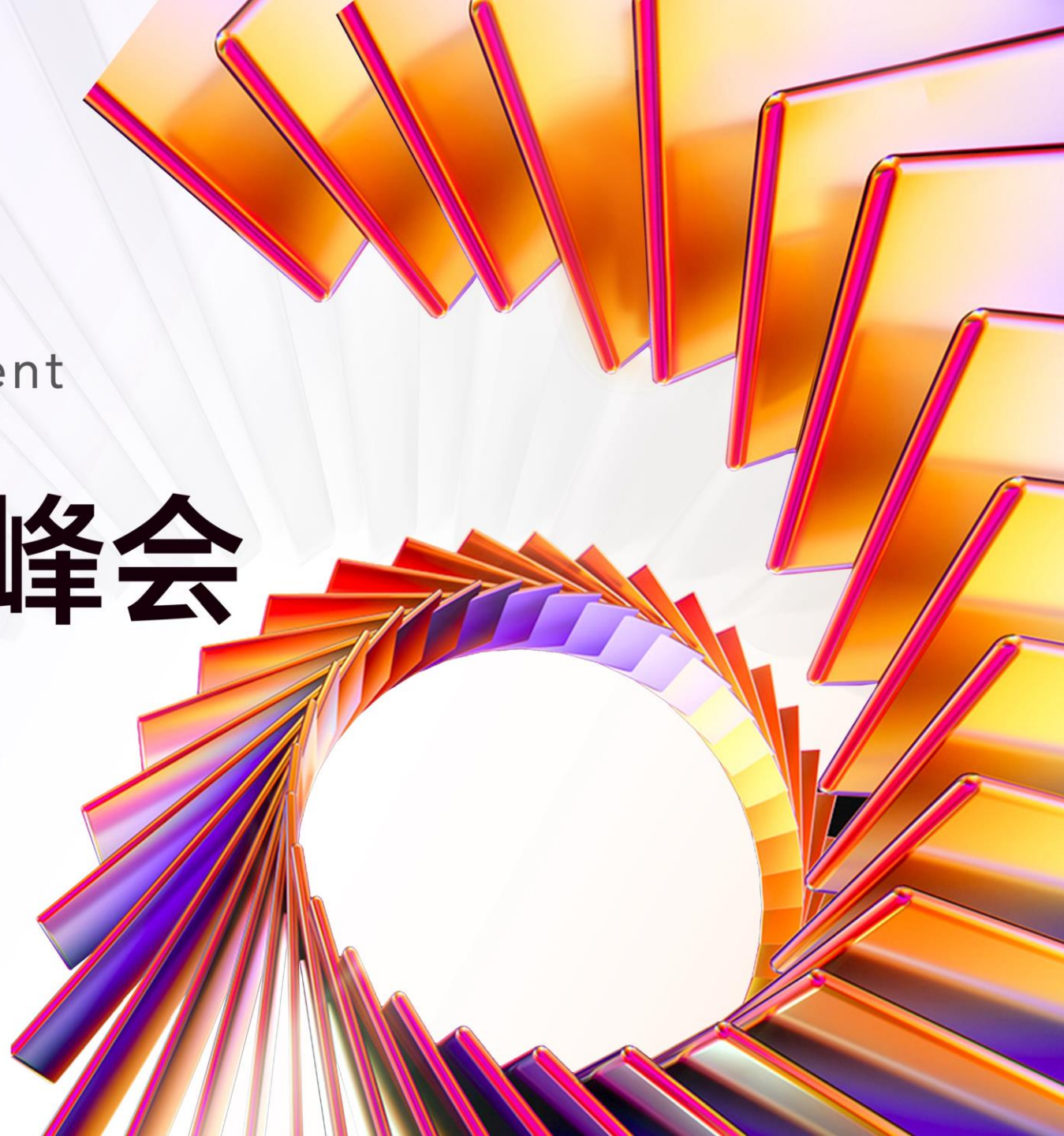
AiDD 7th 2025

第7届 AI+ Development Digital Summit AI+研发数字峰会

拥抱AI 重塑研发

8月8-9日 | 北京站



AiDD 8th 2025

第8届AI+研发数字峰会

拥抱AI 重塑研发 AI+ Development Digital Summit

查看会议详情

下一站预告 11/14-15 | 深圳站 12/19-20 | 上海站

深圳站论坛设置

智能装备与机器人

超越“编程 Copilot”

下一代知识工程

智能网联与汽车智能化

AI 测试工具开发与应用

AI 基础设施和运维

数据智能及其行业应用

可信 AI 安全工程

大模型和 AI 应用评测

多 Agent 协同框架

从智能测试到自主测试

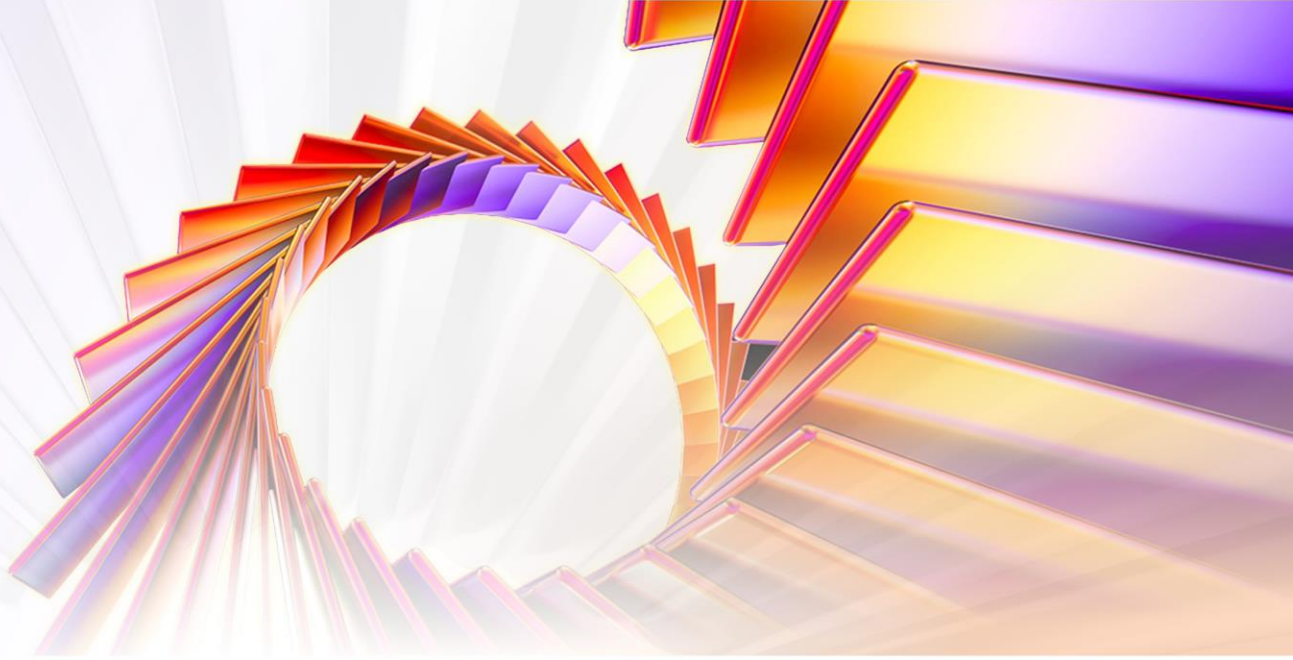
大模型推理优化

多模态 LLM 训练与应用

智能化 DevOps 流水线

上下文工程

AiDD 7th 2025 | 8月8-9日 | 北京站
第7届 AI+ Development Digital Summit
AI+研发数字峰会
拥抱AI 重塑研发



云原生平台下大语言模型PD分离架构的规模化挑战与实践

顾静 | 阿里云

顾静

阿里云 高级工程师

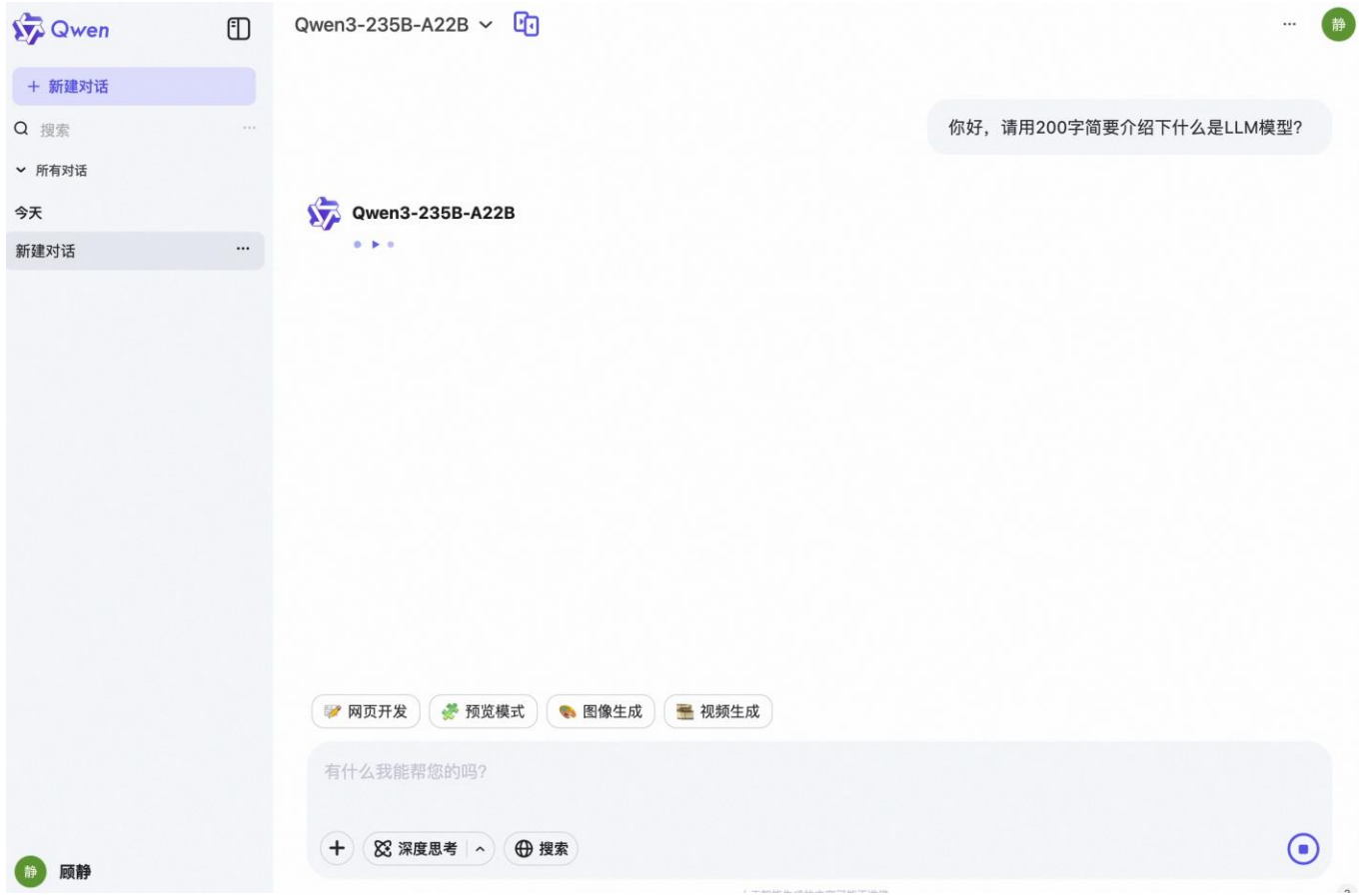
专注于云原生AI工程化领域，致力于攻克大语言模型（LLM）在Kubernetes平台上的大规模部署、性能优化与问题诊断等核心挑战。主导设计并实现了ACK推理套件。该项目旨在为LLM提供一套完整的云原生解决方案，其核心能力包括：支持部署多种形态的LLM推理服务（包括PD分离架构）、基于KVAware的智能路由、以及智能扩缩容等。此外，该套件还集成了全面的可观测性与在线AI Profiling功能，旨在构建一个稳定、高效且成本可控的LLM服务平台。作为一名开源社区的积极贡献者，是 Kubernetes 、 KServe 、 LWS 、 vLLM 及 Dynamo 等多个项目的 Contributor，并深度参与了RBG与Kubeflow社区的核心建设。曾多次在KubeCon等全球技术峰会上分享相关实践经验。

目录
CONTENTS

PART 01

大语言模型PD分离架构介绍

大语言模型（LLM）基本原理



Step 1:

Input: recite the first law

Output: A

Step 2:

Input: recite the first law A

Output: robot

Step 3:

Input: recite the first law A robot

Output: may

图片来源: https://jalammar.github.io/illustrated-gpt2/

Transformer架构

Figure 1: The Transformer - model architecture.

- ✓ 支持无监督预训练，擅长根据用户输入创造性的生成文本。
- ✓ 适用于文本生成、智能助手、代码补全等任务
- ✓ 代表模型：GPT、LLama、Qwen

Decoder Only Models

- ✓ 能够理解复杂输入并生成相关输出。
- ✓ 适用于机器翻译任务
- ✓ 代表模型：T5

Encoder-Decoder Models

- ✓ 擅长理解文本语义，但无法直接生成文本输出。
- ✓ 适用于文本分类、情感分析等任务
- ✓ 代表模型：BERT

Encoder Only Models

图片来源: https://jalammar.github.io/illustrated-gpt2/

Prefill & Decode

Prefill (预填充) – 理解问题阶段

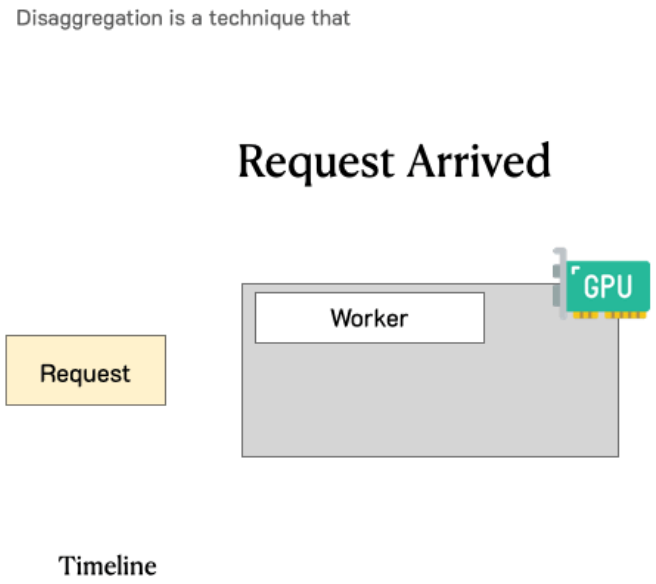
从用户输出的Prompt到生成第一个Token的过程

- 【输入】用户的整个Prompt
- 【处理过程】模型会**并行地**处理Prompt中所有Token。它会根据这些Token计算并缓存KV Cache。
- 【特点】计算密集型；一次性

