服务配置：
 - sglang推理引擎
 - Prefill实例 (TP=8) : 2 x 8卡ACS Pod (184 CPU, 920GiB Mem)
 - Decode实例 (EP=16) : 2 x 8卡ACS Pod (184 CPU, 920GiB Mem)



▶ RBG 开源贡献 SGLang 社区

RBG is open source now

<https://github.com/sgl-project/rbg>



共建 K8s 管理大模型推理负载的标准 API



Dynamo

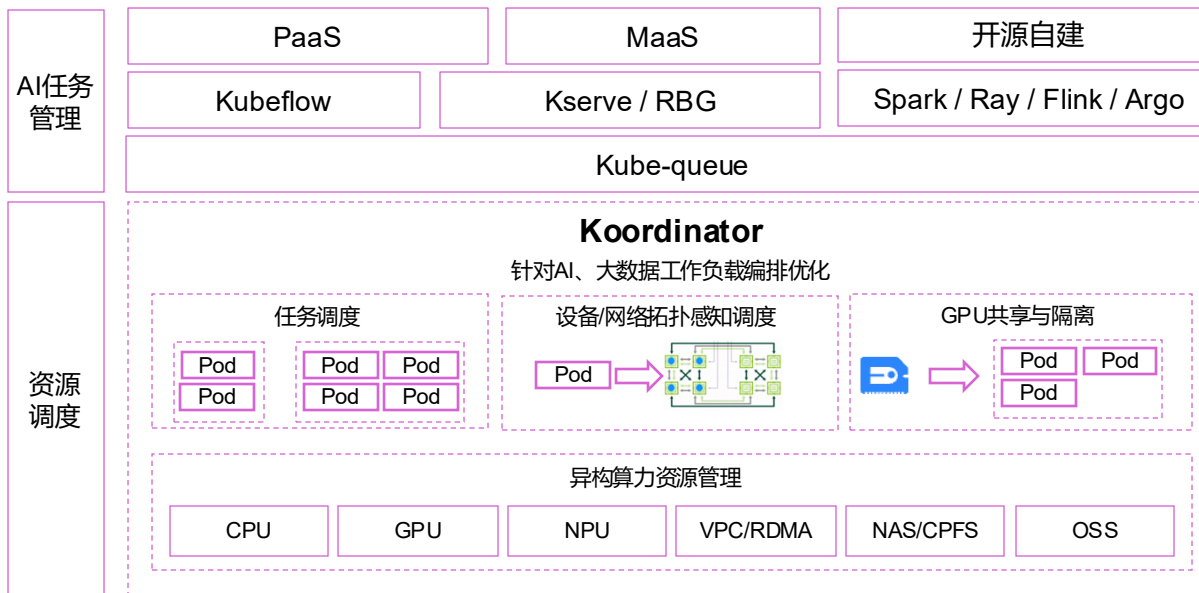


03

AI任务调度

任务队列, Resource Policy, 多集群调度

面向 AI、大数据负载的调度优化



Job Queue & Preemption

- 任务优先级与抢占
- 多级队列
- 大规模GPU训练场景性能优化
- 支持多集群调度扩展

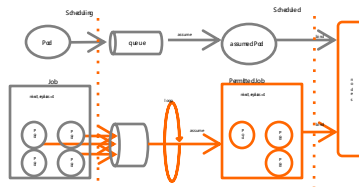


Resource Policy

- 多类型算力灵活编排
