云原生边云协同AI框架实践

×

普杰

华为云边缘云创新Lab 高级工程师
KubeEdge SIG AI Tech Lead

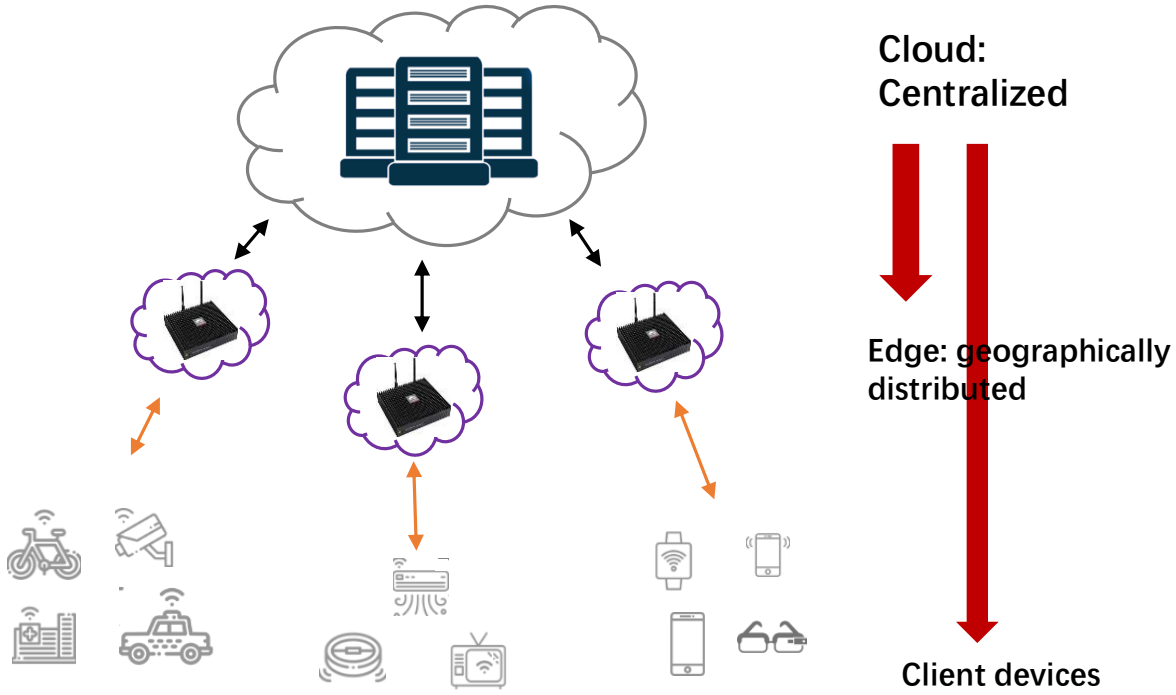
目 录

第一部分

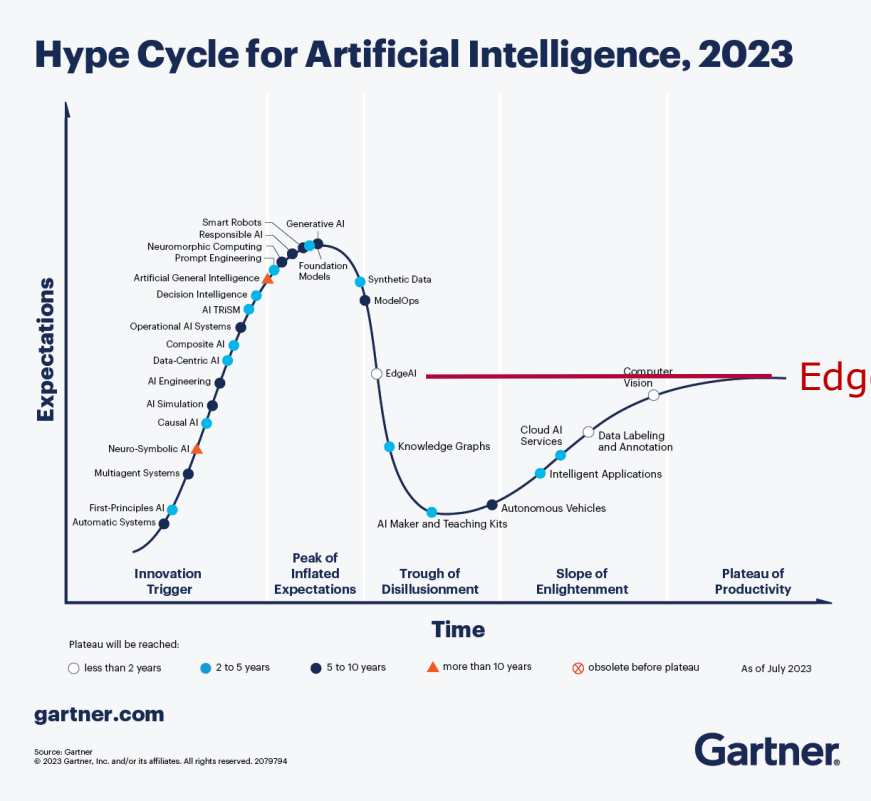
Edge AI现状与趋势

Why Edge AI?

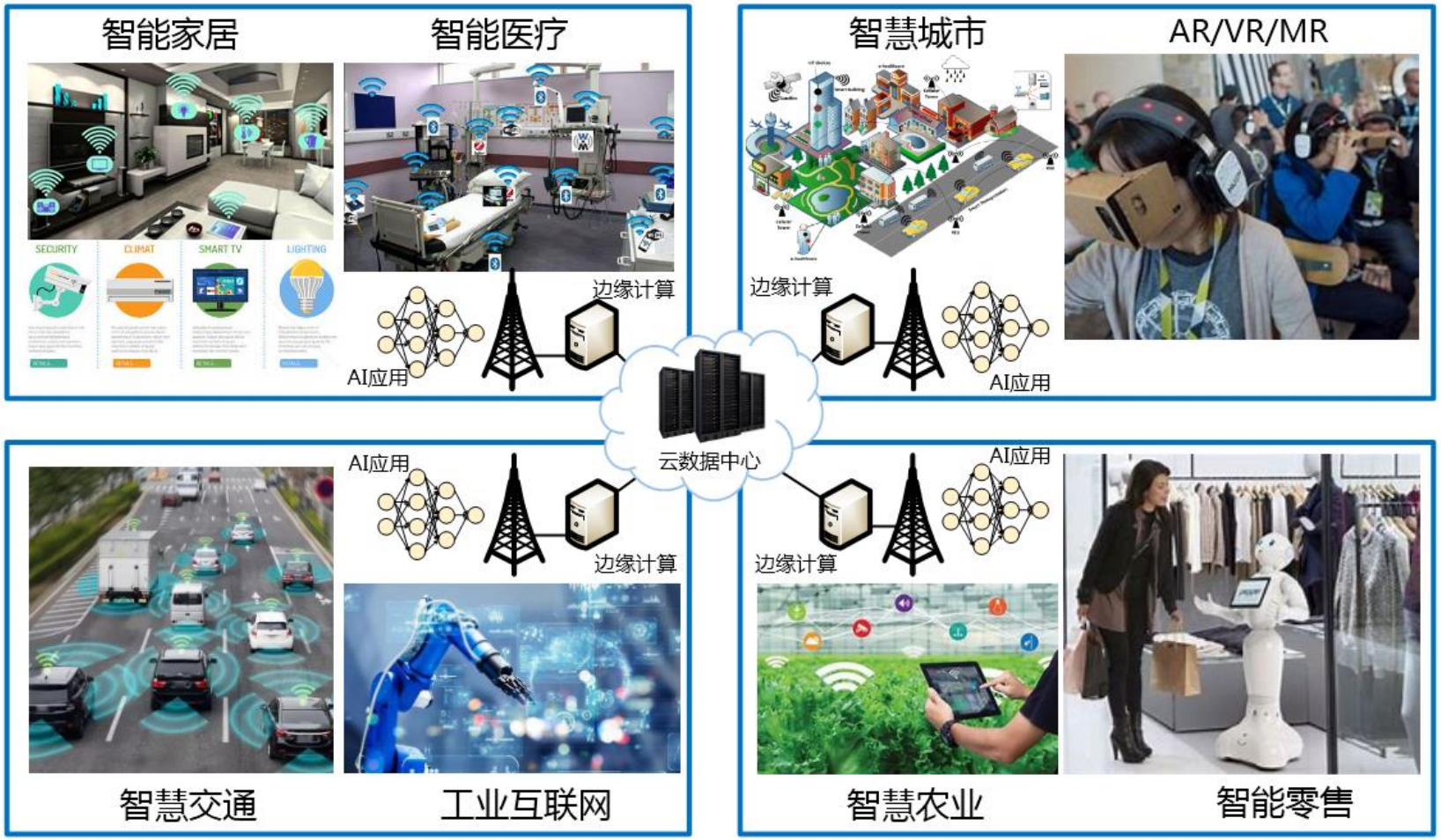
- Cloud中心化的AI计算范式不足以应对端上AI应用对实时性、准确性和强交互性的需求