Decode (解码) – 逐字回答阶段

生成第二个Token到最后一个Token的过程

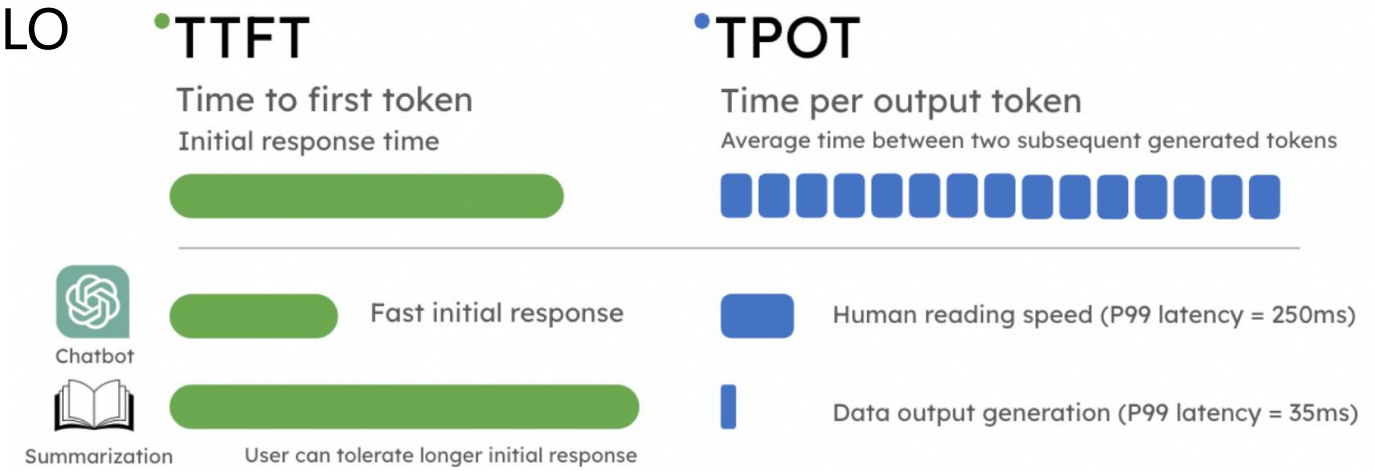
- 【输入】上一步生成的**单个**Token，及KV Cache
- 【处理过程】模型只处理这一个新Token，利用存储的KV Cache计算Attention，输出下一个Token。
- 【特点】显存密集型；迭代性



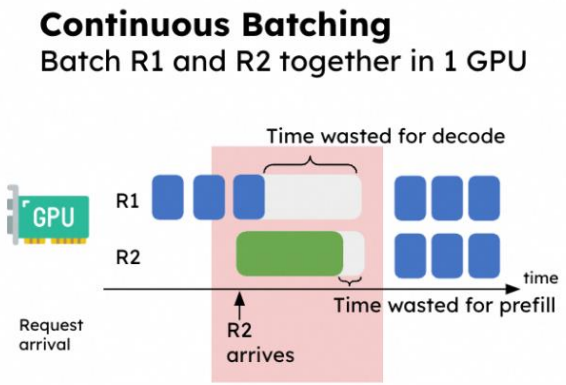
https://hao-ai-lab.github.io/blogs/distserve/

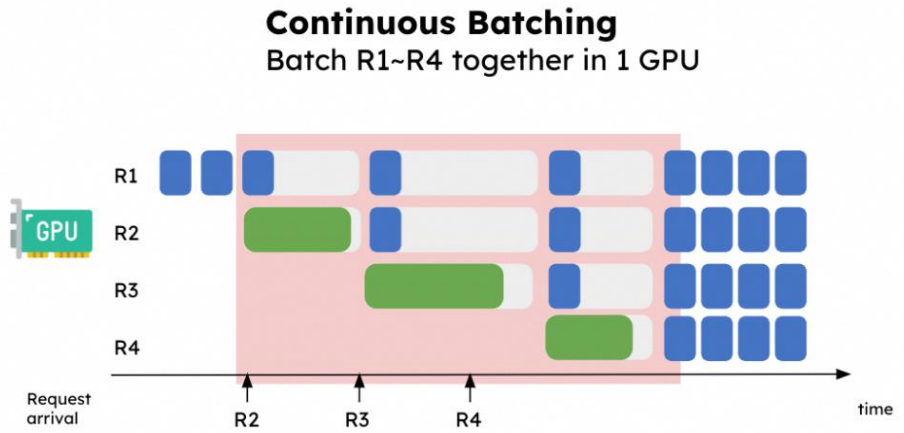
Prefill & Decode

- Prefill 和 Decode 具有不同的SLO



- Continuous Batching机制会导致PD相互影响





为什么要将Prefill和Decode分开部署

资源优化

Prefill需要强大的计算能力，Decode需要高效的内存访问速度。分离后可以将两个阶段部署到不同的硬件上。

可伸缩性

分离后， Prefill可以根据用户请求等指标扩容，Decode可以根据正在进行的对话数量及TPOT指标分别扩容。

吞吐与延迟平衡

Prefill和Decode部署在同一个设备上会相互影响，分离部署后可以降低Decode阶段延迟。

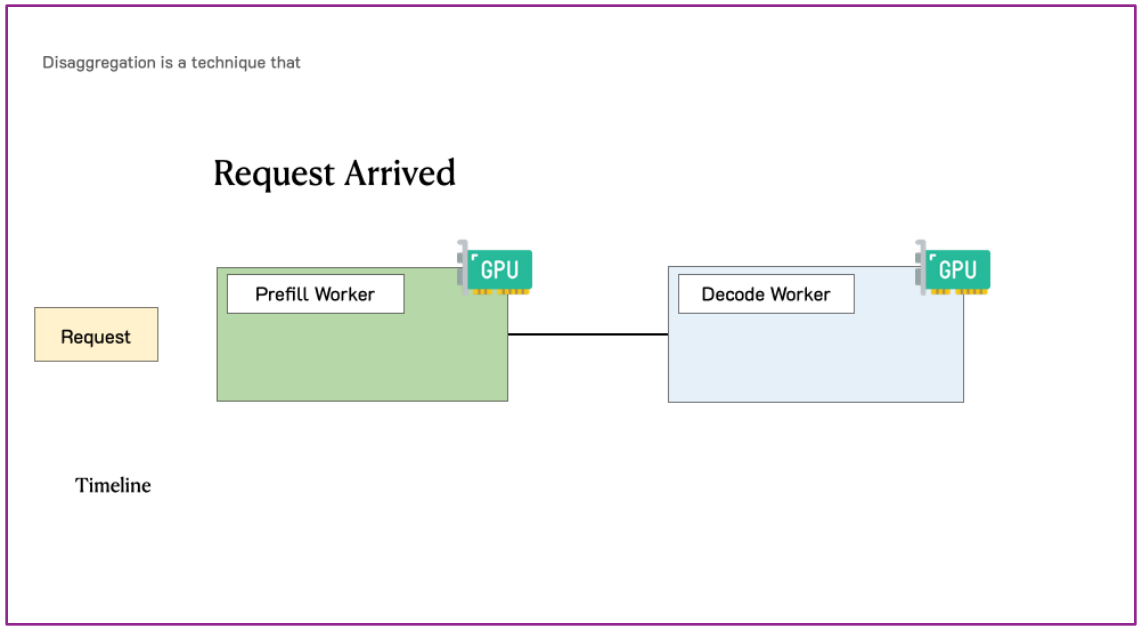
稳定性

分离部署后，两个阶段不再相互影响，极大地减少了长尾问题，提升两阶段的SLO。

PD分离部署

PD不分离

VS.



PD分离

Migrate: 将Prefill阶段计算的KV Cache传输给Decode

https://medium.com/@joaolages/kv-caching-explained-276520203249

Transformer Attention机制

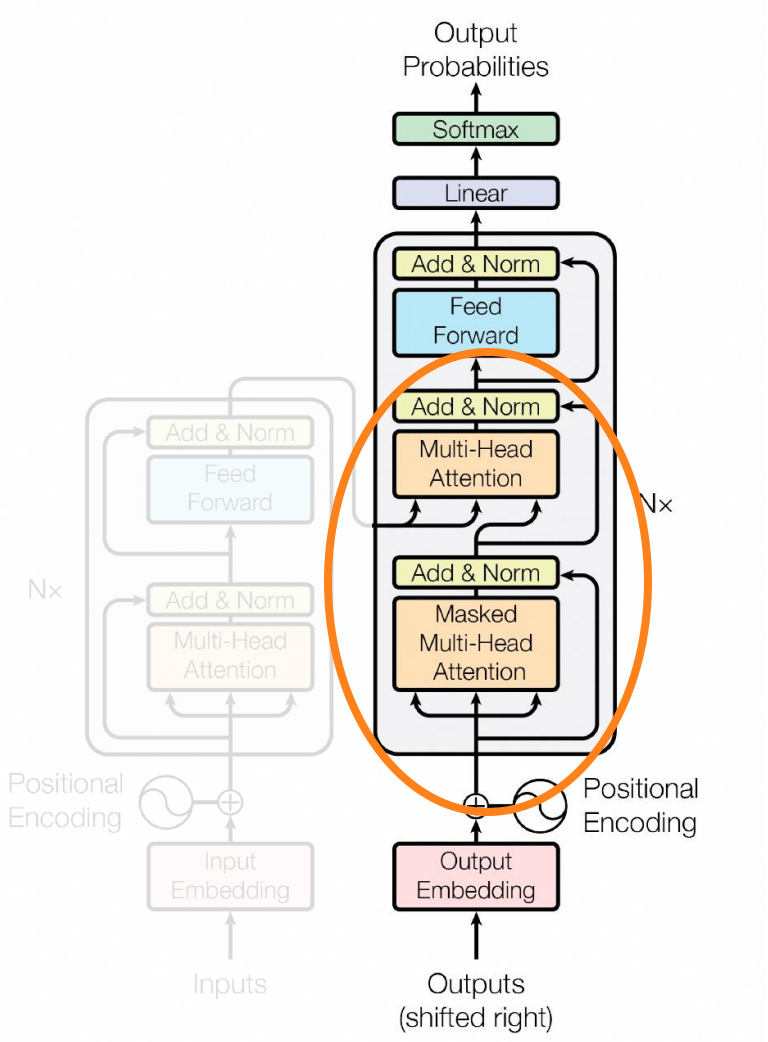
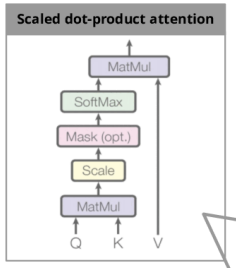


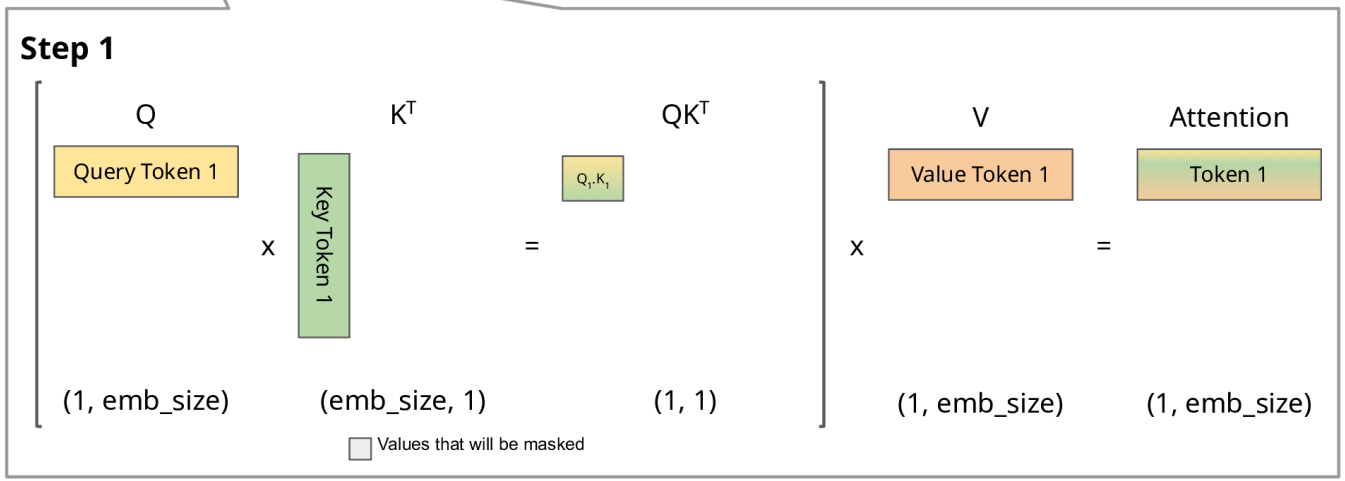
Figure 1: The Transformer - model architecture.

- $self\_attention(x) = \text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) \times V$

在Attention计算中，每个Token都会与序列中的所有前序Token（包括它自己）进行交互，以计算其上下文表示。



Step 1: self_attention(hi)
Step 2: self_attention(hi what)
Step 3: self_attention(hi what can)
Step 4: self_attention(hi what can i)



Zoom-in! (simplified without Scale and Softmax)