- 自定义资源优先级和配比
- 弹性资源自动补偿
- 缩容顺序可控
- 支持ECS, ACS, Spot

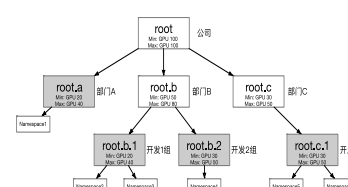
Gang scheduling

- 防止大作业挤占小作业
- 防止资源浪费和死锁



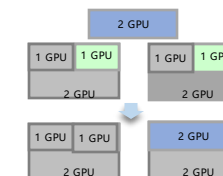
Capacity scheduling

- 多租户配额动态借还
- 多级结构, 灵活对应组织架构
- 兼容Yarn多队列设计



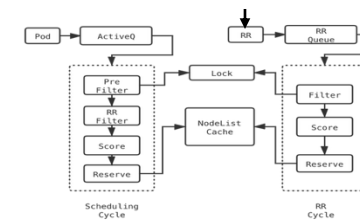
Binpack scheduling

- 防止小作业饿死大作业
- 有效避免资源碎片
- 提升GPU资源利用率



Resource Reservation

- 资源定向分配给特定任务
- 任务原地升级, 资源保持
- 提升调度结果确定性



多集群、跨地域调度GPU算力 – 应对大模型资源量挑战

- 单数据中心或云地域无法满足LLM的GPU资源量需求
- 多地域任务分发和数据同步，带来一致性、效率和复杂度挑战
- 大模型体积，推升多地域部署成本