- 随着大模型的发展，AI 计算对算力需求大幅且快速增长



AI应用到越来越多的边缘场景



分布式协同AI核心驱动力

- 随着边侧算力逐步强化，边缘AI持续演变至分布式协同AI

分布式协同AI概念

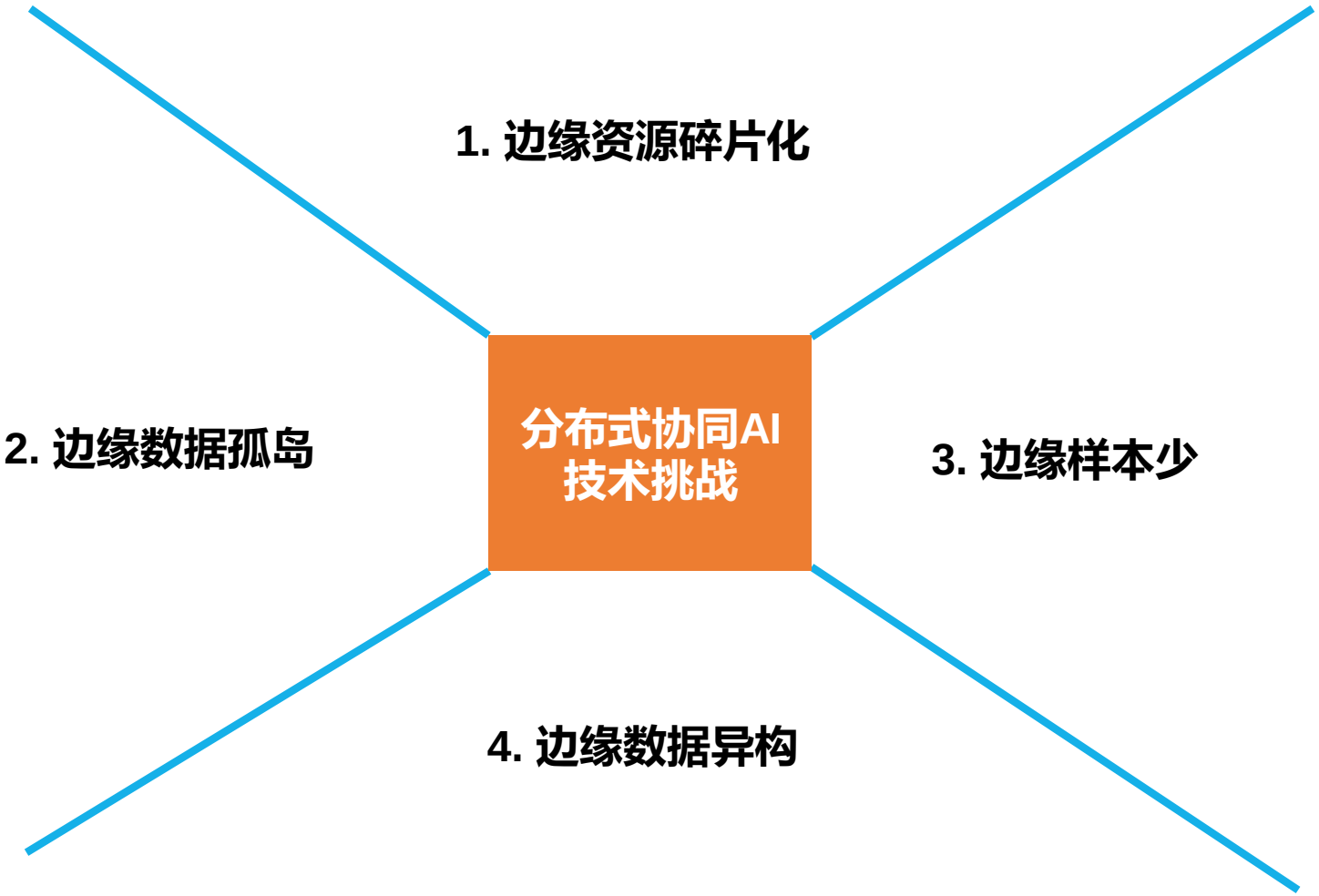
将人工智能相关的部分任务部署到边缘设备，基于边缘设备、边缘服务器、云服务器，利用分布式乃至分布式协同方式实现人工智能的技术