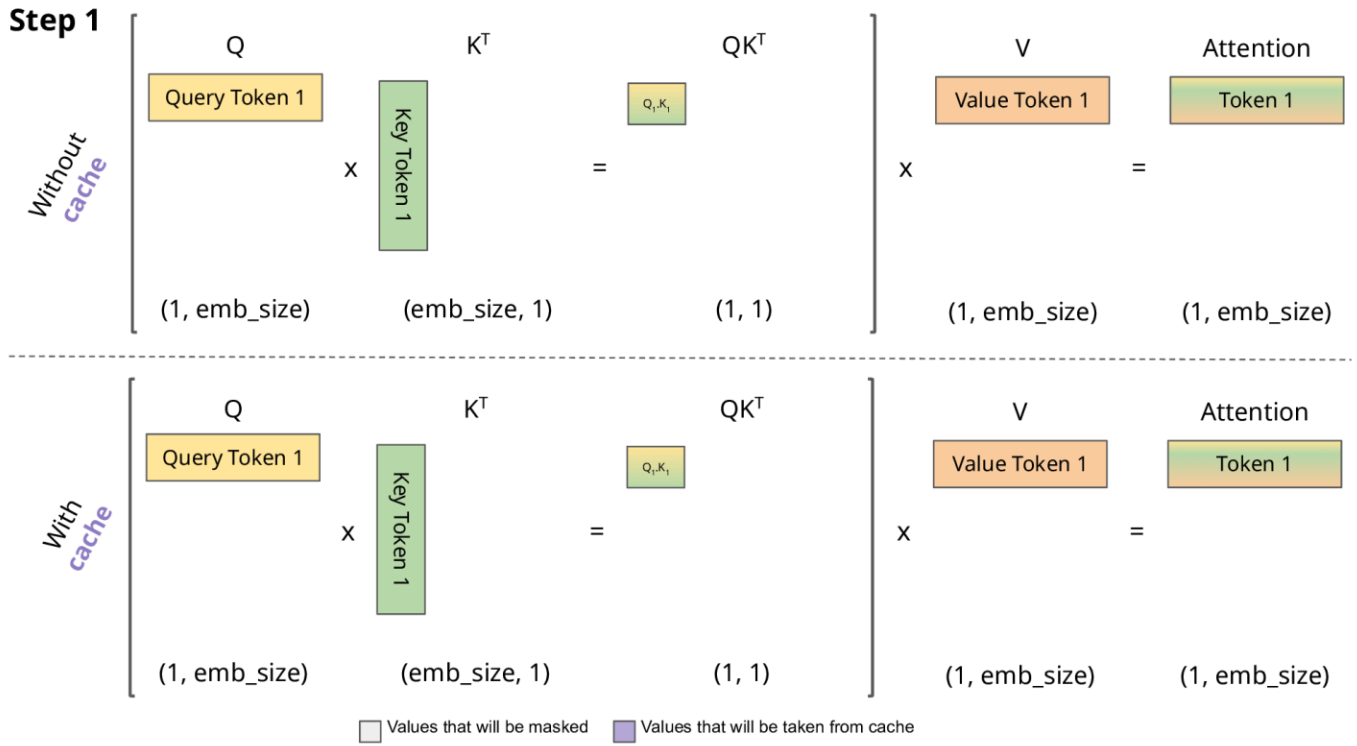
KVCache

➢ KVCahce

一个Token的K和V矩阵被计算出来，就会被存储到GPU显存中。当模型生成下一个Token时，只需要计算新Token自己的Q，K，V即可。

➢ KVCahce的必要性

- 显著提升推理速度
- 节省计算资源
- 支持更长的上下文

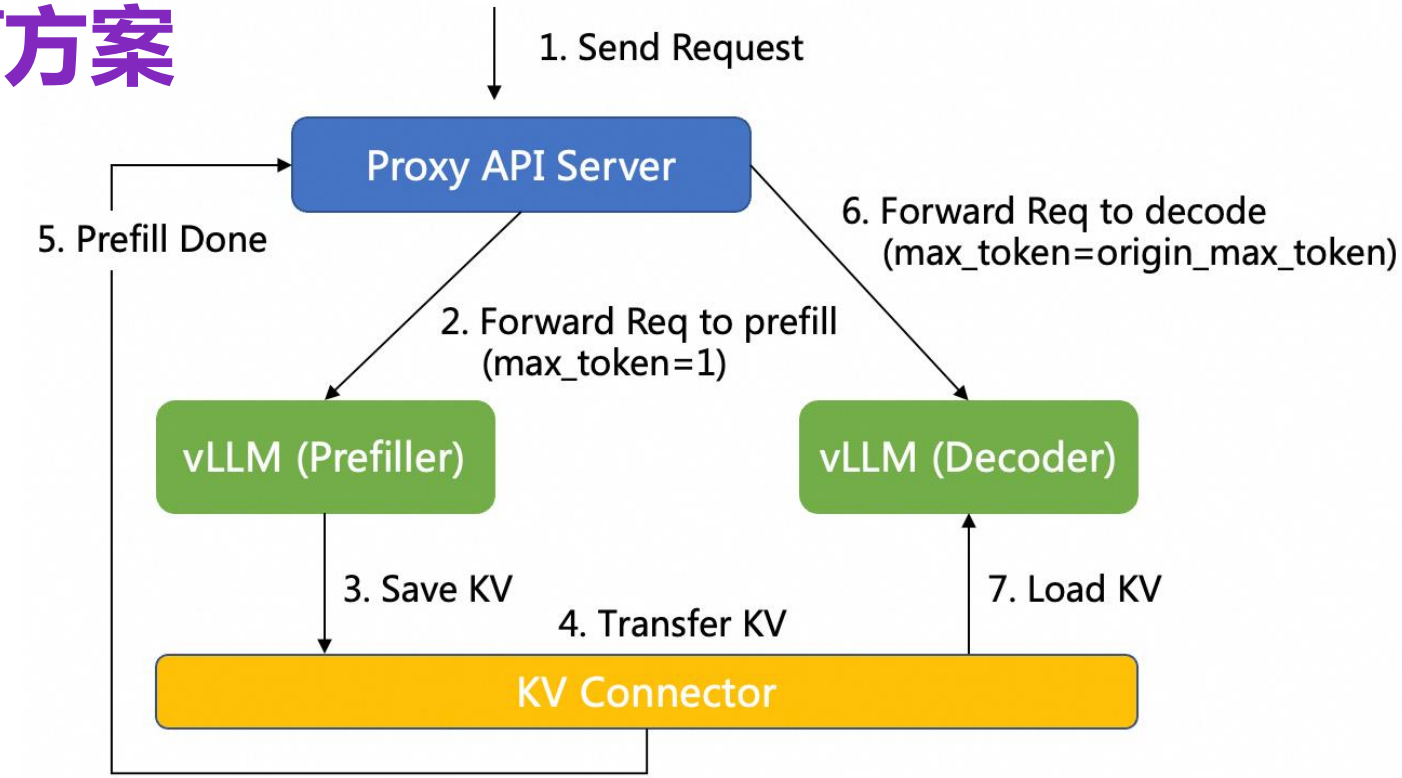


https://medium.com/@joaolages/kv-caching-explained-276520203249

PART 02

PD分离架构实现方案

vLLM PD分离方案



vLLM启动参数增加**--kv-transfer-config**, 用于指定kv connector配置信息, 如kv_role等。

kv_role: kv_producer | kv_comsumer

```
# prefilling instance, which is the KV producer
CUDA_VISIBLE_DEVICES=0 vllm serve $MODEL_NAME \
    --port 8100 \
    --max-model-len 100 \
    --gpu-memory-utilization 0.8 \
    --trust-remote-code \
    --kv-transfer-config \
    {"kv_connector":"PyNcclConnector","kv_role":"kv_producer","kv_rank":0,"kv_parallel_size":2}' &
```

MoonCake + vLLM XpYd

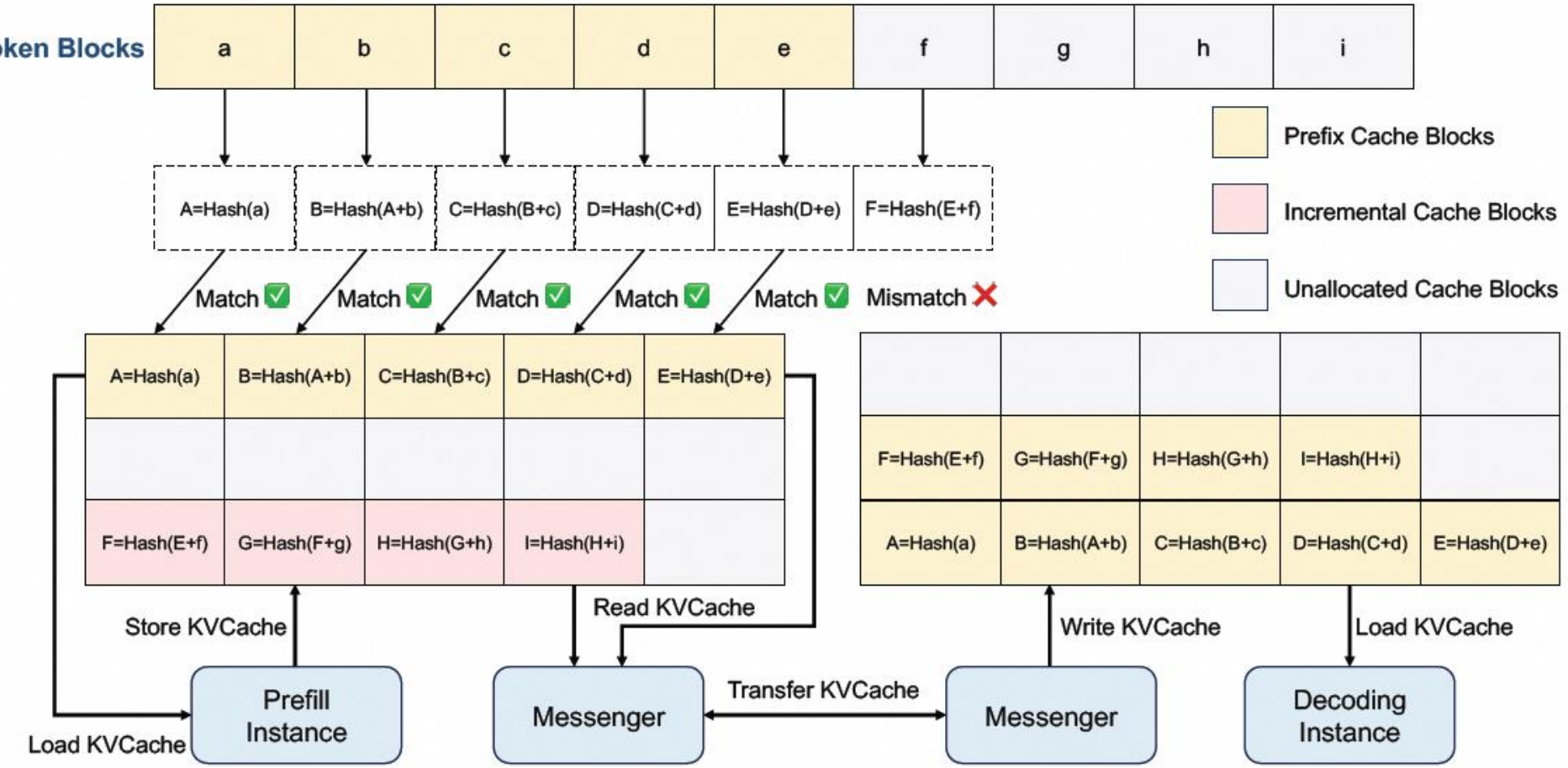
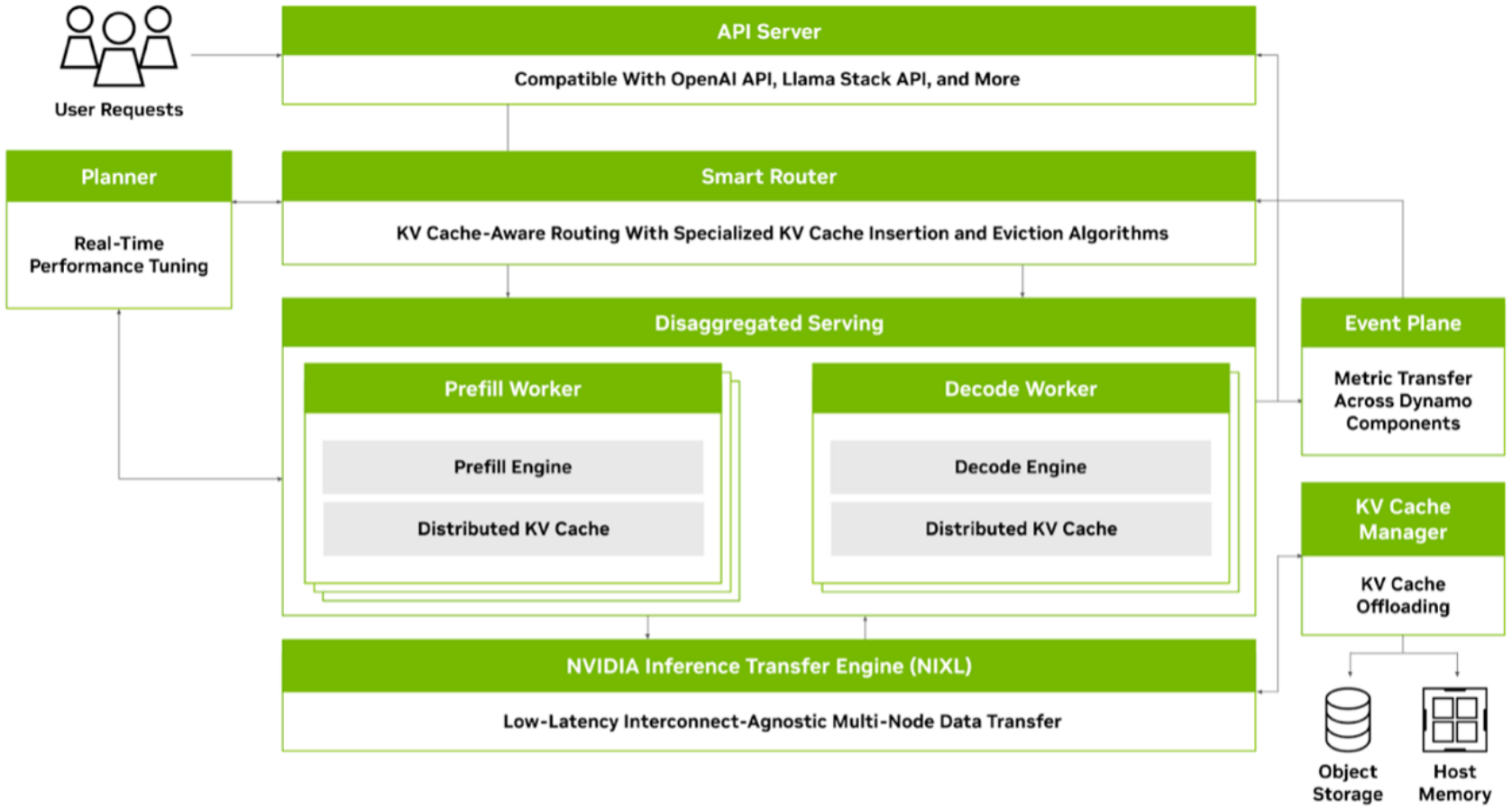


Figure 3: The KVCache pool in CPU memory. Each block is attached with a hash value determined by both its own hash and its prefix for deduplication.