优化GPU
资源调度

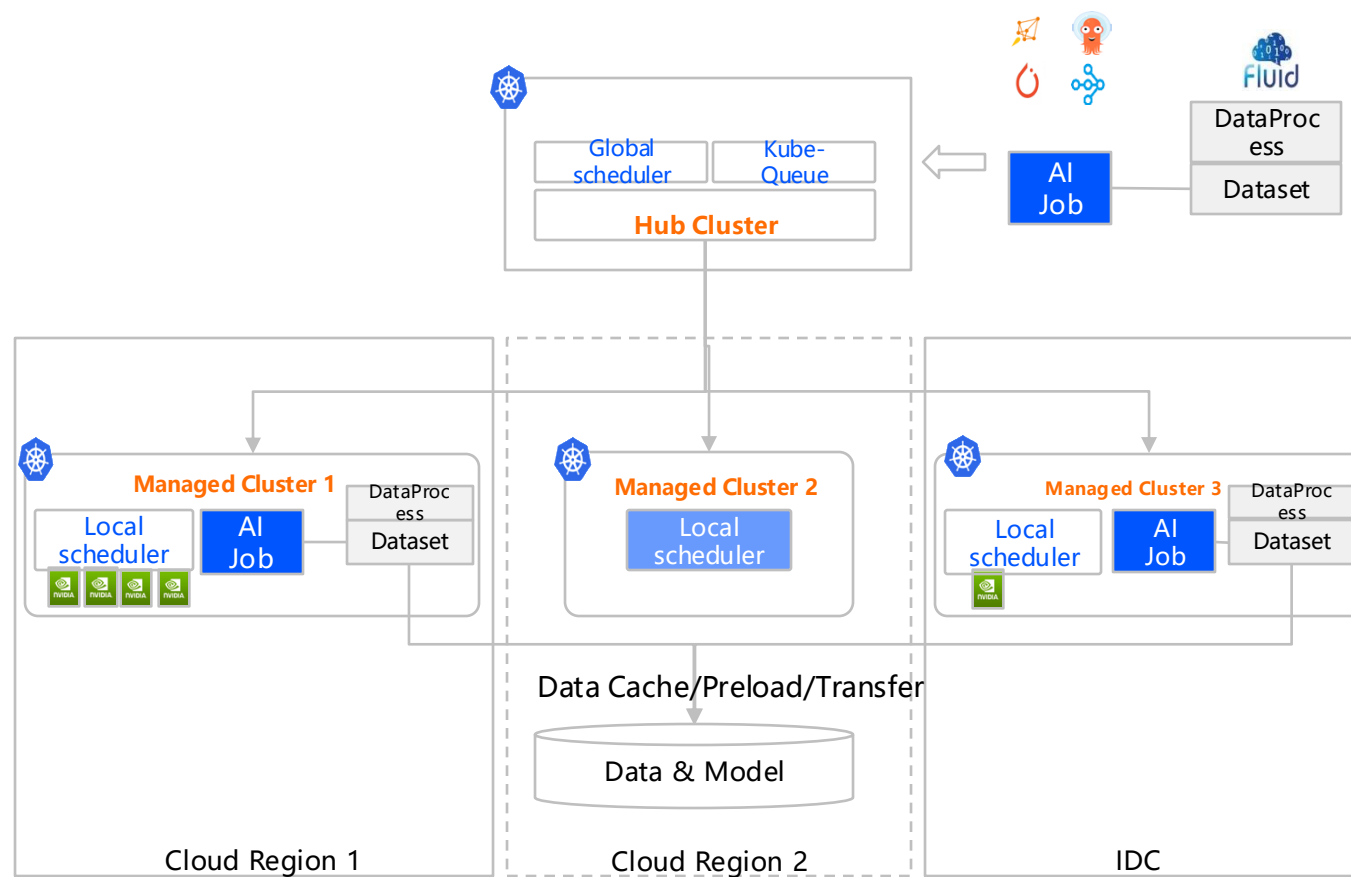
优化模型
访问性能

优化部署
管理易用性

多
Kubernetes
集群统一调度
GPU，动态调
整AI作业分布

自动预热和缓
存模型，加速
LLM推理服务
启动和伸缩

跨地域统一管
理和访问模
型，分配请求
流量，简化用
户体验



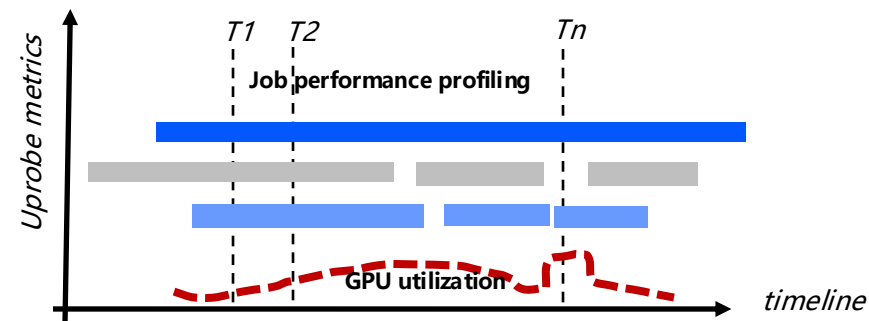
04

AI负载可观测

GPU监控与在线 Profiling

精细化GPU可观测能力

- 支持40余种DCGM监控指标，持续追踪“资源利用率 + 任务性能”
- 基于eBPF的实时GPU Profiling，动态开启，无侵入，轻量级，overhead可控小于3%
- 多级指标关联，综合分析host进程、系统调用、CUDA Kernel、NCCL通信、Pytorch算子耗时，快速定位AI任务性能瓶颈

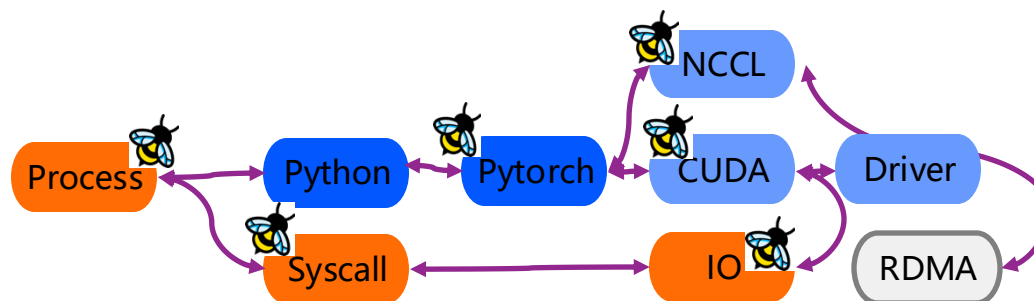
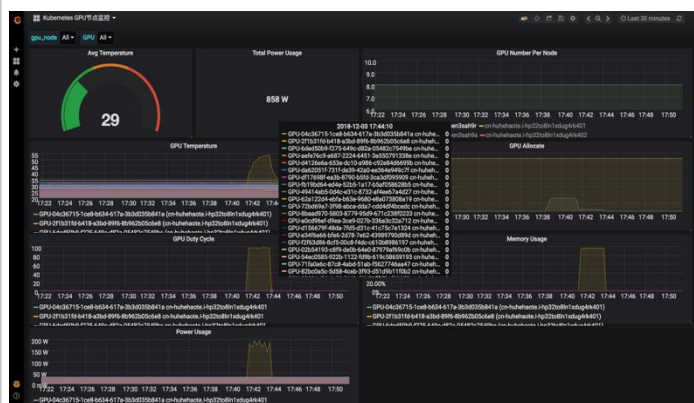