分布式协同AI核心驱动力

01 数据在边缘产生

02 边侧逐步具备AI能力

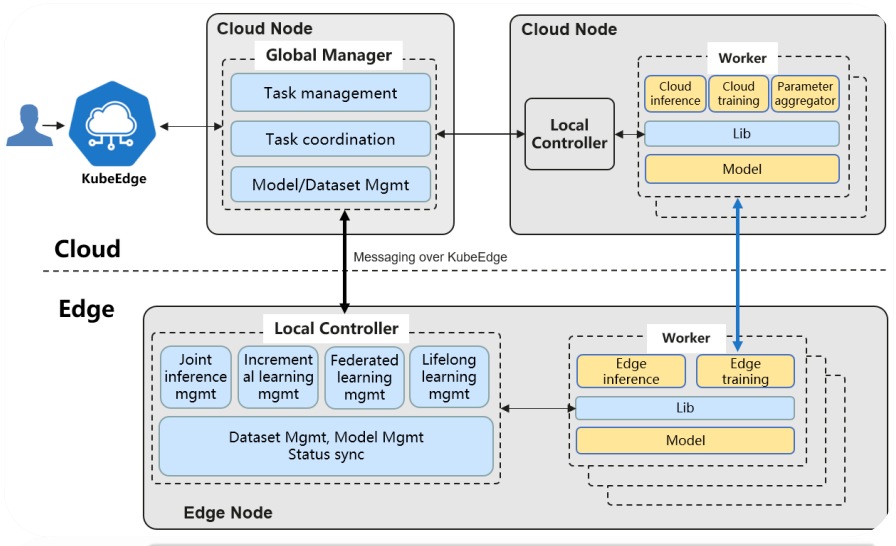
分布式协同AI技术挑战



第二部分

边云协同AI框架

开源分布式协同AI框架KubeEdge-Sedna



SIG成员在AI顶会IJCAI上分享分布式协同AI论文

SIG成员近年发表分布式协同AI顶会论文10+

Sedna斩获中国信通院云边协同应用创新大赛最佳创新奖

首个分布式协同AI开源项目Sedna

基于KubeEdge提供的边云协同能力，支持现有AI类应用无缝下沉到边缘

为分布式协同机器学习服务

- ✓ 降低构建与部署成本
- ✓ 提升模型性能
- ✓ 保护数据隐私

Sedna

基础框架

- ✓ 数据集管理
- ✓ 模型管理
- ✓ ……

训练推理框架

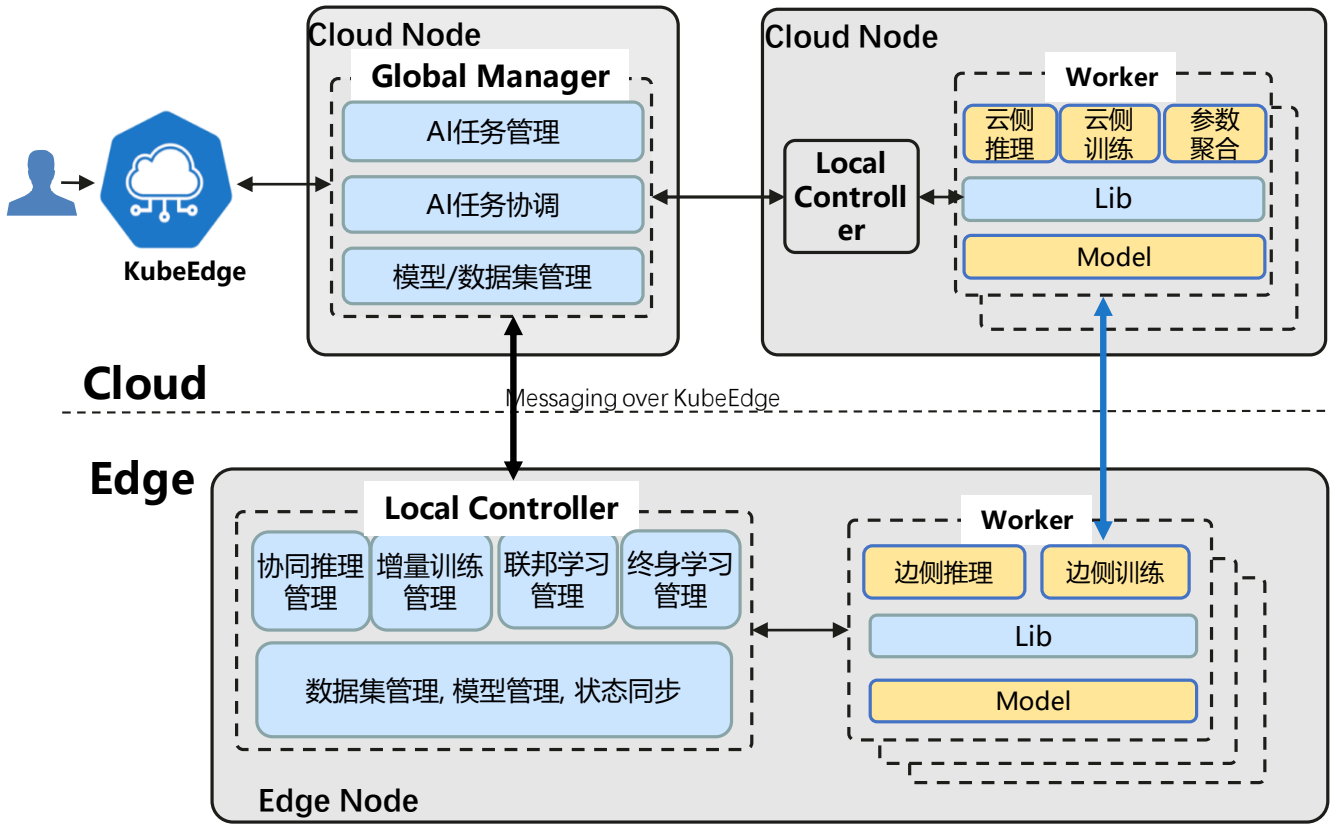
- ✓ 协同推理
- ✓ 增量学习
- ✓ 联邦学习
- ✓ 终身学习

兼容性

- ✓ 主流AI框架
- ✓ 模块算法
- ✓ 可扩展算法接口
- ✓ ……

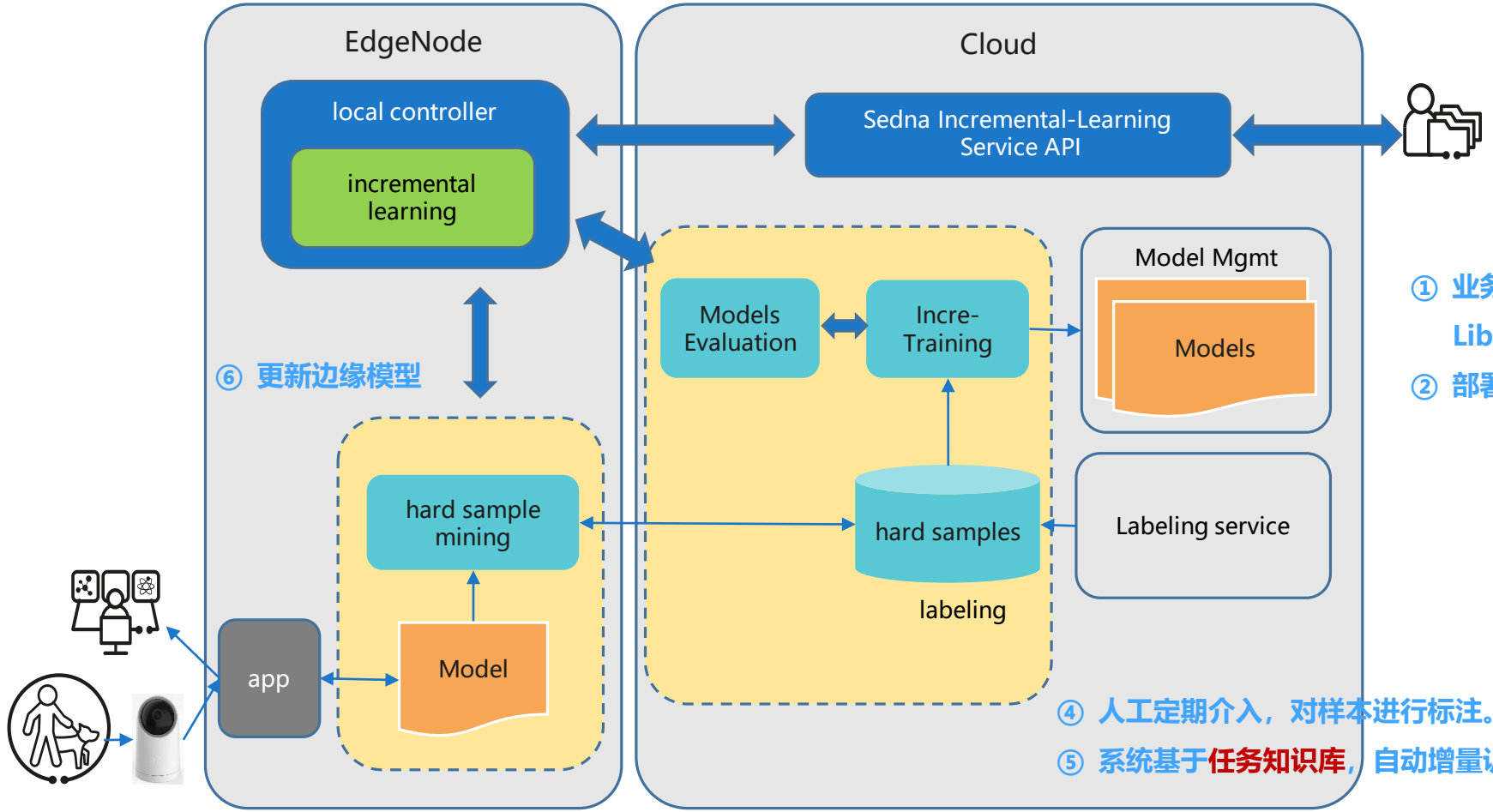
项目地址：https://github.com/kubeedge/sedna

边云协同AI框架 Sedna架构



1. **GlobalCoordinator**
   - 统一边云协同AI任务管理
   - 跨边云协同管理与协同
   - 中心配置管理
2. **LocalController**
   - 特性本地流程控制
   - 本地通用管理: 模型, 数据集等
3. **Worker**
   - 执行训练或推理任务, 训练/推理程序, 基于现有AI框架开
   - 按需启动, docker容器或function
   - 不同特性对应不同的worker组，可部署在边上或云上, 进行协同
4. **Lib**
   - 面向AI开发者和应用开发者, 暴露边云协同AI功能给应用