NVIDA Dynamo XpYd分离方案



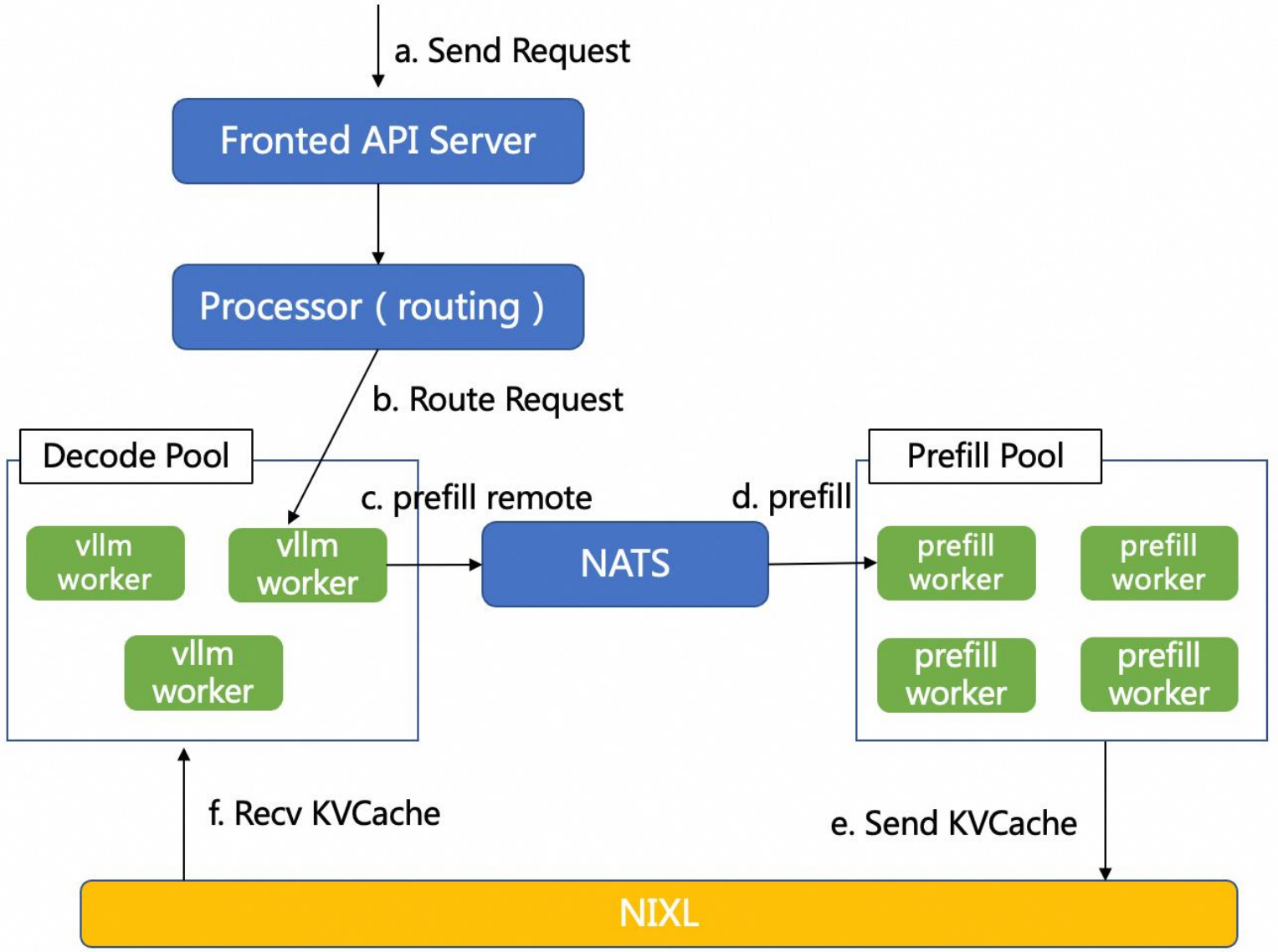
Dynamo PD分离

Fronted
请求入口

Processor
将配置参数解析为vLLM格式；根据路由策略从decode pool中选择一个合适的vllm worker。

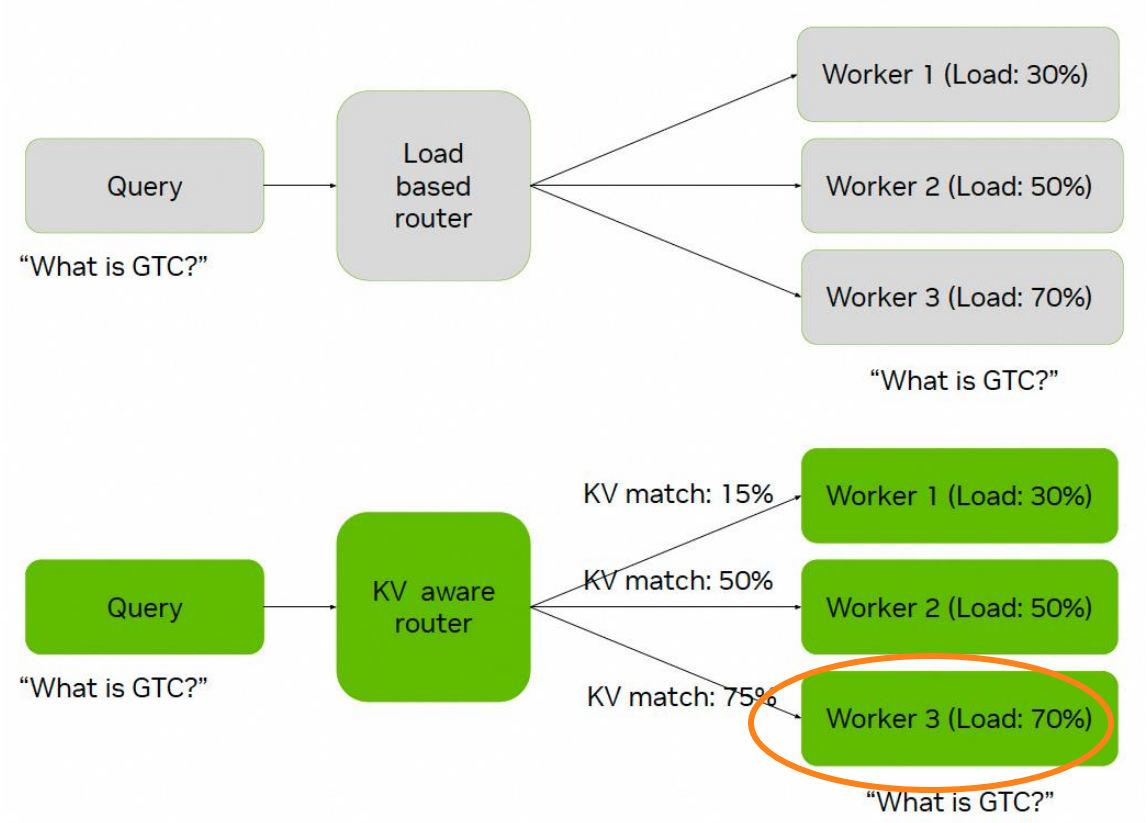
vLLM Worker
vllm实例，负责完成decode & prefill工作

Prefill Worker
vllm实例，负责完成prefill工作



Dynamo PD分离

KV Cache Aware Routing

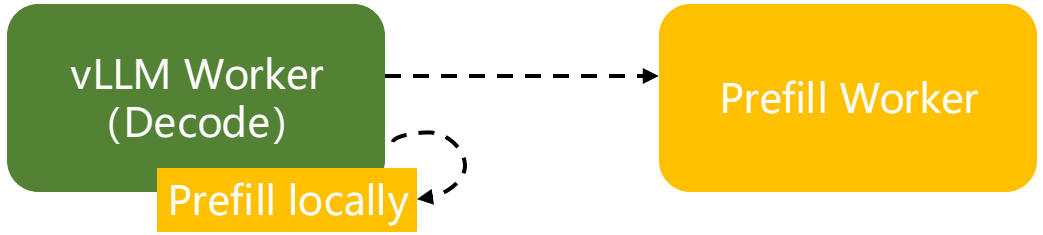


$$Cost\_Function = KV\_match\_ratio - Worker\_load$$

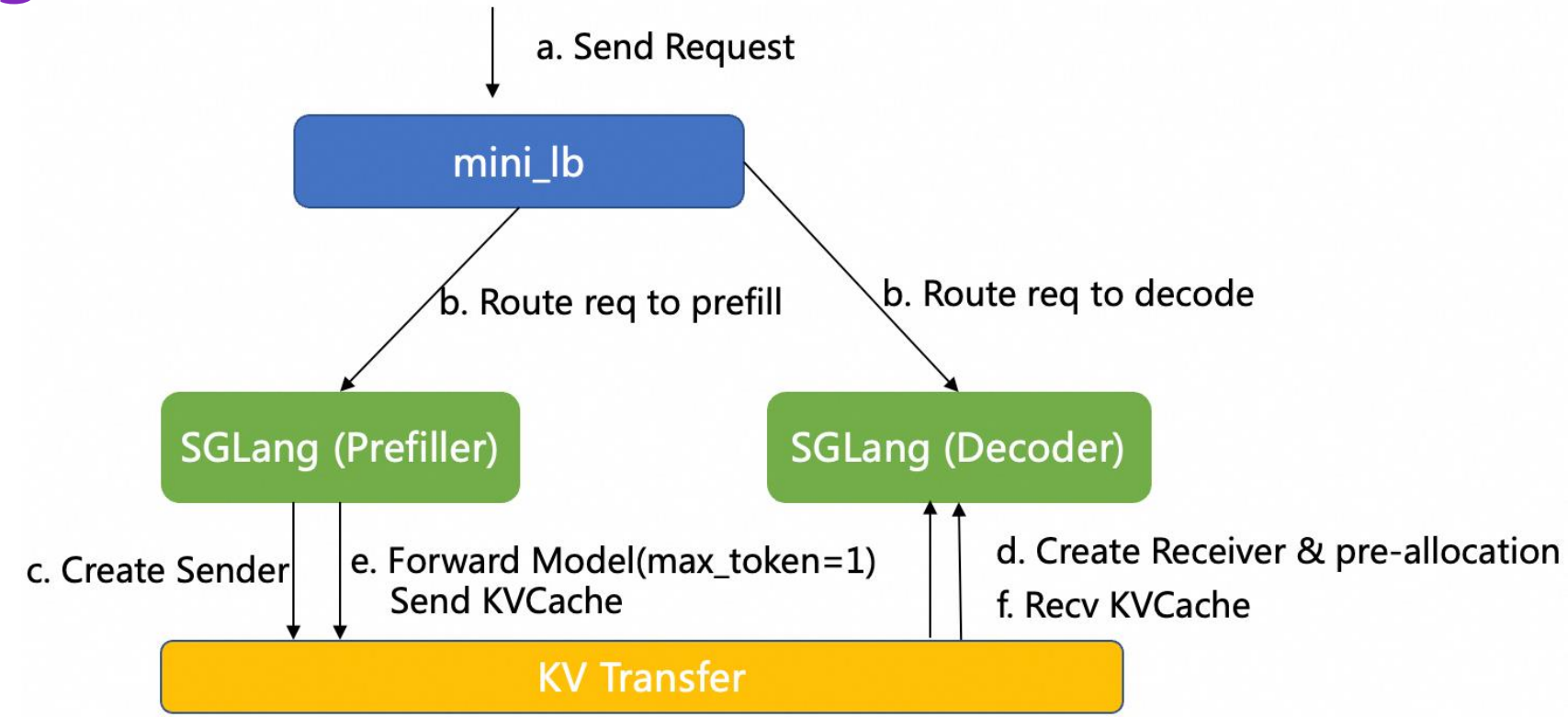
Conditional Disaggregation

1. Prompt中减去命中Prefix cache部分后，剩余的Prompt长度大于指定阈值（默认是10）

2. Prefill queue中prefill requests的数量小于阈值



SGLang PD分离



SGLang启动参数增加—disaggregation-mode参数, 用于指定PD分离配置信息。

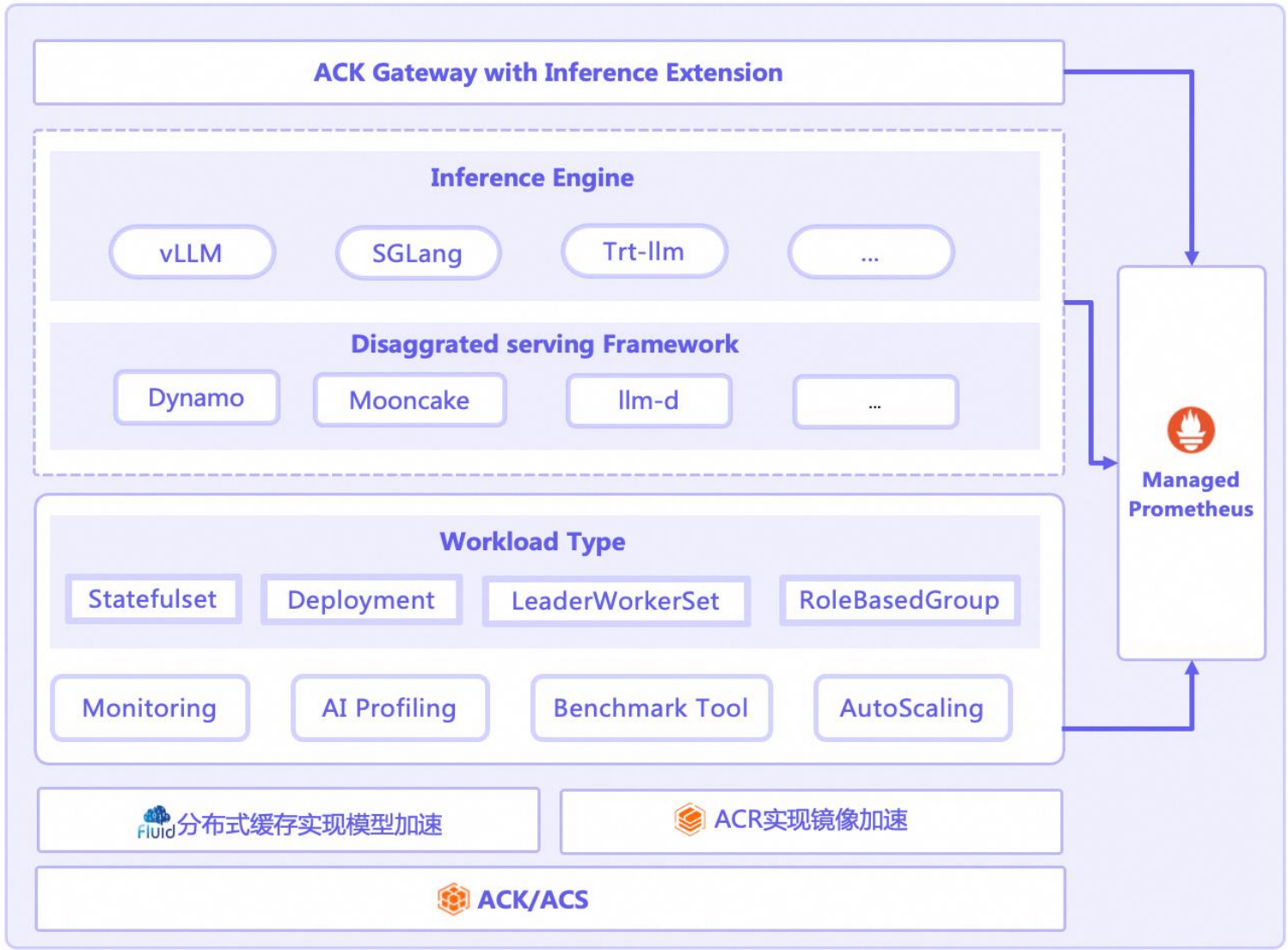
disaggregation-mode: prefill | decode

```
python3 -m sglang.launch_server
--model-path /models/Qwen2.5-7B-Instruct/
--disaggregation-mode prefill
--port 8000
--host 192.126.8.91
```

PART 03

基于推理套件实现PD分离规模化部署

ACK推理套件



推理部署容器负载

RoleBasedGroup （RBG）

- 支持部署 PD 分离架构的 LLM 应用。可以在一个工作负载中部署 Scheduler、Prefill及Decode Worker。
- 支持 、vLLM+Mooncake、SGlang+Mooncake、Dynamo 等多种 PD 分离部署方案

可观测性与自动弹性伸缩

- LLM 观测指标: TTFT, TPOT,Thoughput, Request Rate, KVCache usage, ISL, OSL
- 支持基于 TTFT、TPOT 等用户指标对 P/D 按需扩缩容
- 支持 waiting requests、KV Cache 等系统指标扩缩容

冷启动加速

- 使用Fluid构建分布式缓存及模型预加载能力，加速模型加载
- 使用ACR镜像按需加载能力，加速AI容器启动

模型感知智能路由

ACK Gateway API with Inference Extension

- 基于模型的调度策略，支持 LoRA/KVCache Aware、Waiting request num、Model Priority等策略
- 多模型版本的灰度发布能力