AI任务性能诊断效率提升20%

多视角统一监控：
节点，GPU设备，容器，应用

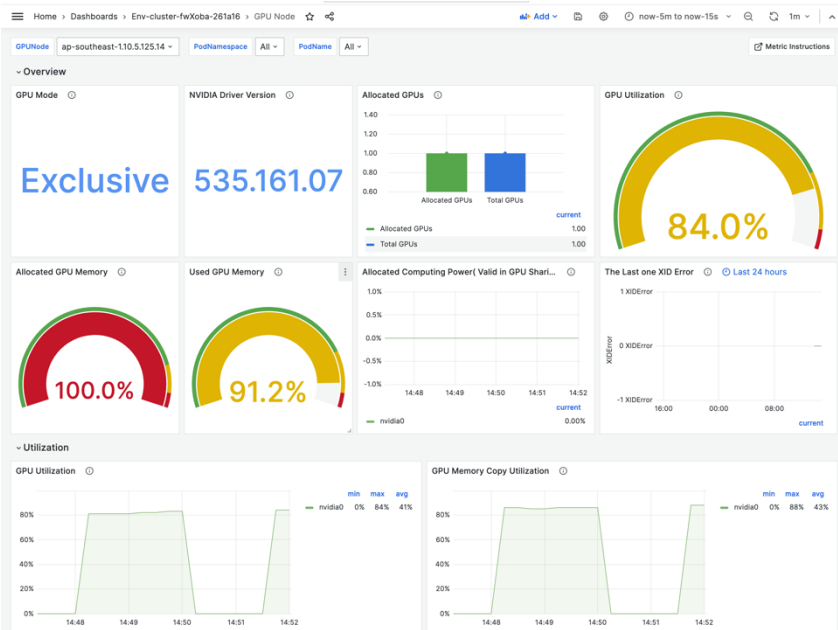
指标体系集成：
DCGM，AMPerf

内置NPD：
自动检测和告警设备异常

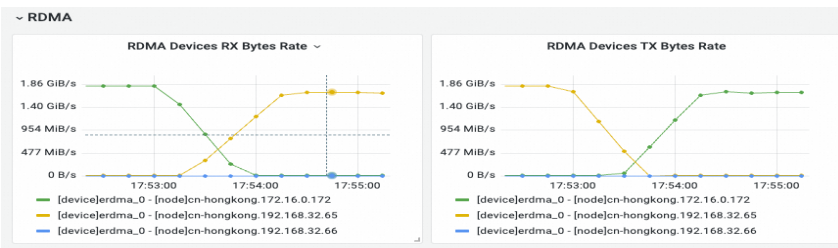


多维AI资源和任务监控大盘

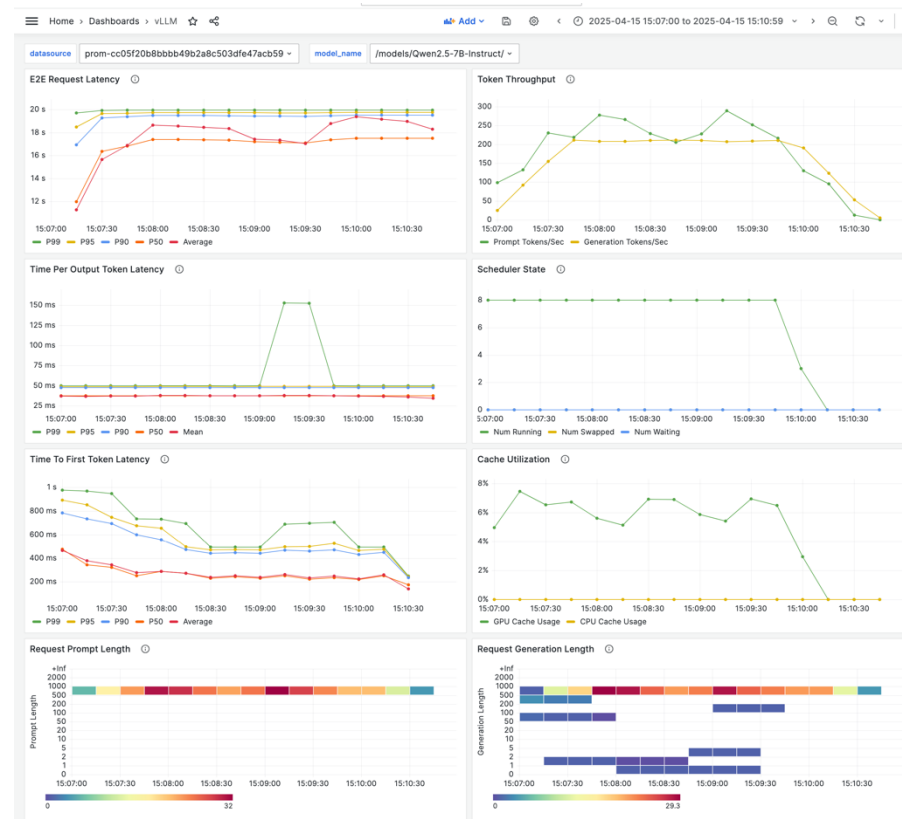
GPU metrics (Nvidia DCMG)