边云协同增量学习：小样本和非同分布下，模型越用越聪明



① 业务APP开发者：开发时使用边云AI Lib库，集成边云协同增量学习功能

② 部署业务APP，启动增量学习

④ 人工定期介入，对样本进行标注。

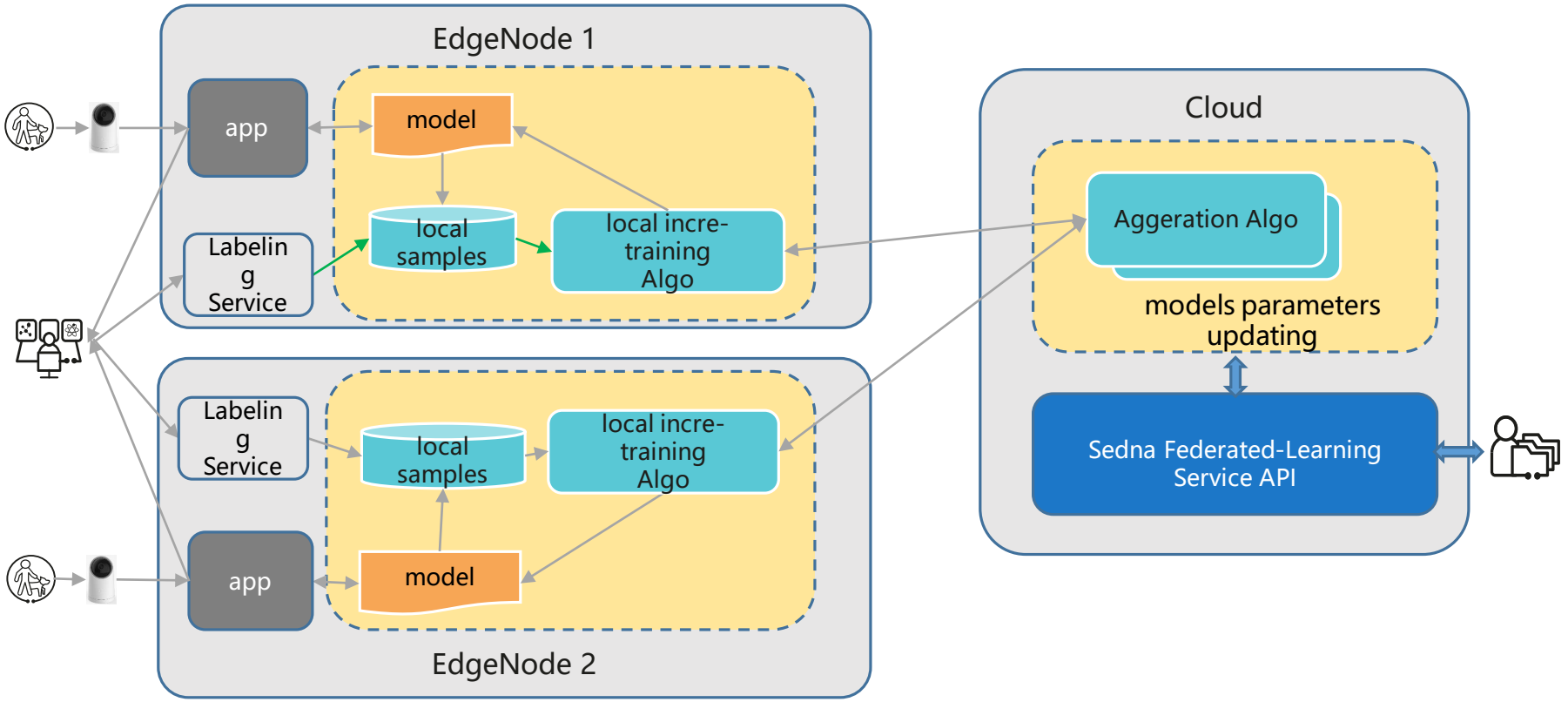
⑤ 系统基于**任务知识库**，自动增量训练产生新模型

③ APP运行，识别**难例**，上传到云上标注服务中

边云协同联邦学习：
孤岛数据不出边缘，知识聚合产生模型

③ 多任务检测，划分Non-IID样本集，与云端配合识别相似任务

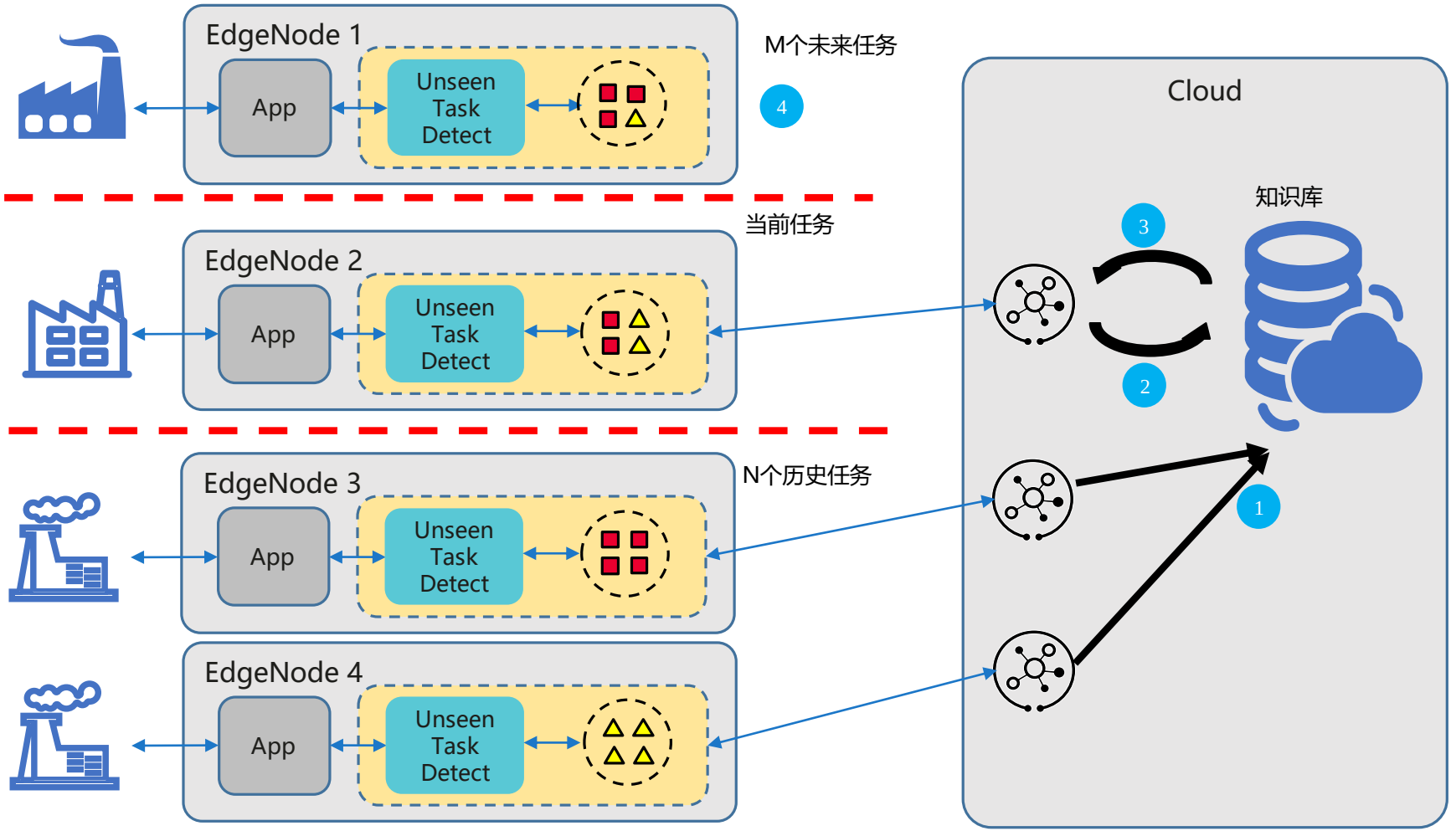
④ 本地训练，模型参数上传云端，云端运行跨边迁移+模型聚合算法。



① 开发者：导入边缘AI Lib库，开发边云协同联邦学习程序。

② 启动联邦学习任务，部署训练程序到边缘

边云协同终身学习：
云侧知识库记忆，解决新情景下数据异构和小样本



任务模型

任务样本

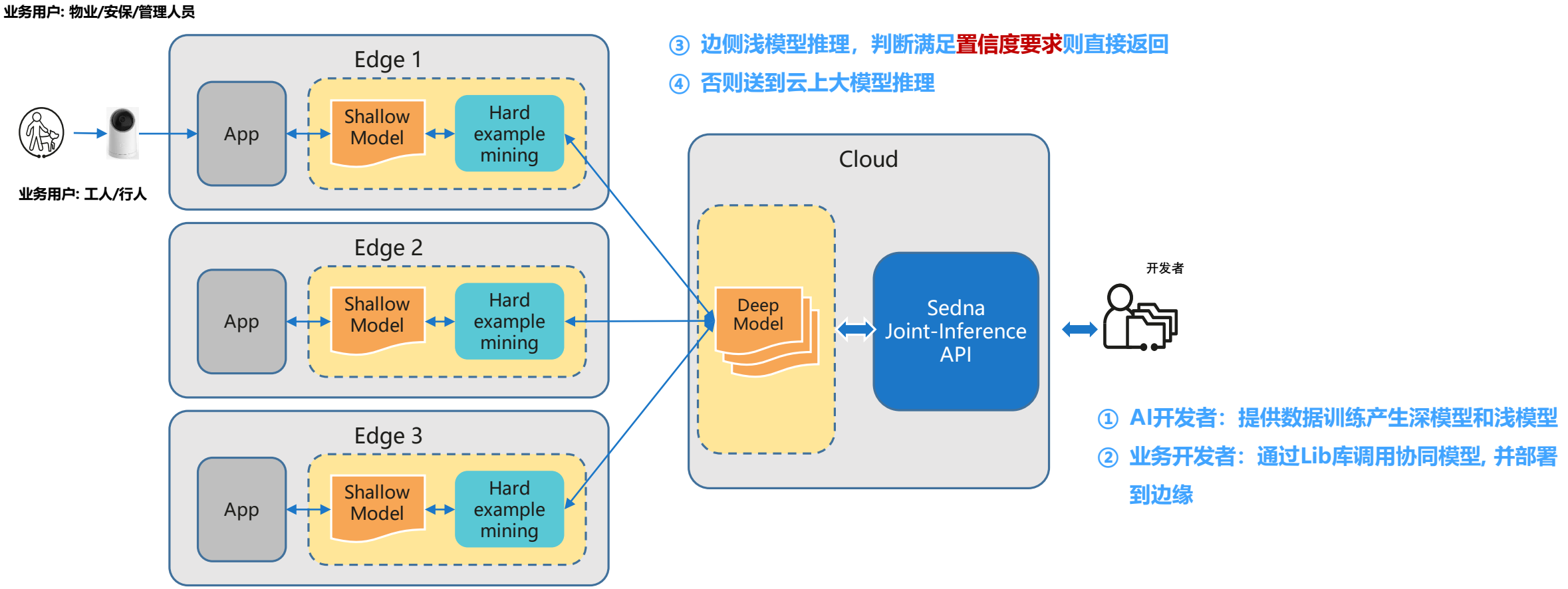
① 基于N个历史任务，完成云端知识库初始化

② 基于云端知识库，学习当前边侧任务

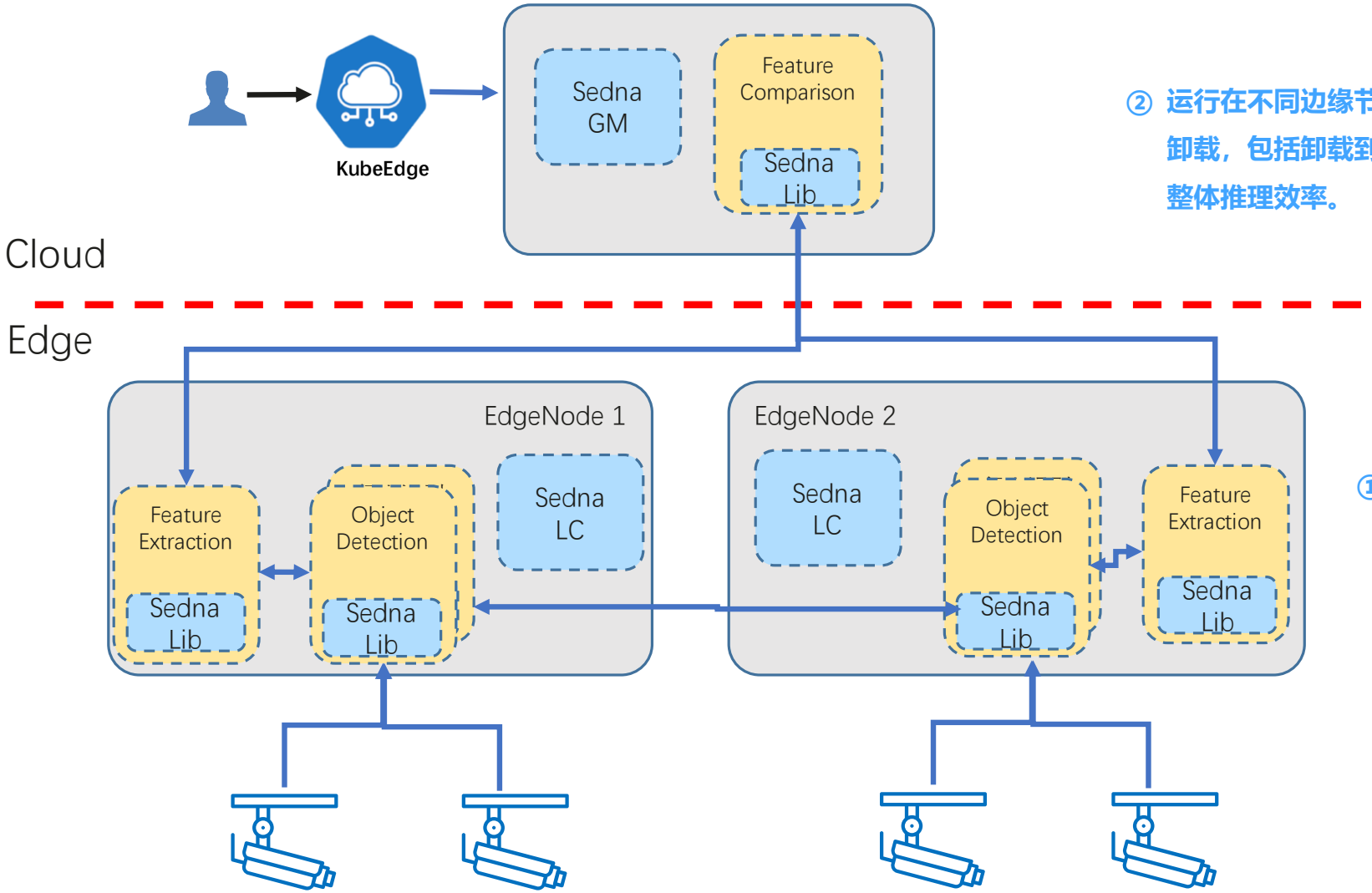
③ 基于当前边侧任务，更新云端知识库

④ 重复步骤2-3以处理M个未来任务

边云协同推理：
边侧资源受限条件下，提升整体推理性能



多边协同推理：
联合多边缘信息提升精度，卸载任务到多边缘提升资源利用率



② 运行在不同边缘节点的AI算法可以灵活资源卸载，包括卸载到其他边缘或云端，以提升整体推理效率。

① 部署在不同边缘节点的多个摄像头，可以提供不同视角、不同时间、不同光照的多维度信息，以提升整体识别准确度。

第三部分

Sedna-GM：一个K8S Operator

Operator：特定应用扩展K8S API的控制器

“Kubernetes Operator 是一种特定于应用的控制器，可扩展 Kubernetes API 的功能，来代表 Kubernetes 用户创建、配置和管理复杂应用的实例。”

•API: The data that describes the operand’s configuration. The API includes:

- **Custom resource definition (CRD)**, which defines a schema of settings available for configuring the operand.
- **Programmatic API**, which defines the same data schema as the CRD and is implemented using the operator’s programming language, such as Go.
- **Custom resource (CR)**, which specifies values for the settings defined by the CRD; these values describe the configuration of an operand.