AiDD 7th 2025

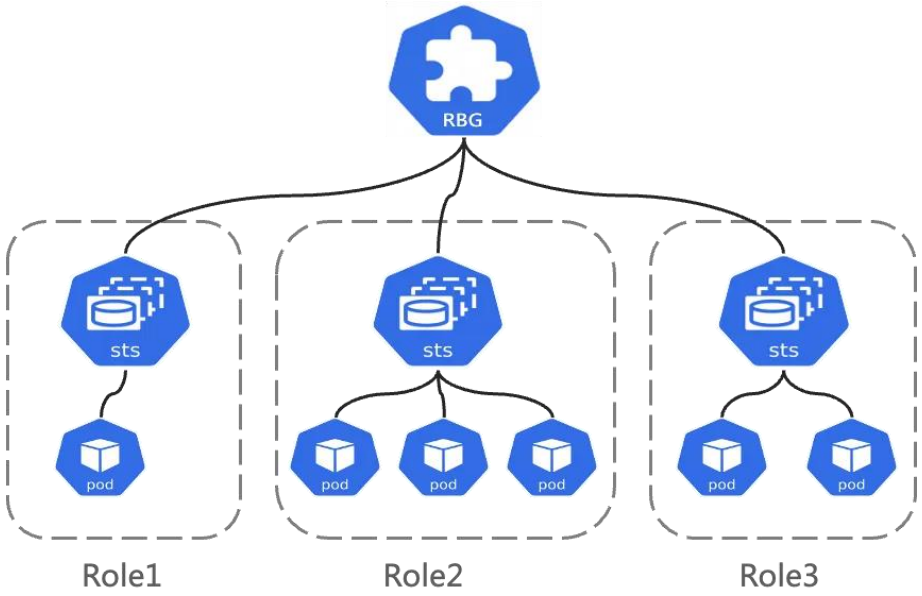
PART 3-1

推理部署负载 RoleBasedGroup

RoleBasedGroup

A workload API for deploying a group of pods as a unit of replication

- A group is a “Super Pod”: customized multiple roles
- Built on top of StatefulSet/LWS and supports most of their semantics




```yaml
apiVersion: workloads.x-k8s.io/v1alpha1
kind: RoleBasedGroup
metadata:
  name: pd-disagg
spec:
  roles:
    - name: scheduler
      replicas: 1
      template:
        spec:
          containers:
            - name: scheduler
              ...

    - name: prefill
      replicas: 3
      template:
        spec:
          containers:
            - name: vllm
              ...

    - name: decode
      replicas: 2
      template:
        spec:
          containers:
            - name: vllm
              ...
```

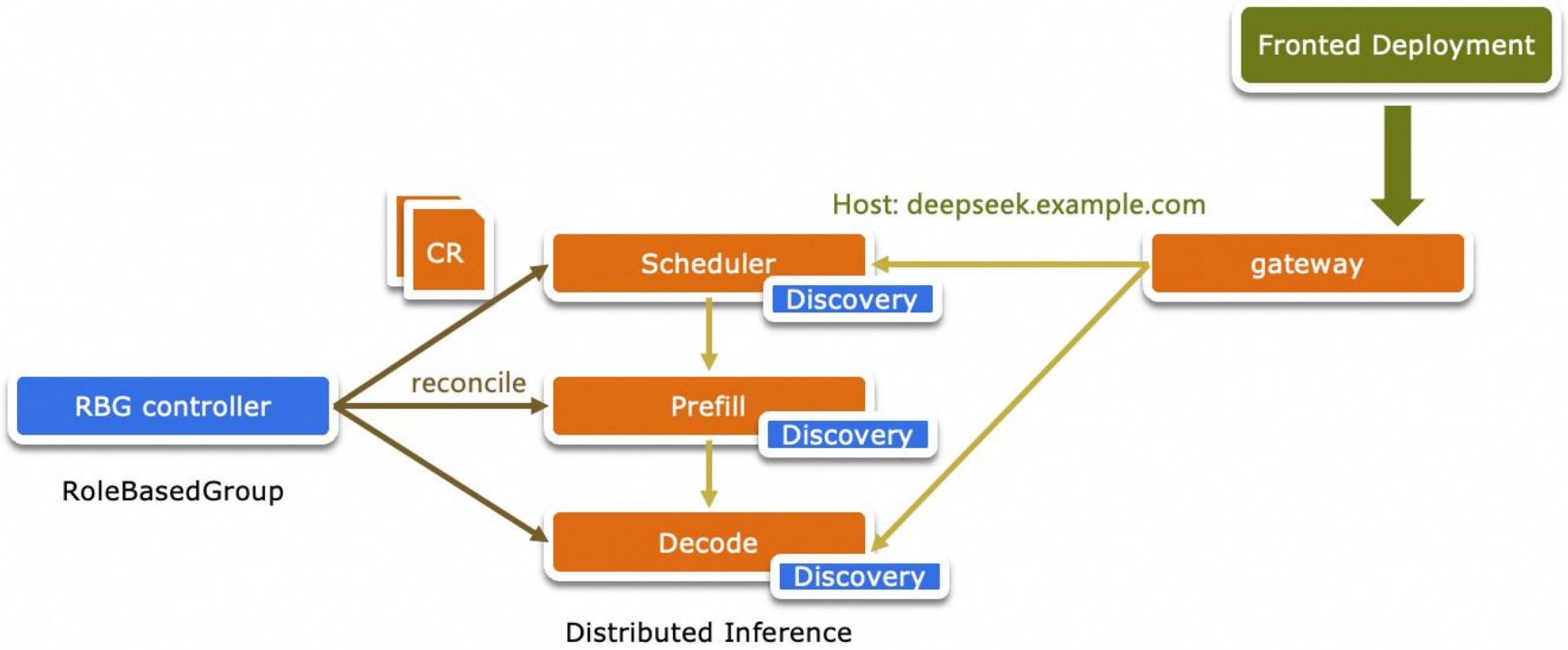
RoleBasedGroup

➢Orchestration

- 灵活的多角色定义
- 基于角色的启动控制
- 基于Role扩缩容

➢Runtime

- 自动服务发现
- 故障自愈
- 滚动更新
- Gang Scheduling



Multi-Support, Automation, Scalability

多角色定义

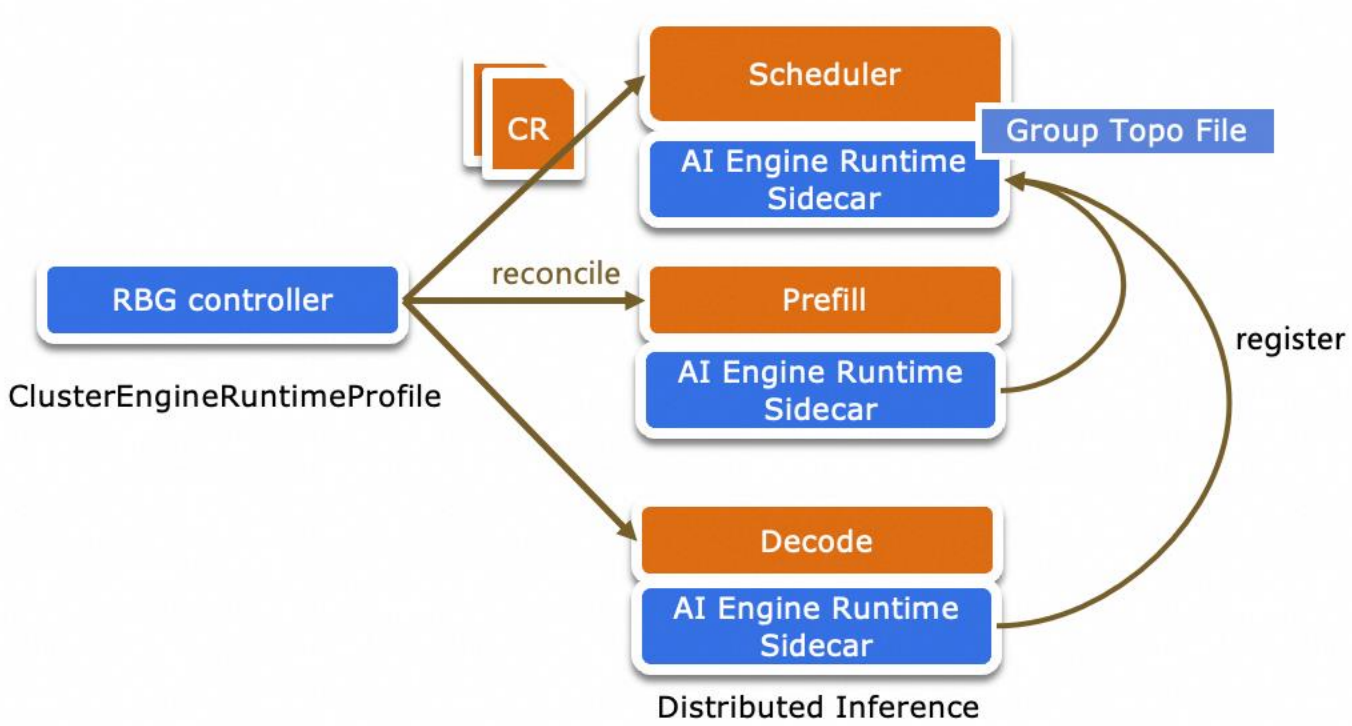
```yaml
apiVersion: workloads.x-k8s.io/v1alpha1
kind: RoleBasedGroup
metadata:
  name: dynamo-pd
spec:
  roles:
  - name: processor
    replicas: 1
    dependencies: [" prefill" , "decode" ]
    workload:
       apiVersion: apps/v1
       kind: statefulset #Also support leaderWorkerSet
    template:
       spec:
         containers:
         - name: processor
           command:
            - "dynamo serve graphs.disagg-sche-route:Frontend -f
configs/disagg-sche-route.yaml"
           env:
          - name: ETCD_ENDPOINTS
             value: http://etcd:2379
         - name: NATS_SERVER
            value: nats://nats:4222
         image: zibai-test/dynamo:v0.2.0
```

```yaml
roles:
   - name: prefill
     replicas: 2
     template:
        spec:
          containers:
          - name: vllm-worker
             command:
              - "dynamo serve graphs.disagg-prefill:PrefillWorker "
            image:  zibai-test/dynamo:v0.2.0
       engineRuntimes:
         - profileName: patio-runtime
```

```yaml
roles:
   - name: decode
     replicas: 1
     template:
        spec:
          containers:
          - name: vllm-worker
            command:
              - "dynamo serve graphs.disagg-prefill:PrefillWorker"
            image:  zibai-test/dynamo:v0.2.0
       engineRuntimes:
         - profileName: patio-runtime
```

AI Engine Runtime

- 统一监控端点，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）。



AI Engine Runtime

```yaml
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
      volumeMounts:
        - name: patio-group-config
          mountPath: /etc/patio
```

Mooncake Runtime

```yaml
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
      ...
```

Mount GroupTopo file for Scheduler

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

Register worker in Mooncake format

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

Set worker_role to decode

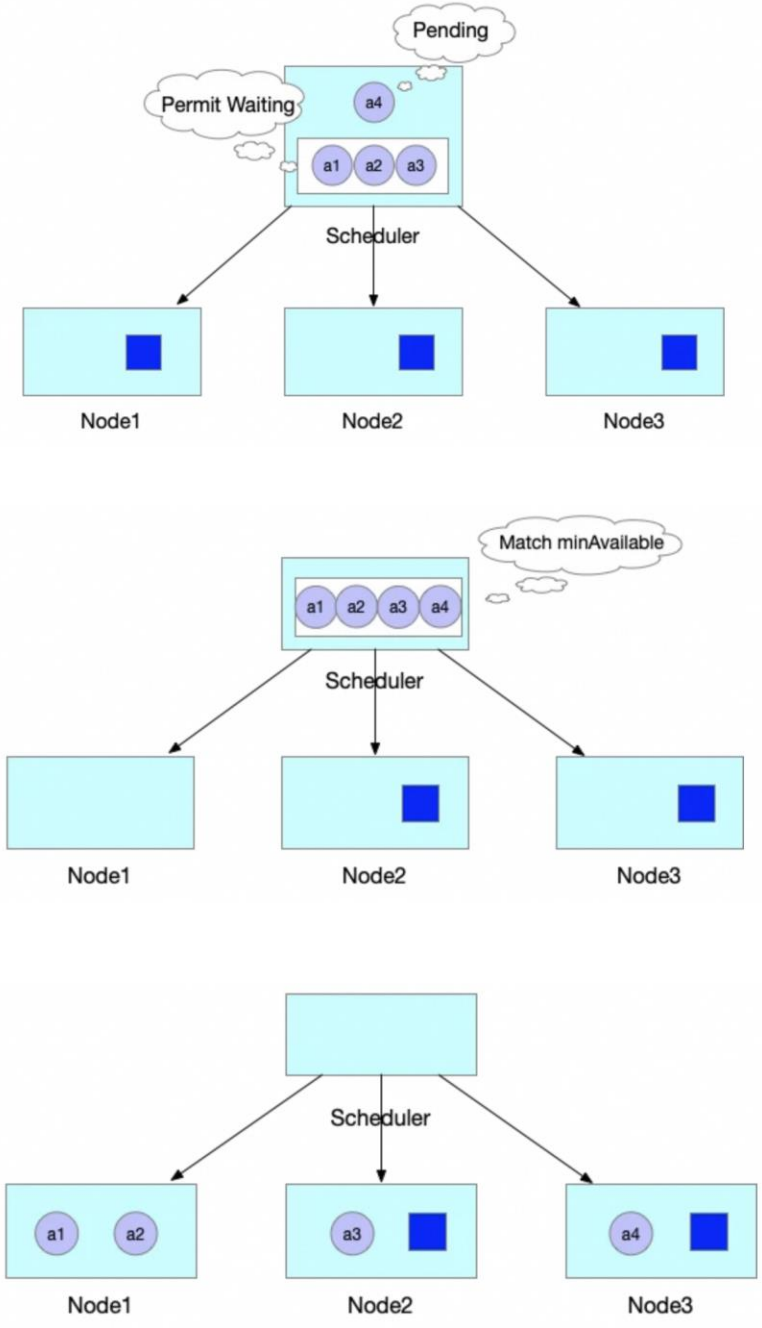
Gang Scheduling

- 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。

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

```yaml
# 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 is always equal to the number of pods in a rbg



Run 3P1D Qwen3-32B with Dynamo Using RBG