RDMA metrics



vLLM metrics



在线GPU Profiling, 精准定位 AI 应用性能瓶颈

无侵入

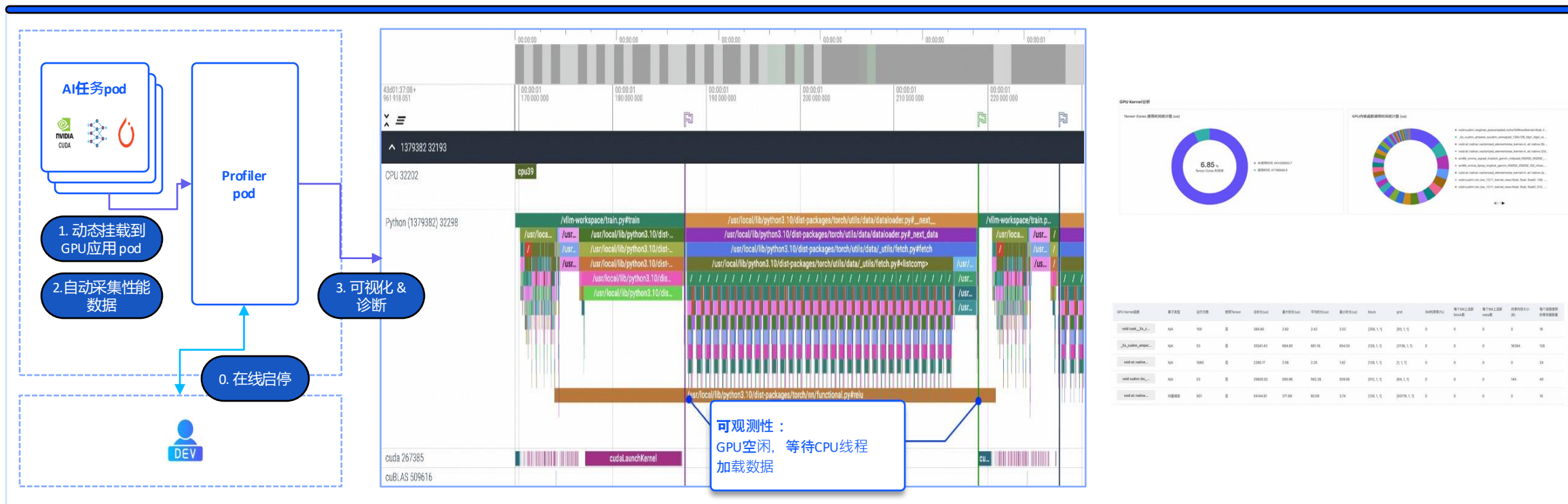
AI任务 (训练或推理) 的性能分析, 无需修改代码或中断服务

时间轴数据聚合

聚合全链路数据到一个时间轴, 包括 Python 进程、系统调用、CPU 调用、CUDA 核函数及 NCCL通信等 CUDA 库调用数据

AI应用性能分析

通过对 GPU 应用执行时间的深入洞察分析, 潜在性能瓶颈定位效率提升 50%



案例：AI Profiling快速定位显存OOM

某电商客户痛点

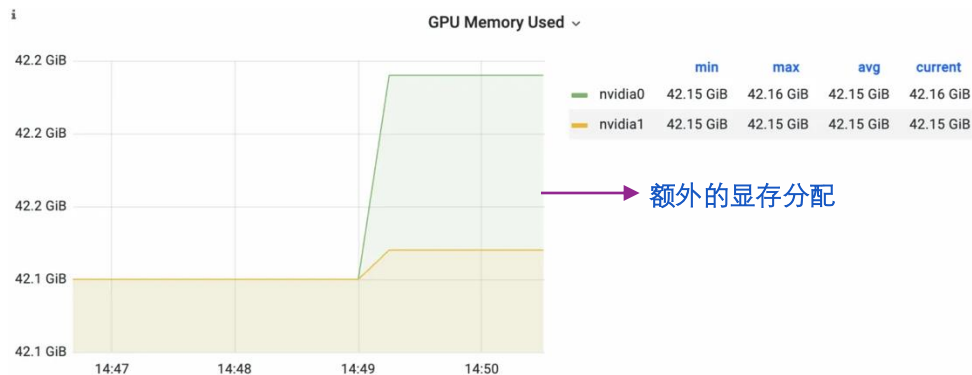
运行推理任务时出现 vLLM 服务显存 OOM 问题

问题发生后，无法在测试环境中复现，需要从线上环境大量指标中排查。用户无从下手，问题定位难度高、时间长，需要：

1. 确定是否存在异常
2. 确定分析方向，快速定位问题发生原因
3. 找出优化手段方案

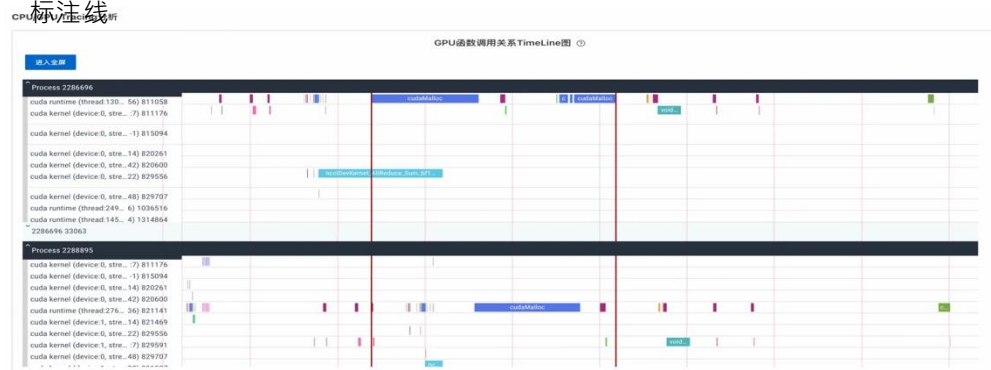
1. GPU 监控大盘：初步定位问题原因

接收到推理请求后，显存用量增长，一直未被释放。而且发现 KVCache 利用率低于 10%，排除 KVCache 占用，判断可能存在异常

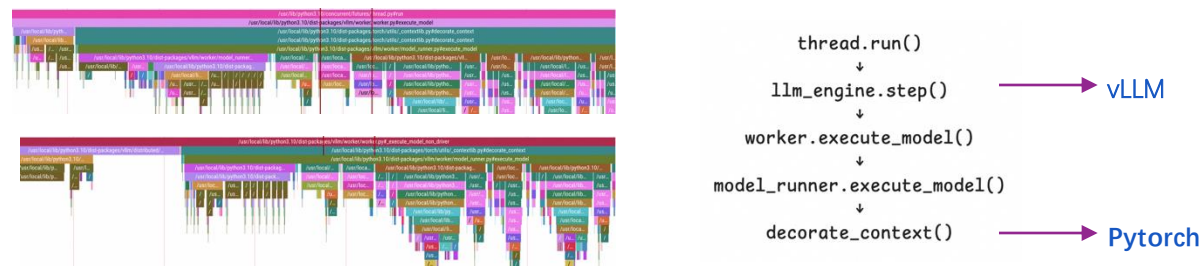


2. AI Profiling：实时采集分析推理服务数据，精细化分析根因