•**Controller**: The brains of the operator. The controller creates managed resources based on the description in the custom resource; controllers are implemented using the operator’s programming language, such as Go.

https://www.redhat.com/zh/topics/containers/what-is-a-kubernetes-operator
https://developers.redhat.com/articles/2021/06/22/kubernetes-operators-101-part-2-how-operators-work

为什么使用Operator?

- **Kubernetes生态系统**

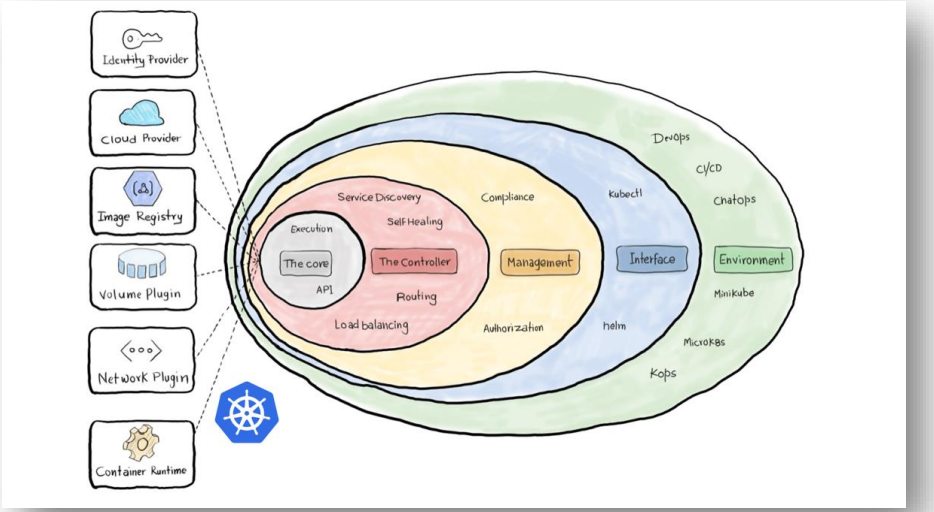
  比如监控、日志、Dashboard等

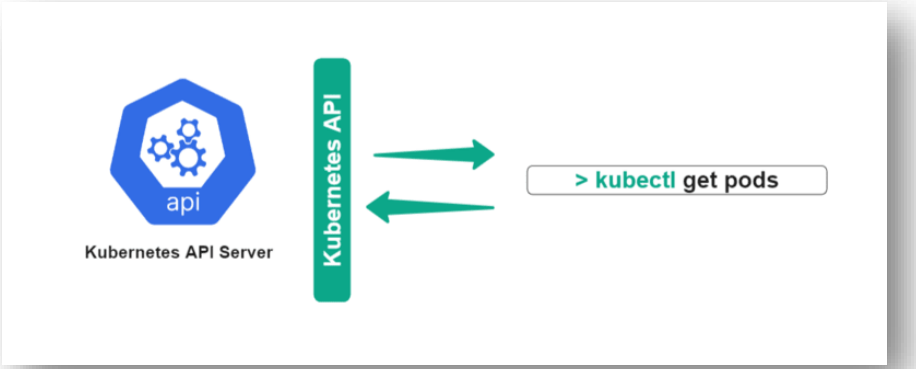
- **Kubernetes集群基础能力**

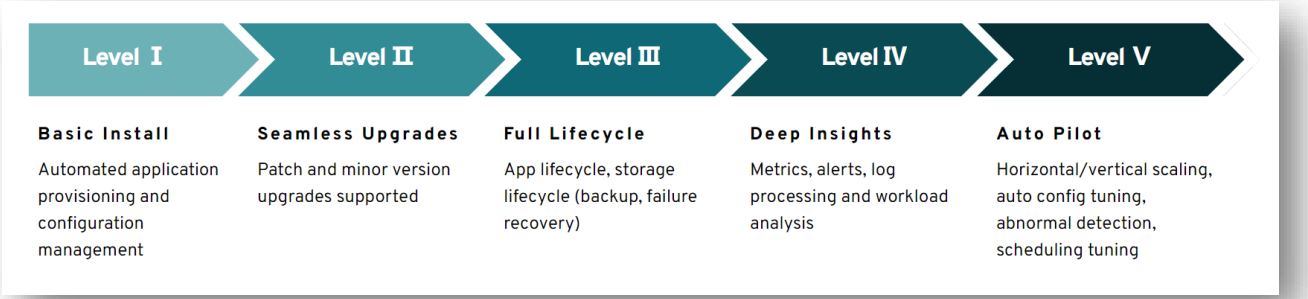
  比如自动化安装、配置、更新等。

- **Kubernetes API**

  避免了重复开发资源的增删改查等框架代码

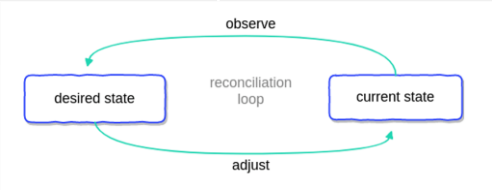


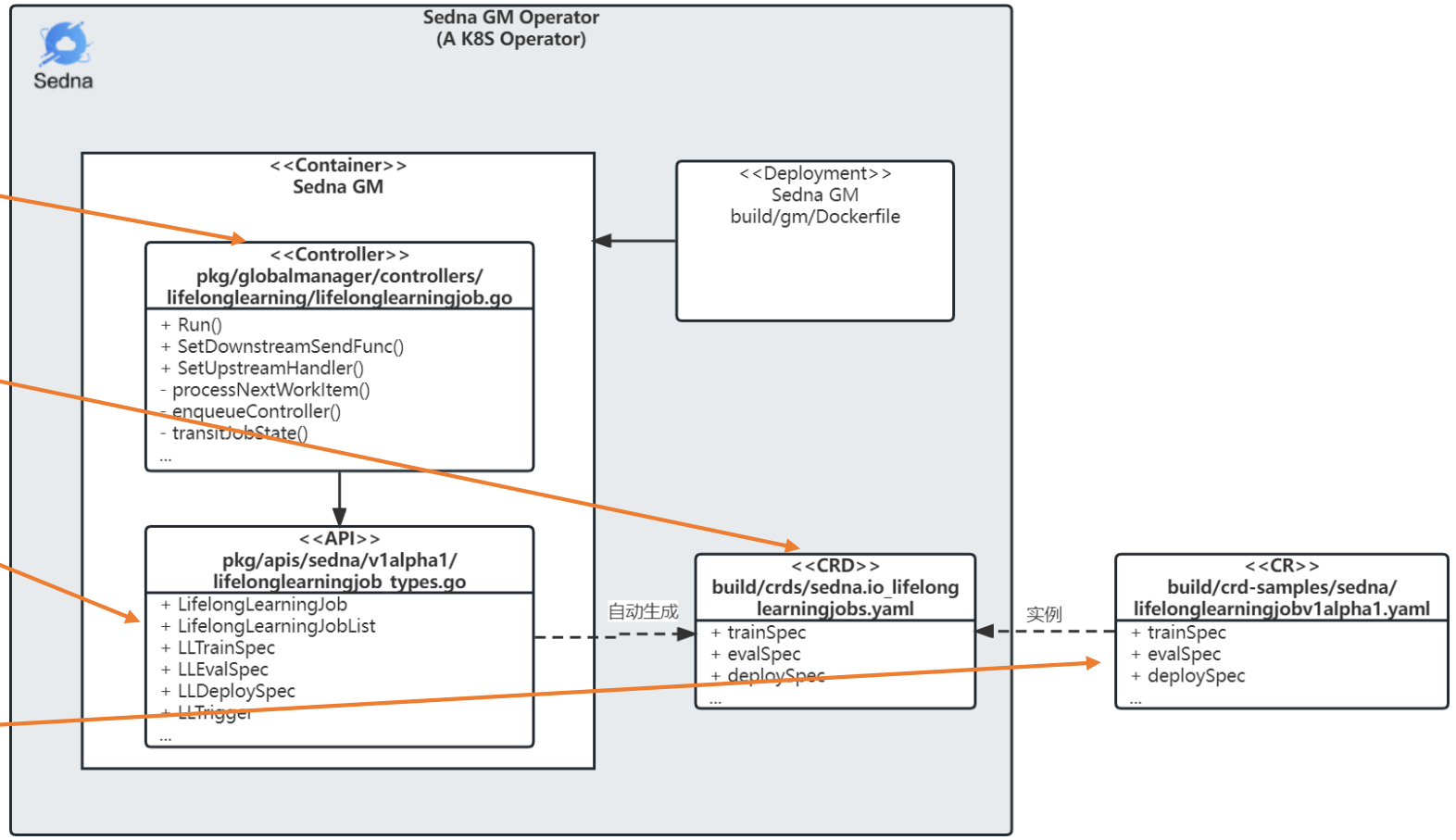




如何打造一个Operator?

Operator组件	描述
Controller	负责资源的reconcile核心逻辑实现。
CRD	自定义资源类型，以yaml文件呈现。
API	定义了与CRD相同的数据结构，面向编程实现，如Go语言。
CR	CRD定义的设置的值，这些值描述operand的配置。





API

pkg/apis/your_app/v1aplpha1/your_api.go

```
type LifelongLearningJob struct {
            metav1.TypeMeta   `json:",inline"`
            metav1.ObjectMeta `json:"metadata"`
            Spec              LLJobSpec   `json:"spec"`
            Status            LLJobStatus `json:"status,omitempty"`
}

type LLJobSpec struct {
            Dataset    LLDataset    `json:"dataset"`
            TrainSpec  LLTrainSpec  `json:"trainSpec"`
            EvalSpec   LLEvalSpec   `json:"evalSpec"`
            DeploySpec LLDeploySpec `json:"deploySpec"`
}

// LLJobStatus represents the current state of a lifelonglearning job
type LLJobStatus struct {
            // The latest available observations of a lifelonglearning job's current state.
            // +optional
            Conditions []LLJobCondition `json:"conditions,omitempty"`

            …
}
```

根据定义的结构，利用code-generator生成对应的operator代码

deepcopy-gen: 生成深度拷贝对象方法

client-gen: 为资源生成标准的操作方法(get;list;watch;create;update;patch;delete)

informer-gen: 生成informer，提供事件机制(AddFunc,UpdateFunc,DeleteFunc)来响应kubernetes的event

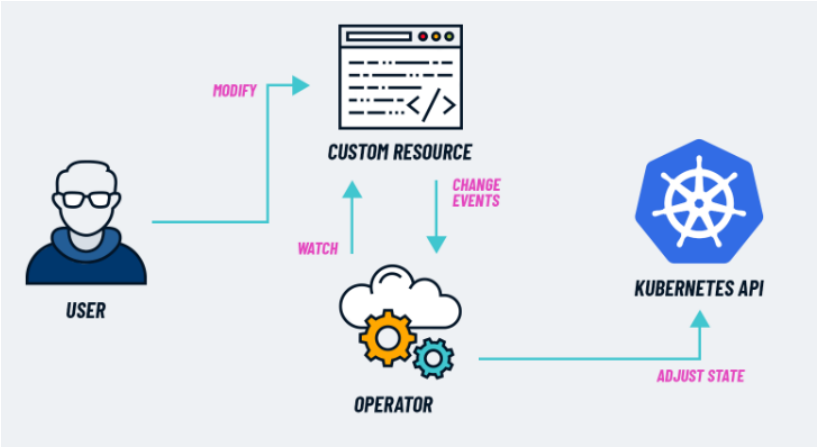
lister-gen: 为get和list方法提供只读缓存层

https://github.com/kubernetes/code-generator

CR & CRD

```
apiVersion: sedna.io/v1alpha1
kind: LifelongLearningJob
metadata:
  name: $job_name
spec:
  dataset:
    name: "lifelong-robo-dataset"
    trainProb: 0.8
  trainSpec: …
  evalSpec: …
  deploySpec:
    template:
      spec:
        nodeName: $INFER_NODE
        dnsPolicy: ClusterFirstWithHostNet
        hostNetwork: true
        containers:
        - image: $edge_image
          name:  infer-worker