```yaml
apiVersion: workloads.x-k8s.io/v1alpha1
kind: RoleBasedGroup
metadata:
  name: dynamo-pd
spec:
  roles:
    - name: processor
      replicas: 1
      template:
        spec:
          containers:
            - name: processor
              command:
                - sh
                - -c
                - "cd /workspace/examples/llm; dynamo serve graphs.agg_router:Fro
ntend -f ./configs/qwq-32b.yaml"
              ...
```

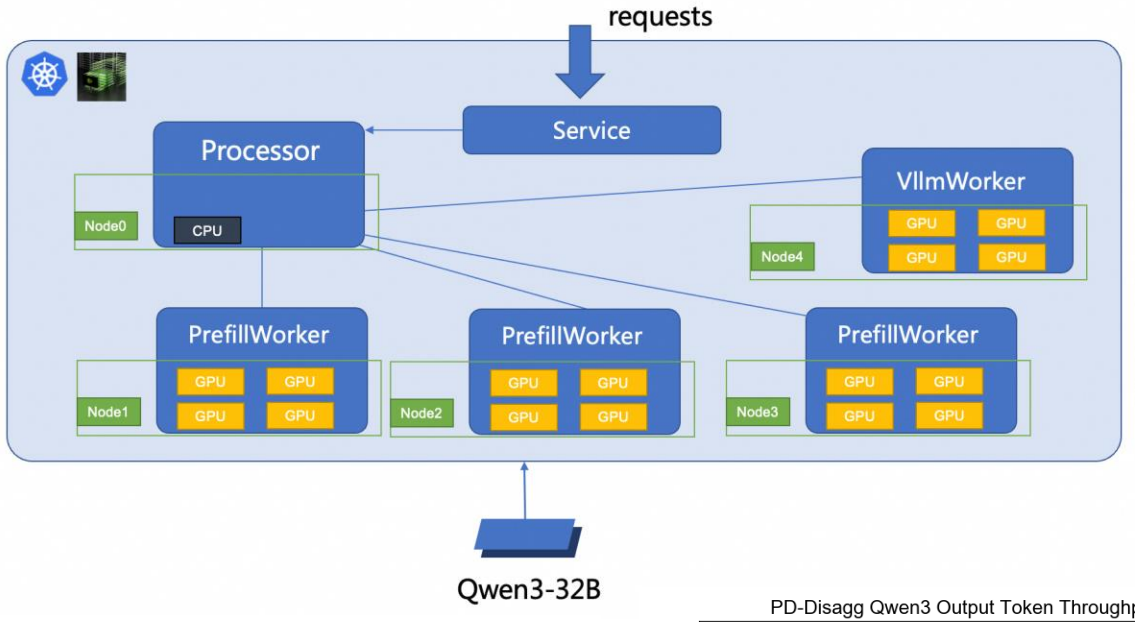
Processor Role

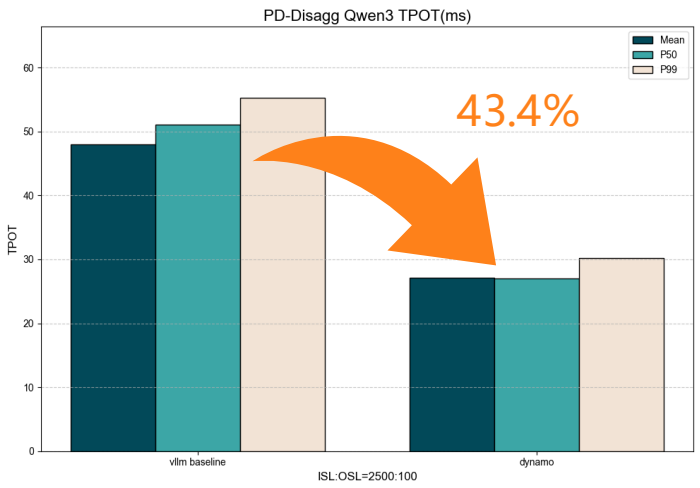
```yaml
    - name: prefill
      replicas: 3
      template:
        spec:
          containers:
            - name: prefillWorker
              command:
                - sh
                - -c
                - "cd /workspace/examples/llm; dynamo serve components.prefill_wo
rker:PrefillWorker -f ./configs/qwq-32b.yaml"
```

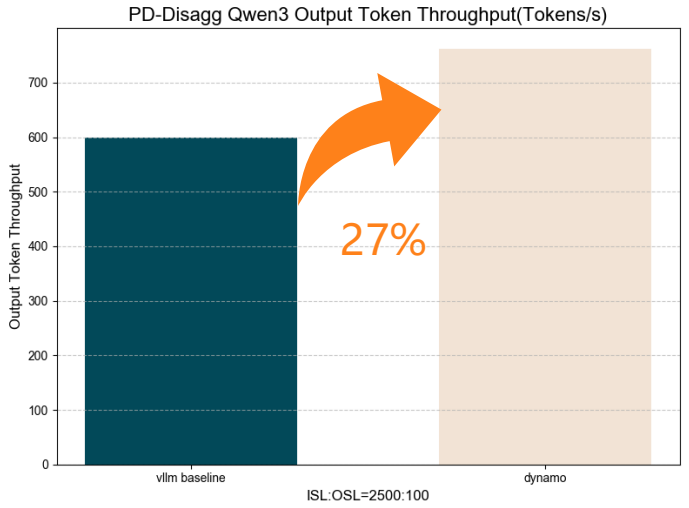
Prefill Role

```yaml
    - name: decode
      replicas: 1
      template:
        spec:
          containers:
            - name: vllmWorker
              command:
                - sh
                - -c
                - "cd /workspace/examples/llm; dynamo serve components.worker:Vll
mWorker -f ./configs/qwq-32b.yaml --service-name VllmWorker"
```

Decode Role





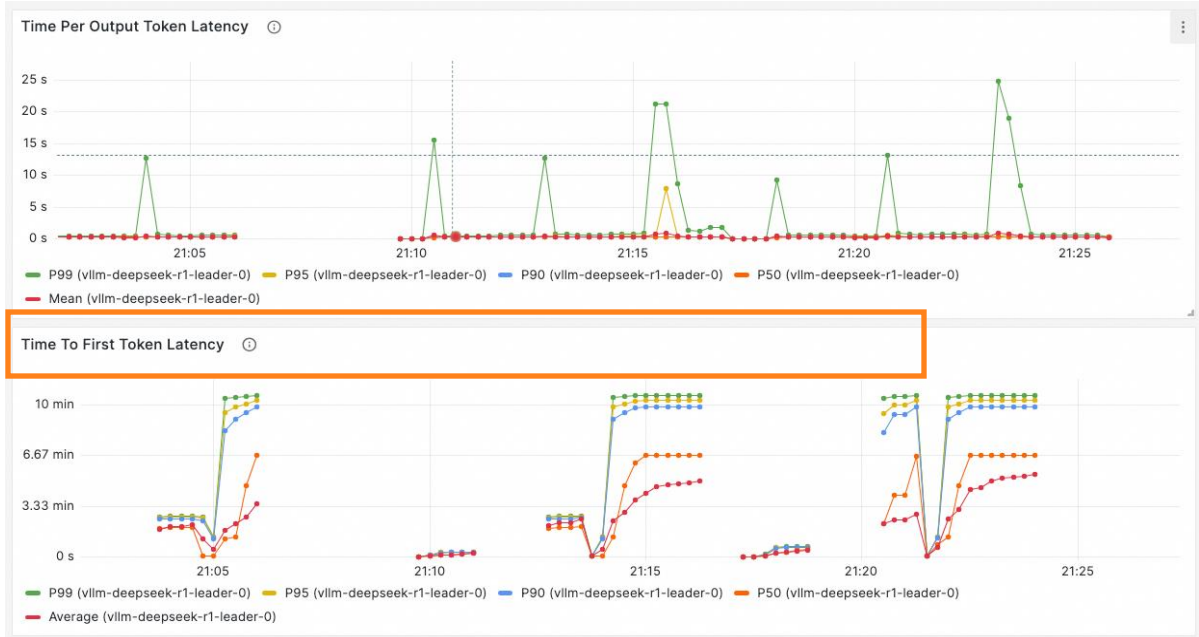


PART 3-2

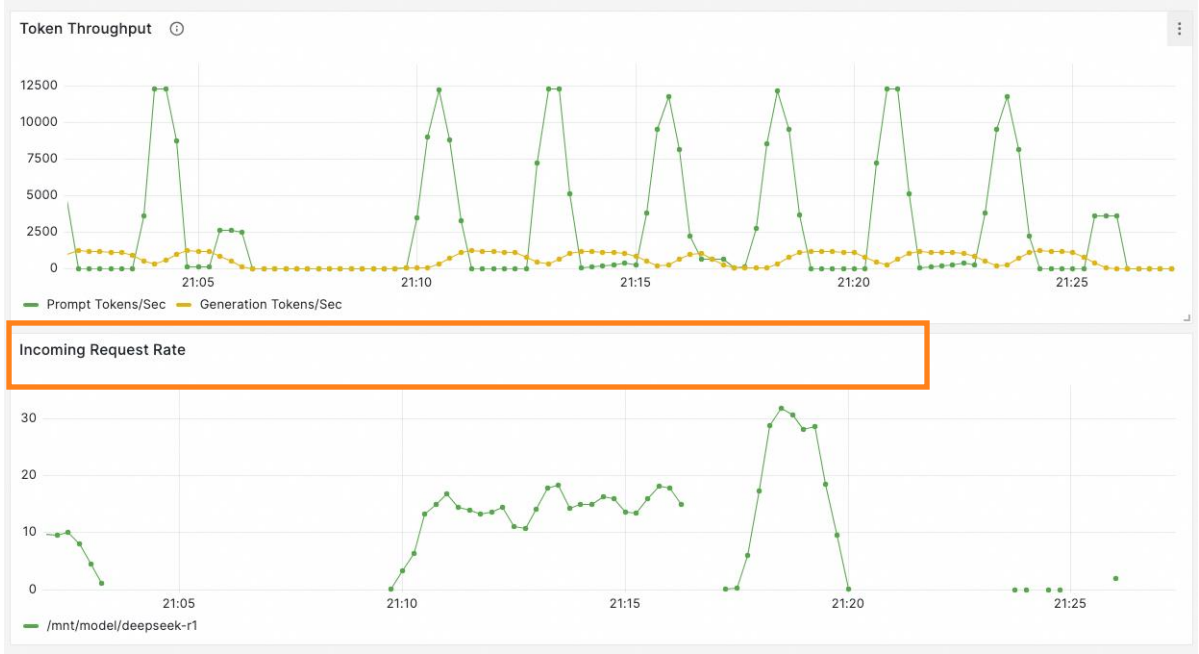
可观测与自动弹性伸缩

Monitoring

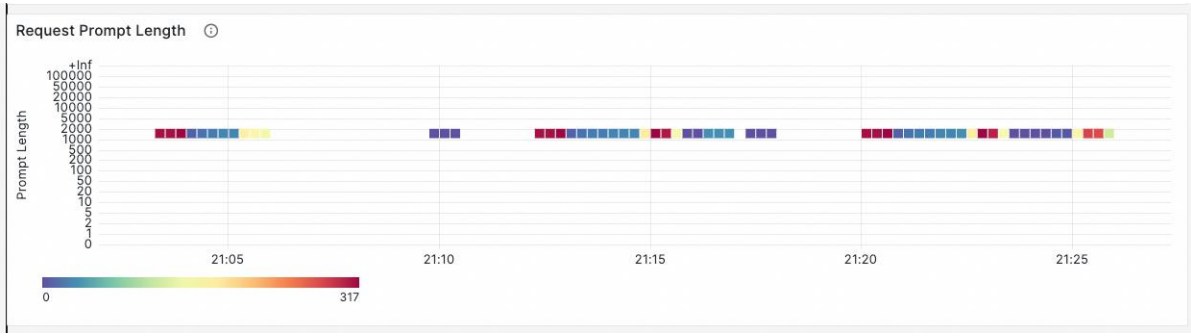
SLA Metrics: TTFT and TPOT



Throughput and Request Rate



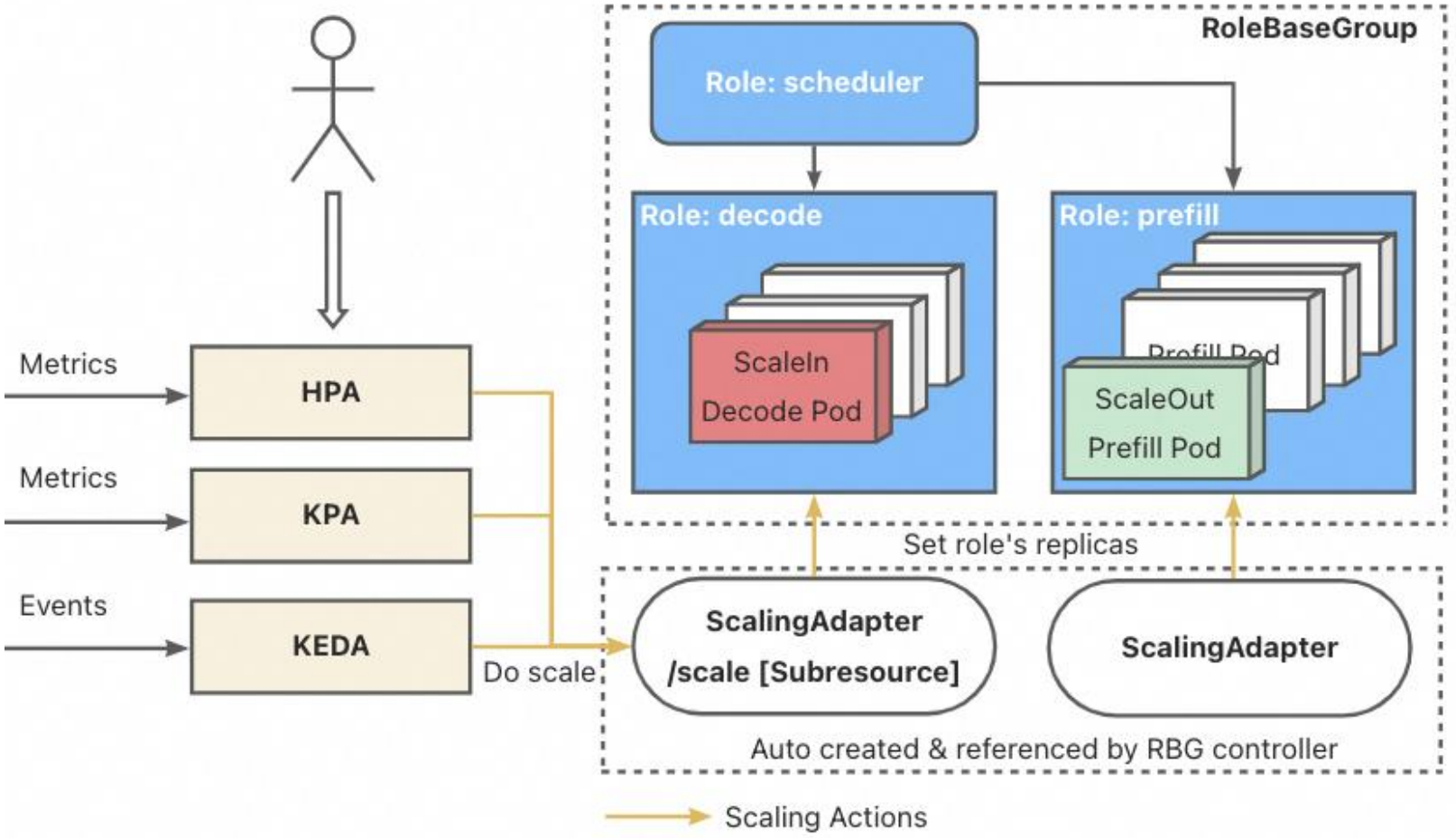
Request Feature Distribution (Input/Output Length, etc.)



Pod-level Performance

Autoscaling with HPA/KEDA/KPA

RBG scaling provides a scaling adapter that enables HPA/KEDA/KPA to adjust the number of role replicas.