1) 对正在运行的推理服务发起Profiling任务，自动生成函数调用时间轴，发现 cudaMalloc 申请显存数值与监控数值一致，确定为下图cudaMalloc方法触发，对 cudaMalloc 时间段打上标注线



2) 放大标记线对应的 Python 调用，发现 GPU 显存增长过程中的方法栈调用如下图所示：



3) 确定优化方案

结合Pytorch源码分析，可以判断额外显存增长是 Pytorch 的缓存管理机制预留了显存，可以通过调整 Pytorch 环境变量CUDA_PYTORCH_CUDA_ALLOC_CONF将max_split_size_mb调大来规避。

05

异构集群稳定性

GPU故障发现、诊断与自愈

以Llama3.1 405B为例

- 16K H100, 54 days, >100M\$
- 417 unexpected interruptions, 58.7% GPU related

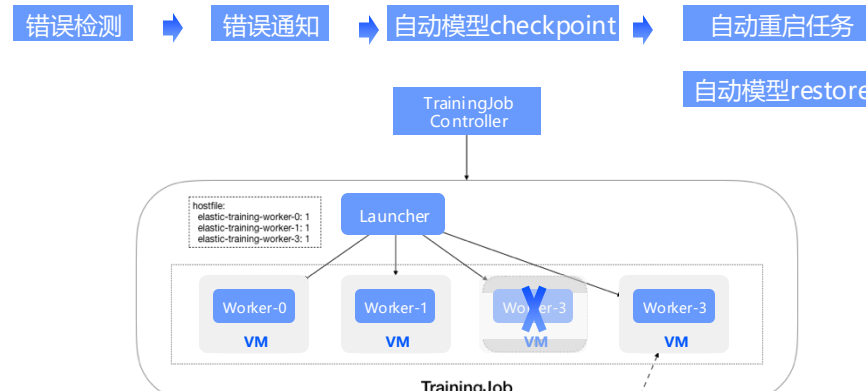
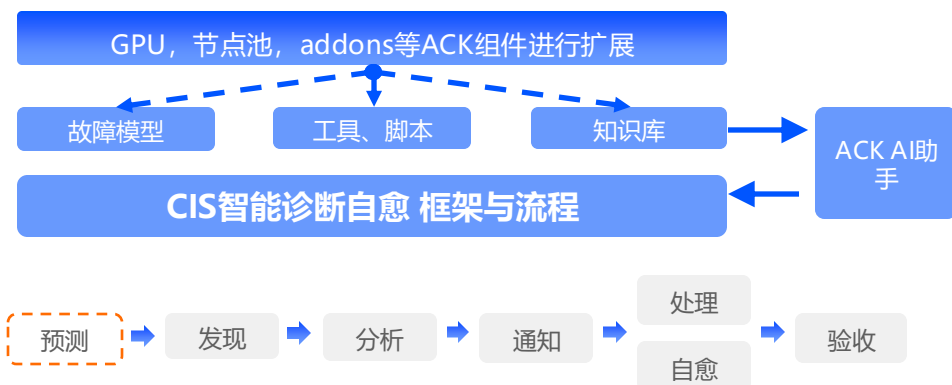
"Despite the large number of failures, significant manual intervention was required only 3 times during this period, with the rest of issues handled by automation." - 《The llama3 herd of models》