          imagePullPolicy: IfNotPresent
          args: ["predict.py"]
          env:
            - name: "test_data"
              value: "/data/test_data"

          volumeMounts:
          - name: unseenurl
            mountPath: /data/unseen_samples
          resources:
            limits:
              cpu: 6
              memory: 12Gi
```



CR: 用于创建或更新自定义资源实例的yaml文件。

Controller

```
cmd/sedna-gm/sedna-gm.go/main() 【1】
pkg/globalmanager/controllers/manager.go/New() 【2】读取GM配置文件。
pkg/globalmanager/controllers/manager.go/Start() 【3】启动GM进程。
    - clientset.NewForConfig(): 【4】调用client-go生成了Sedna CRD client。
    - NewUpstreamController(): 【5】创建UpstreamController，每个GM进程有
一个UpstreamController
    - uc.Run(stopCh): 启动一个for循环协程，来处理
        - pkg/globalmanager/controllers/upstream.go/syncEdgeUpdate()
    - NewRegistry(): 【6】注册所有controller。
        - f.SetDownstreamSendFunc()【7】云端消息同步到边缘
            ->
pkg/globalmanager/controllers/lifelonglearning/downstream.go
        - f.SetUpstreamHandler()【8】边缘消息同步到云端
            ->
pkg/globalmanager/controllers/lifelonglearning/upstream.go/updateFromEd
ge()
        - f.Run()【9】Controller核心处理逻辑
    - ws.ListenAndServe() 【10】
```

整体的初始化逻辑。

【1】main函数入口

```
func main() {
   rand.Seed(time.Now().UnixNano())

   command := app.NewControllerCommand()
   logs.InitLogs()
   defer logs.FlushLogs()

   if err := command.Execute(); err != nil {
      os.Exit(1)
   }
}
```

【2】GM系统配置加载

```
// New creates the controller manager
func New(cc *config.ControllerConfig) *Manager {
   config.InitConfigure(cc)
   return &Manager{
      Config: cc,
   }
}
```

【3】GM整体初始化

```
// Start starts the controllers it has managed
func (m *Manager) Start() error {
   ...
   sednaClient, err := clientset.NewForConfig(kubecfg)
   ...
   sednaInformerFactory := sednainformers.NewSharedInformerFactoryWithOptions(sednaClient,
genResyncPeriod(minResyncPeriod), sednainformers.WithNamespace(namespace))

   uc, _ := NewUpstreamController(context)
   downstreamSendFunc := messagelayer.NewContextMessageLayer().SendResourceObject
   stopCh := make(chan struct{})
   go uc.Run(stopCh)

   for name, factory := range NewRegistry() {
      ...
      f.SetDownstreamSendFunc(downstreamSendFunc)
      f.SetUpstreamHandler(uc.Add)
      ...
      // 启动各个特性对应controller
      go f.Run(stopCh)
   }

   ...

   ws := websocket.NewServer(addr)
   ...
}
```

- 初始化Sedna CRD client，Controller会监听Sedna CR 增删改查的变化，并执行对应的处理逻辑。
- 初始化UpstreamController，用于处理边缘LC上传的消息
- 针对每个特性（协同推理、终身学习等），绑定对应的消息处理函数