```yaml
apiVersion: workloads.x-k8s.io/v1alpha1
kind: RoleBasedGroupScalingAdapter
metadata:
  name: llm-system-prefill-adapter
spec:
  scaleTargetRef:
    name: llm-system
    role: prefill
```

```yaml
apiVersion: workloads.x-k8s.io/v1alpha1
kind: RoleBasedGroupScalingAdapter
metadata:
  name: llm-system-decode-adapter
spec:
  scaleTargetRef:
    name: llm-system
    role: decode
```

LLM Benchmark

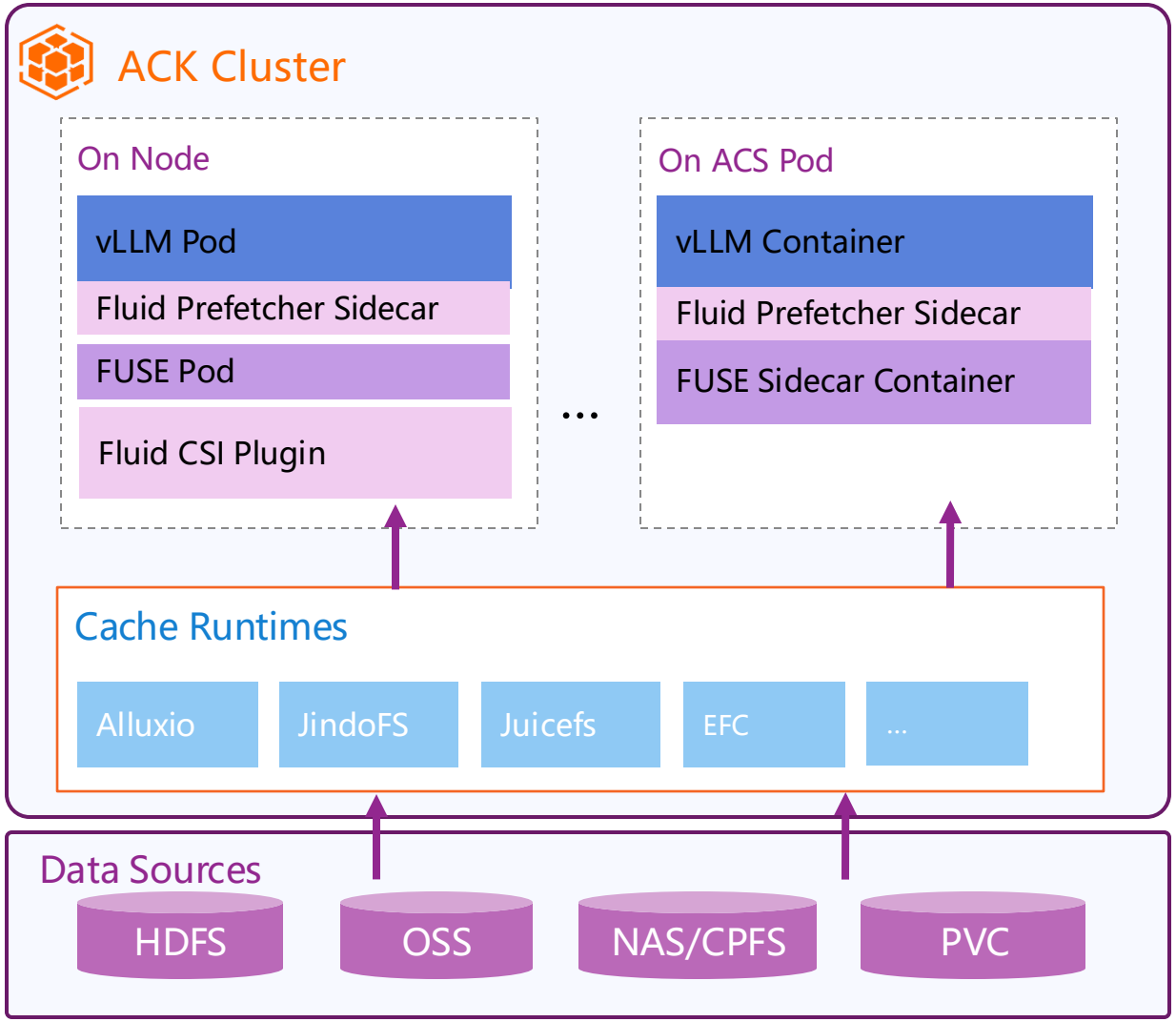
- ✓ 操作简单，多种工具集成在一个镜像中
- ✓ 支持多种框架如vllm benchmark，sglang benchmark，multi-round-qa等

```
python3 benchmarks/benchmark_serving.py \
--backend vllm \ # 指定推理框架
--model /root/Qwen2.5-7B-Instruct-GPTQ-Int4 \ # 推理模型
--trust-remote-code \
--dataset-name sonnet \ # 压测数据集
--dataset-path benchmarks/sonnet.txt \
--sonnet-input-len 1024 \ # input size
--sonnet-output-len 6 \ # output size
--sonnet-prefix-len 50 \ # input prefix size
--num-prompts 200 \ # 压测prompt数量
--request-rate 2 \ # qps
--port ${port} \
--host ${host} \
--save-result \
--result-dir ./results \
--result-filename qps-2.json
```