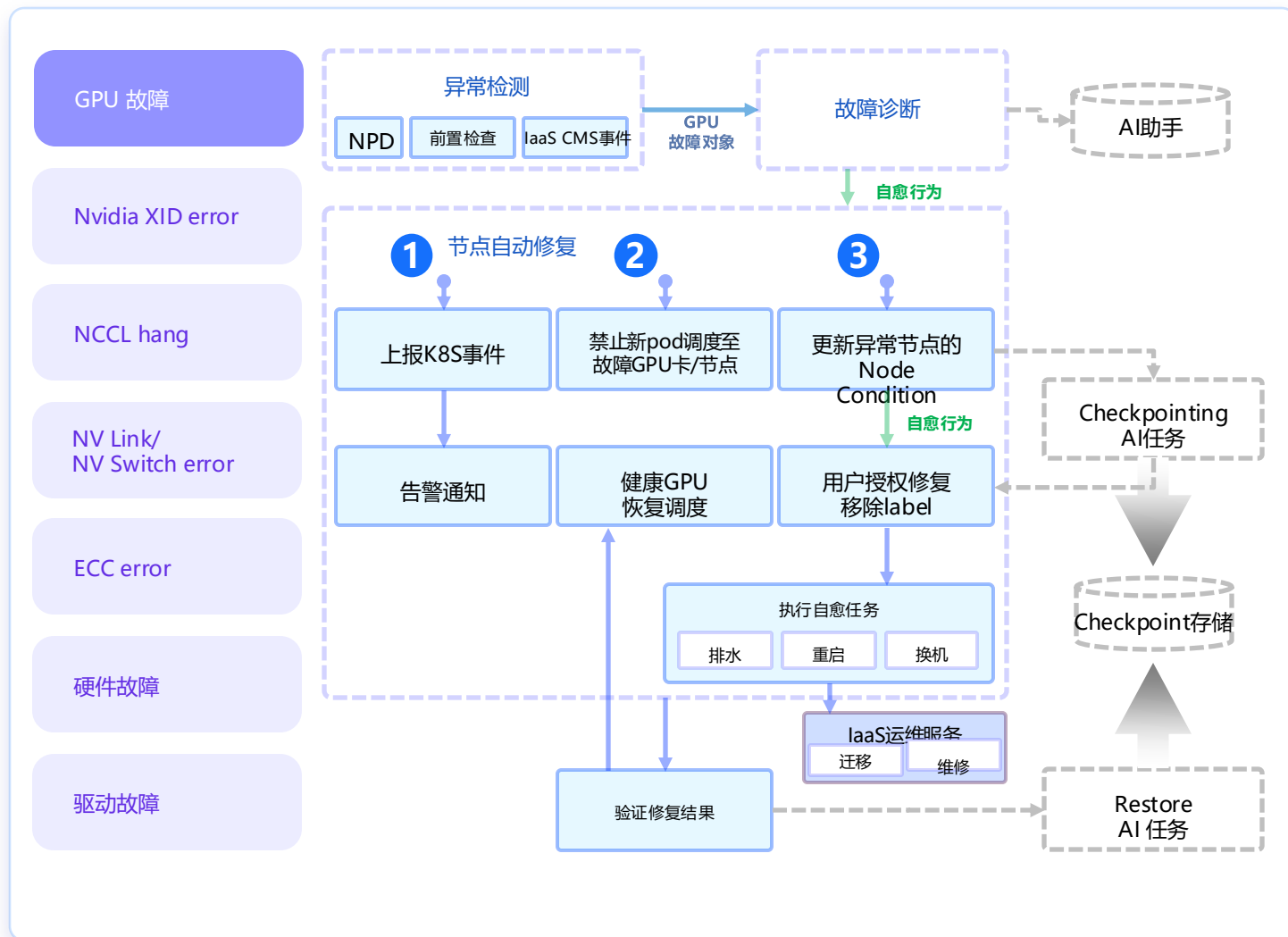
GPU资源故障处理能力

- ACK智能诊断框架，结合agent多步诊断
- 常见GPU故障（硬件、内核、驱动、ECC、NCCL）端到端处理流程
- 故障资源通知迁移，K8s调度自动隔离异常设备

GPU任务错误恢复能力

- 自动重调度AI任务
- AI训练任务弹性伸缩
- 支持多种checkpoint & restore方案





能力提升

主动GPU故障检测

主动监控并检测 GPU 故障，支持 EGS 节点及灵骏节点，如 Nvidia XID 错误、ECC 错误、驱动故障、节点及网络故障。

自动隔离 & 修复

发现异常后，立即将故障 GPU/节点从 K8s 调度中隔离，并自动触发硬件故障恢复与修复流程。支持更精细化的卡级隔离，单机正常 GPU 可继续运行任务，减少对业务影响。

用户可控的自愈流程

用户可以根据业务情况，自由选择修复方式。支持在异常发生后自动执行修复，无需人工干预即可完成全流程自愈，缩短节点异常时间；用户也可以先对异常节点上业务及数据进行处理，再授权自动修复。