```
Starting initial single prompt test run...
Initial test run completed. Starting main benchmark run...
Traffic request rate: 2.0
Burstiness factor: 1.0 (Poisson process)
Maximum request concurrency: None
100%| 200/200 [01:39<00:00,  2.02it/s]
============ Serving Benchmark Result ============
Successful requests:                     200
Benchmark duration (s):                  99.11
Total input tokens:                      203135
Total generated tokens:                  1200
Request throughput (req/s):              2.02
Output token throughput (tok/s):         12.11
Total Token throughput (tok/s):          2061.80
---------------Time to First Token----------------
Mean TTFT (ms):                          369.50
Median TTFT (ms):                        260.50
P99 TTFT (ms):                           962.72
-----Time per Output Token (excl. 1st token)------
Mean TPOT (ms):                          91.61
Median TPOT (ms):                        18.04
P99 TPOT (ms):                           559.66
---------------Inter-token Latency----------------
Mean ITL (ms):                           91.55
Median ITL (ms):                         13.40
P99 ITL (ms):                            1946.27
==================================================
```

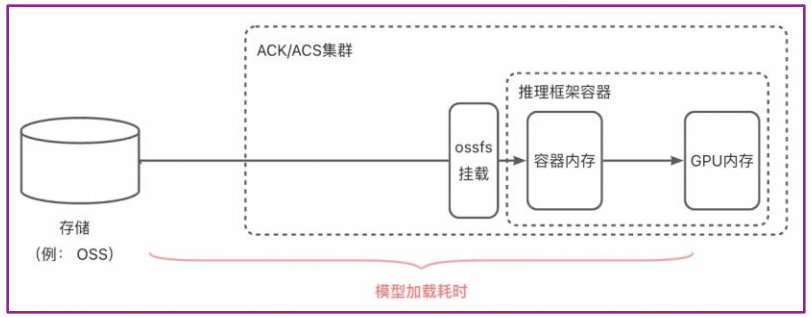
PART 3-3
模型加速与Profiling

基于Fluid的模型加载加速方案

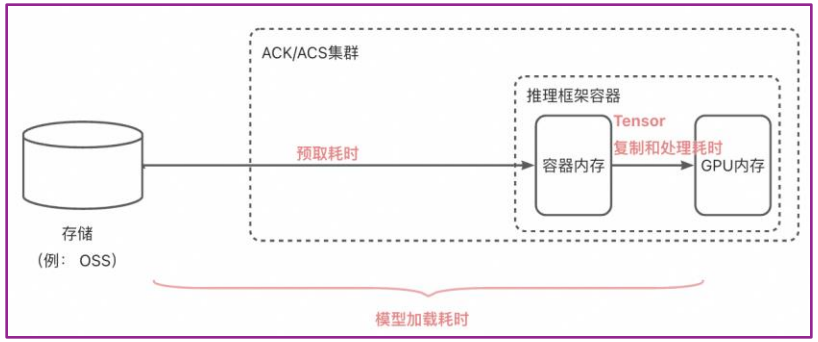


基于Fluid的模型加载加速方案

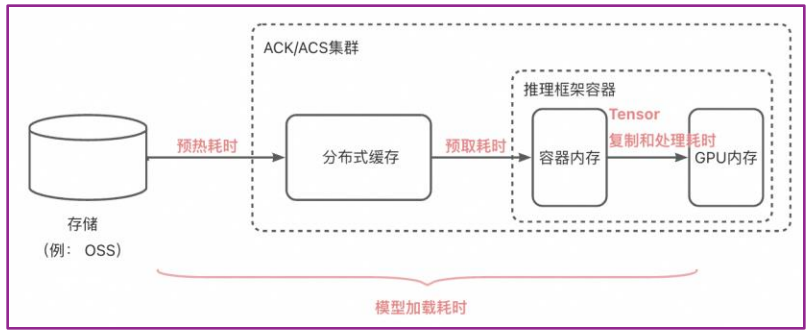
Baseline方案



Fluid方案（预取优化）



Fluid方案（分布式缓存 + 预取优化）



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 （ossfs-pro挂载）	39min	-
Fluid预取优化	242s（预取耗时） +72s（Tensor耗时）= **314s**	**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**	**减少90.2%**

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

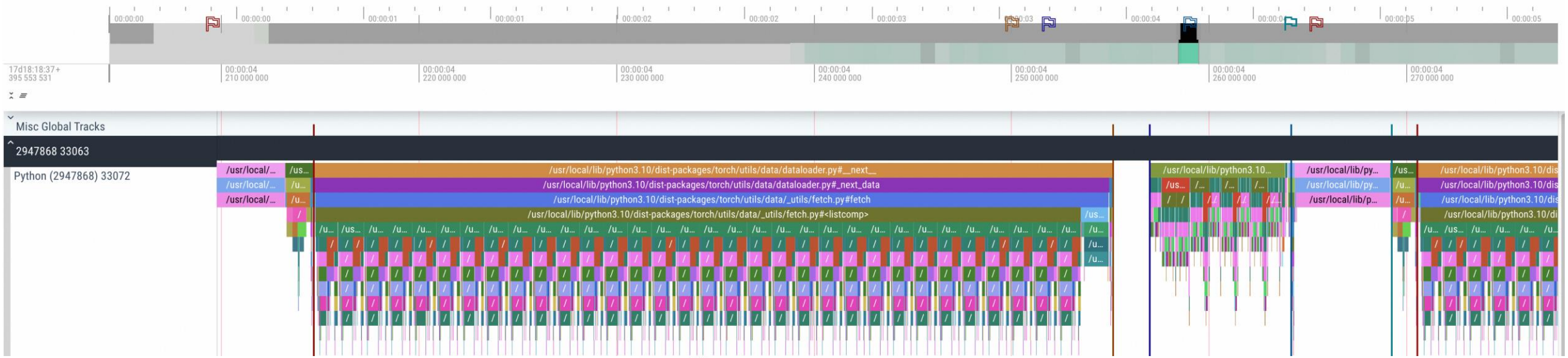
OSS理论带宽(100Gbps)利用率已达84%

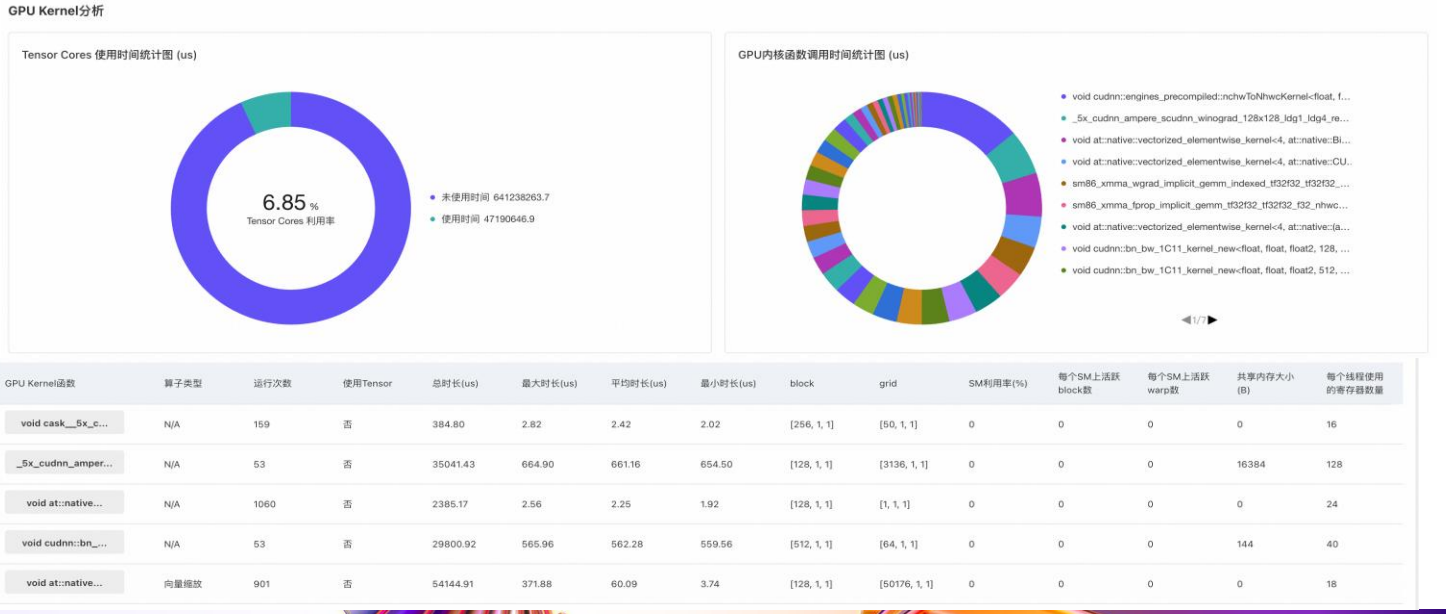
分布式缓存优化

* 上述数据从以下环境测得：

- 集群环境：
  - cn-beijing地域（OSS理论带宽100Gbps）
  - 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）

AI Profiling



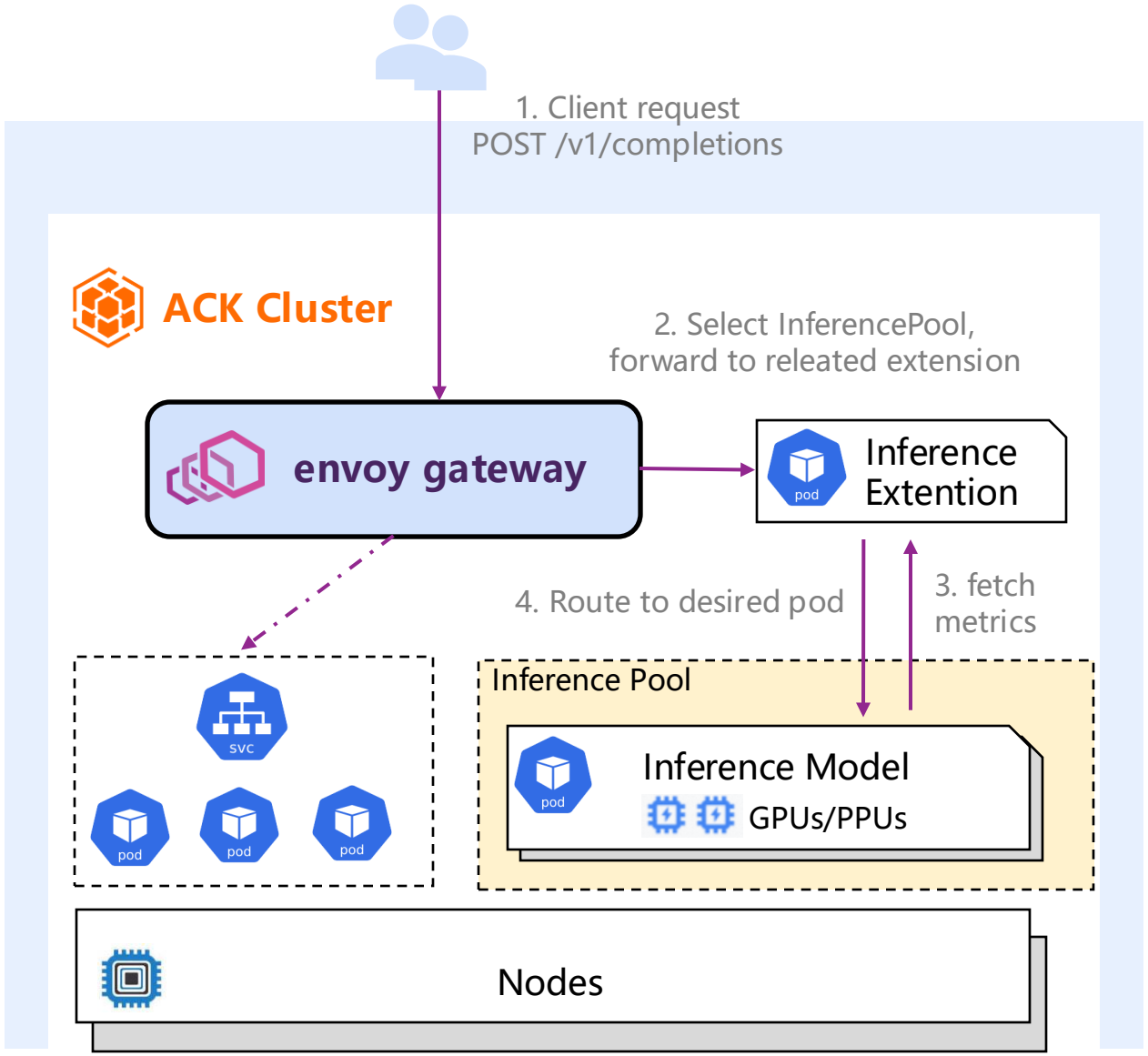


- ✓ 提供CPU、Python、Syscall、CUDA Lib、CUDA Kernel 的多个维度的Profiling项
- ✓ Online Profiling，业务无侵入，按需开启、低开
- ✓ 用户友好，有丰富的可视化页面
- ✓ 支持Step By Step的命令行、控制台、OpenAPI等多种使用方式

PART 3-4

智能路由网关

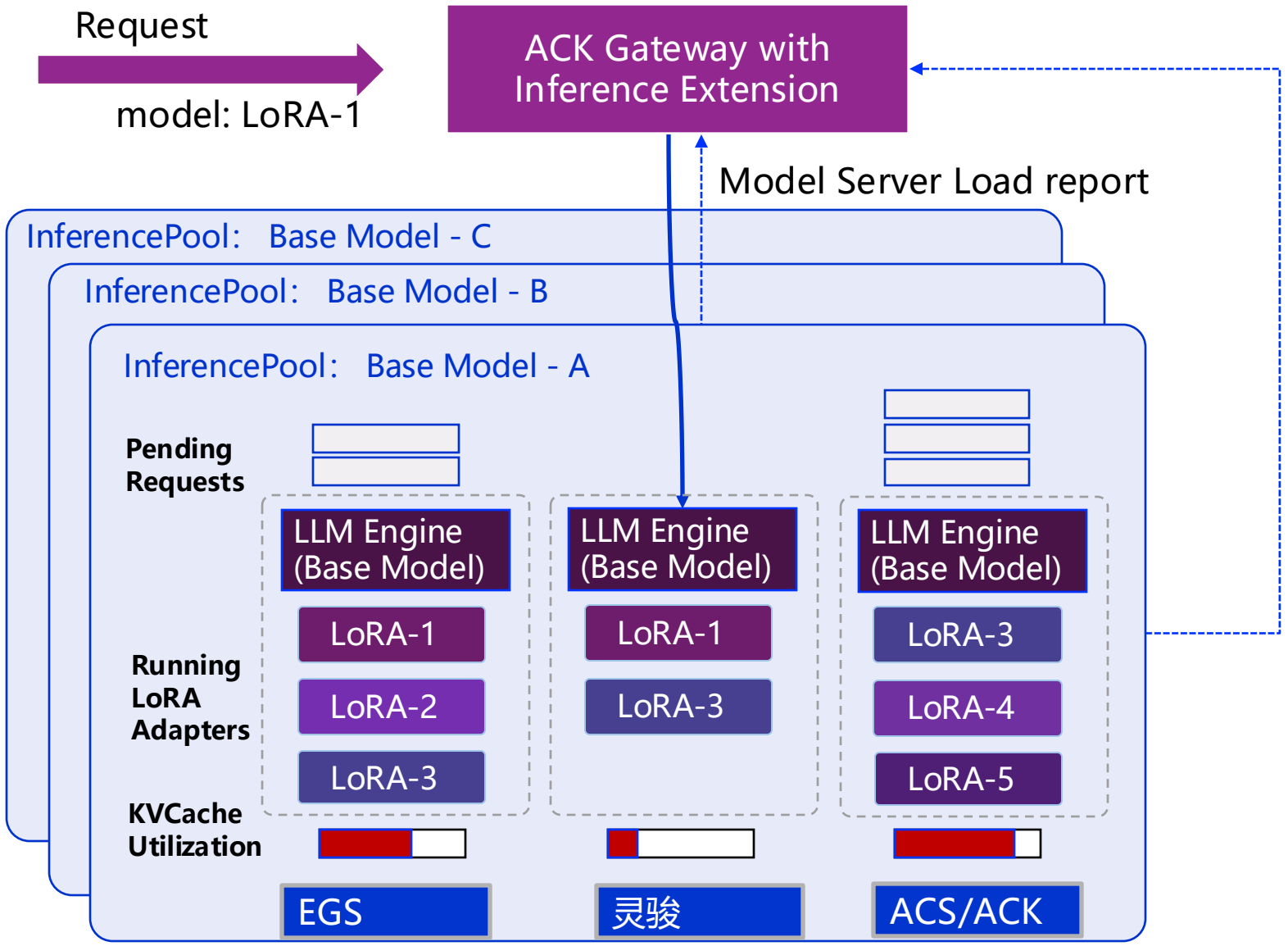
LLM 路由-Gateway Inference Extention



➢ 请求处理流程

1. Gateway选择正确的InferencePool（运行模型服务器框架的一组端点）或Service进行路由。
2. 如果请求需要路由到InferencePool，Gateway将把请求信息转发到InferencePool所关联的Inference extension。
3. Inference extension将从InferencePool端点中获取指标。
4. Inference extension将指示Gateway该请求应该路由到Pod。
5. 网关将请求路由到期望的Pod。

ACK Gateway With Inference Extension



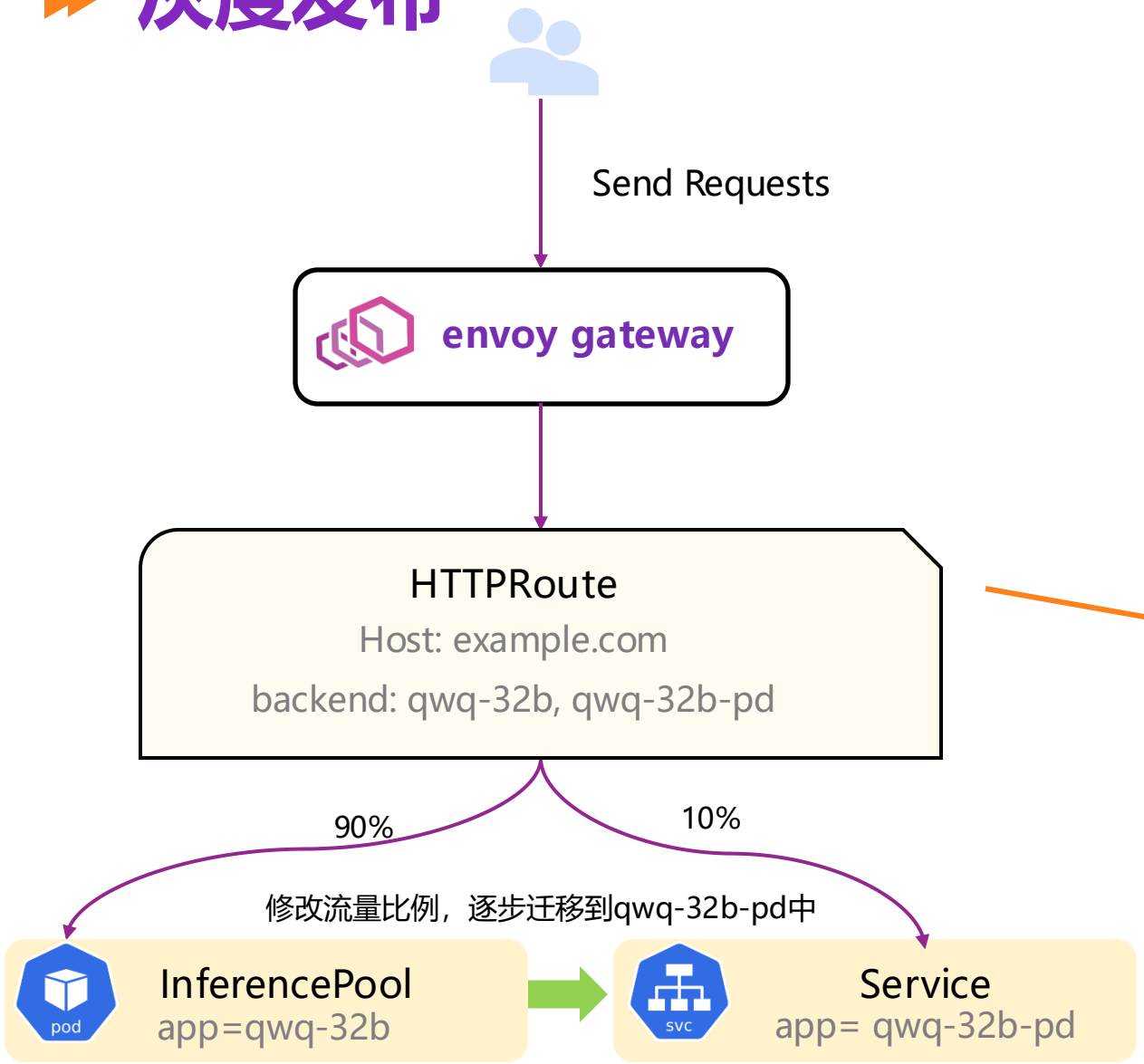
Model-aware Intelligence Routing

- 支持多种调度策略
  - 基于请求队列长度
  - 基于LoRA Adapter
  - KVCache-Aware
  - 基于模型重要性
- 支持异构推理环境

Queue Time	↓	90%
Avg TTFT	↓	73%
Cache Usage	↑	20%

非RoLA环境

灰度发布




```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: inference-route
  namespace: default
spec:
  parentRefs:
    - group: gateway.networking.k8s.io
      kind: Gateway
      name: inference-gateway
      sectionName: llm-gw
  rules:
    - backendRefs:
        - group: inference.networking.x-k8s.io
          kind: InferencePool
          name: qwq-32b-pool
          weight: 90
        - group: ''
          kind: Service
          name: qwq-32-pd-disagg-service
          port: 8008
          weight: 10
      matches:
        - path:
            type: PathPrefix
            value: /
```

PART 04

总结与展望

总结与展望

LLM基础知识

- Decode-Only架构是Token by Token方式生成文本，引入KV Cache可以减少计算量，提升推理效率。
- PD分离架构使得Prefill/Decoder阶段相互不影响，因此可以分开优化保证各自SLO，提升系统整体吞吐。

PD分离实现方案

- PD分离架构是指将Prefill和Decode阶段部署在不同机器上，在Prefill计算完成后将KV传输给Decode继续推理。
- 开源社区有众多实现方案如vLLM、SGLang、Dynamo等。各框架的PD分离方案实现各不相同。

总结与展望

➢ 云原生平台下PD分离架构规模化部署方案

- ACK推理套件提供了一系列工具支持在云原生平台下规模化部署PD分离架构。
- **应用部署**：RoleBasedGroup支持部署不同推理引擎（vLLM、SGlang、Dynamo等）PD分离架构。
- **弹性扩缩容**：支持Prefill/Decode分开弹性扩缩容，支持HPA/KEDA等策略
- **冷启动**：Fluid Cache可以极大地提升模型加载速度，减少应用冷启动时间
- **推理网关**：ACK Gateway是一款针对LLM推理研发的网关，支持KV Cache Aware、Pending queue等多种调度策略。

➢ 展望

- 推动开源社区PD分离框架实现方案，降低复杂性
- 持续集成最新推理引擎及推理优化技术，如DeepEP、KVCache Offloading等，提升LLM推理效率，降低成本

Reference

- vLLM: https://github.com/vllm-project/vllm.git
-  Dynamo: https://github.com/ai-dynamo/dynamo.git
- SGLang: https://github.com/sgl-project/sglang.git
- RoleBasedGroup (RBG): https://github.com/AliyunContainerService/rolebasedgroup

AiDD 8th 2025

第8届AI+研发数字峰会

拥抱AI 重塑研发 AI+ Development Digital Summit

查看会议详情

下一站预告 11/14-15 | 深圳站 12/19-20 | 上海站

深圳站论坛设置

智能装备与机器人

超越“编程 Copilot”

下一代知识工程

智能网联与汽车智能化

AI 测试工具开发与应用

AI 基础设施和运维

数据智能及其行业应用

可信 AI 安全工程

大模型和 AI 应用评测

多 Agent 协同框架

从智能测试到自主测试

大模型推理优化

多模态 LLM 训练与应用

智能化 DevOps 流水线

上下文工程

AiDD

「深行·浅智」

Walk Deep, Think Light.

2025.11.16

AiDD首届麦理浩径徒步

KEYLINK ing

科技生态圈峰会 + 深度研习

——1000 + 技术团队的共同选择

AiDD峰会详情

上海站

AI+研发数字峰会

AiDD 峰会

时间：

2025.05.23-24

北京站

AI+研发数字峰会

AiDD 峰会

时间：

2025.08.08-09

深圳站

AI+研发数字峰会

AiDD 峰会

时间：

2025.11.14-15

香港站

「深行·浅智」

AiDD 首届麦理浩径徒步

时间：

2025.11.16

上海站

AI+产品创新峰会

AiDD × 联合峰会

时间：

2025.12.19-20

第7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