典型场景

业务痛点

GPU 掉卡，训练任务突然中断/推理服务响应失败

解决方案

开启节点池的节点自愈，发现异常后自动隔离 GPU/节点并触发修复，缩短资源不可用时间，用户可同时对任务进行checkpointing & restore，重新调度等处理，减轻对业务影响

06 未来工作

► 完善Day 1能力，向Day 2演进

1. RBG完善LLM推理生命周期管理

- 开源共建LLM on Kubernetes的标准API，兼容社区部署方案，如AiBrix, Dynamo, LLM-d等
- 支持Role原地升级
- 支持推理任务拓扑和资源拓扑感知的调度

2. 稳定性系统演进

- 从Reactive 故障处理，向Proactive 异常探测，和Predictive 故障预测
- 任务异常发现与自愈，与资源故障自愈流程融合

3. LLM推理Benchmark系统

- 易于复现和评测，兼容主流模型、数据集、引擎和优化架构

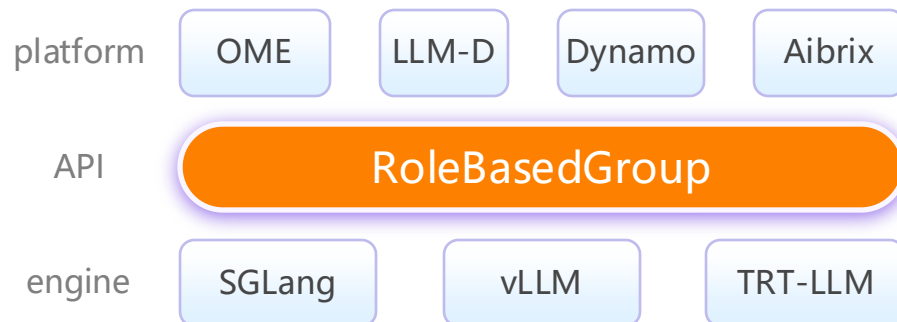
4. Agentic AI基础设施

- 基于Serverless容器技术，打造安全、快速、按需启停、极致弹性的沙箱环境支持Agent生成代码的执行，和Tool using。



▶ RBG 开源贡献 SGLang 社区

RBG is open source now



<https://github.com/sgl-project/rbg>

共建 K8s 管理大模型推理负载的标准 API



Dynamo



科技生态圈峰会 + 深度研习

——1000+ 技术团队的选择



NiDD 峰会

NiDD 峰会 上海站

AI+研发数字峰会

时间: 2026.05.22-23

NiDD 峰会 北京站

AI+研发数字峰会

时间: 2026.08.21-22

NiDD 峰会 深圳站

AI+研发数字峰会

时间: 2026.11.20-21



AiDD峰会详情

NiDD × K AI+产品峰会

NiDD × K 上海站

AI+产品创新峰会

时间: 2026.01.16-17

NiDD × K 北京站

AI+产品创新峰会

时间: 2026.08.23-24



产品峰会详情



2026 AI+ PRODUCT INNOVATION SUMMIT

01.16-17 · ShangHai

AI+产品创新峰会



Track 1: AI 产品战略与创新设计

从0到1的AI原生产品构建

论坛1: AI时代的用户洞察与需求发现

论坛2: AI原生产品战略与商业模式重构

论坛3: Agentic AI产品创新与交互设计

2-hour Speech: 回归本质



用户洞察的第一性

——2小时思维与方法论工作坊

在数字爆炸、AI迅速发展的时代，
仍然考验“看见”的“同理心”



Track 2: AI 产品开发与工程实践

从1到10的工程化落地实践

论坛1: 面向Agent智能体的产品开发

论坛2: 具身智能与AI硬件产品

论坛3: AI产品出海与本地化开发

Panel 1: 出海前瞻



“出海避坑地图”圆桌对话

——不止于翻译:

AI时代的出海新范式



Track 3: AI 产品运营与智能演化

从10到100的AI产品运营

论坛1: AI赋能产品运营与增长黑客

论坛2: AI产品的数据飞轮与智能演化

论坛3: 行业爆款AI产品案例拆解

Panel 2: 失败复盘



为什么很多AI产品“叫好不叫座”?

——从伪需求到真价值:

AI产品商业化落地的关键挑战

智能重构产品 数据驱动增长
Reinventing Products with Intelligence, Driven by Data

80+

业界大咖

汇聚产品大咖的真知灼见

40+

创新案例

开拓全新AI+产品视角

10+

分论坛

涵盖产品到商业增长的全链路

800+

听众规模

共同探索产品的智能重构



扫码查看会议详情



第8届 AI+ 研发数字峰会
AI+ Development Digital Summit

感谢聆听!

扫码领取会议PPT资料