【4】CRD client初始化

```
// New creates a new LifelongLearningJob controller that keeps the relevant
pods
// in sync with their corresponding LifelongLearningJob objects.
func New(cc *runtime.ControllerContext) (runtime.FeatureControllerI, error)
{
   cfg := cc.Config

   podInformer := cc.KubeInformerFactory.Core().V1().Pods()

   // 获取LifelongLearningJob的Informer
   jobInformer :=
cc.SednaInformerFactory.Sedna().V1alpha1().LifelongLearningJobs()

   eventBroadcaster := record.NewBroadcaster()
   eventBroadcaster.StartRecordingToSink(&v1core.EventSinkImpl{Interface:
cc.KubeClient.CoreV1().Events("")})

   // 配置LifelongLearningJob Controller的参数
   jc := &Controller{
      kubeClient: cc.KubeClient,
      client:     cc.SednaClient.SednaV1alpha1(),
      queue:
workqueue.NewNamedRateLimitingQueue(workqueue.NewItemExponentialFailureRateL
imiter(runtime.DefaultBackOff, runtime.MaxBackOff), Name),
      cfg:        cfg,
   }
```

```
   // 绑定LifelongLearningJob CRD资源的Add、Update、Delete对应事件的回调函数。
   jobInformer.Informer().AddEventHandler(cache.ResourceEventHandlerFuncs{
      AddFunc: func(obj interface{}) {
         jc.enqueueController(obj, true)
         jc.syncToEdge(watch.Added, obj)
      },
      UpdateFunc: func(old, cur interface{}) {
         jc.enqueueController(cur, true)
         jc.syncToEdge(watch.Added, cur)
      },
      DeleteFunc: func(obj interface{}) {
         jc.enqueueController(obj, true)
         jc.syncToEdge(watch.Deleted, obj)
      },
   })
   jc.jobLister = jobInformer.Lister()
   jc.jobStoreSynced = jobInformer.Informer().HasSynced

   // 绑定Pod对应的增删改对应事件的回调函数。
   podInformer.Informer().AddEventHandler(cache.ResourceEventHandlerFuncs{
      AddFunc:    jc.addPod,
      UpdateFunc: jc.updatePod,
      DeleteFunc: jc.deletePod,
   })
   jc.podStore = podInformer.Lister()
   jc.podStoreSynced = podInformer.Informer().HasSynced

   return jc, nil
}
```

【5】消息处理初始化

```
// syncEdgeUpdate receives the updates from edge and syncs these to k8s.
func (uc *UpstreamController) syncEdgeUpdate() {
   for {
      select {
      case <-uc.messageLayer.Done():
         klog.Info("Stop sedna upstream loop")
         return
      default:
      }

      update, err := uc.messageLayer.ReceiveResourceUpdate()
            ...

      handler, ok := uc.updateHandlers[kind]
      if ok {
         err := handler(name, namespace, operation, update.Content)
         ...
      }
   }
}
```

【6】Controller注册

```
func NewRegistry() Registry {
   return Registry{
      ji.Name:      ji.New,
      fe.Name:      fe.New,
      fl.Name:      fl.New,
      il.Name:      il.New,
      ll.Name:      ll.New,
      reid.Name:    reid.New,
      va.Name:      va.New,
      dataset.Name: dataset.New,
      objs.Name:    objs.New,
   }
}
```

每个资源对象都会注册一个对应的controller。

【7】云端消息同步到边缘

```
func (c *Controller) syncToEdge(eventType watch.EventType, obj interface{}) error {

   // 获取到对应的数据集指定的节点（Dataset CRD对象中有一个字段记录了Node名称）
   ds, err := c.client.Datasets(job.Namespace).Get(context.TODO(), dataName, metav1.GetOptions{})

   // 获取到训练、评估、部署对应的节点名称
   getAnnotationsNodeName := func(nodeName sednav1.LLJobStage) string {
      return runtime.AnnotationsKeyPrefix + string(nodeName)
   }
   ann := job.GetAnnotations()
   if ann != nil {
      trainNodeName = ann[getAnnotationsNodeName(sednav1.LLJobTrain)]
      evalNodeName = ann[getAnnotationsNodeName(sednav1.LLJobEval)]
      deployNodeName = ann[getAnnotationsNodeName(sednav1.LLJobDeploy)]
   }

   ...

   // 根据LifelongLearningJob所处阶段不同，发送消息到不同的节点上
   switch jobStage {
   case sednav1.LLJobTrain:
      doJobStageEvent(trainNodeName)
   case sednav1.LLJobEval:
      doJobStageEvent(evalNodeName)
   case sednav1.LLJobDeploy:
      doJobStageEvent(deployNodeName)
   }

   return nil
}
```

【8】边缘消息同步到云端

```
// updateFromEdge syncs the edge updates to k8s
func (c *Controller) updateFromEdge(name, namespace, operation string, content
[]byte) error {
   var jobStatus struct {
      Phase  string `json:"phase"`
      Status string `json:"status"`
   }

   // 把边缘消息结构体进行解析。
   err := json.Unmarshal(content, &jobStatus)
   ...
   cond := sednav1.LLJobCondition{
      Status:             v1.ConditionTrue,
      LastHeartbeatTime:  metav1.Now(),
      LastTransitionTime: metav1.Now(),
      Data:               string(condDataBytes),
      Message:            "reported by lc",
   }
```

```
   // 根据不同的边缘节点任务状态实现，变更当前LifelongLearningJob的整体状态
   switch strings.ToLower(jobStatus.Status) {
   case "ready":
      cond.Type = sednav1.LLJobStageCondReady
   case "completed":
      cond.Type = sednav1.LLJobStageCondCompleted
   case "failed":
      cond.Type = sednav1.LLJobStageCondFailed
   case "waiting":
      cond.Type = sednav1.LLJobStageCondWaiting
   default:
      return fmt.Errorf("invalid condition type: %v", jobStatus.Status)
   }

   // 将当前LifelongLearningJob的整体状态写回k8s，也就是LifelongLearningJob这个CR的
Status字段。
   err = c.appendStatusCondition(name, namespace, cond)
   ...
}
```

【9】Controller核心处理逻辑

```
// Run starts the main goroutine responsible for watching and syncing jobs.
func (c *Controller) Run(stopCh <-chan struct{}) {
   workers := 1

   defer utilruntime.HandleCrash()
   defer c.queue.ShutDown()

   klog.Infof("Starting %s controller", Name)
   defer klog.Infof("Shutting down %s controller", Name)

   if !cache.WaitForNamedCacheSync(Name, stopCh, c.podStoreSynced, c.jobStoreSynced)
{
      klog.Errorf("failed to wait for %s caches to sync", Name)

      return
   }
   klog.Infof("Starting %s workers", Name)
   for i := 0; i < workers; i++ {
      go wait.Until(c.worker, time.Second, stopCh)
   }

   <-stopCh
}
```

先会通过WaitForNamedCacheSync去等待Pod和LifelongLearningJob资源对象是否已经同步到Informer中。

如果已经同步，则会启动指定数量的worker对LifelongLearningJob进行处理。

【9】Controller核心处理逻辑

```
func (c *Controller) sync(key string) (bool, error) {
   //省略了部分代码
   ns, name, err := cache.SplitMetaNamespaceKey(key)

   sharedJob, err := c.jobLister.LifelongLearningJobs(ns).Get(name)

   // if job was finished previously, we don't want to redo the termination
   if IsJobFinished(&job) {
      return true, nil
   }

   // transit this job's state machine
   needUpdated, err = c.transitJobState(&job)


   if needUpdated {
      if err := c.updateJobStatus(&job); err != nil {
         return forget, err
      }

      if jobFailed && !IsJobFinished(&job) {
         // returning an error will re-enqueue LifelongLearningJob after the backoff period
         return forget, fmt.Errorf("failed pod(s) detected for lifelonglearningjob key %q", key)
      }

      forget = true
   }

   return forget, err
}
```

sync是对具体的LifelongLearningJob进行逻辑处理了，主要做了这么几件事：

通过SplitMetaNamespaceKey将LifelongLearningJob的key切分为namespace和name。

通过c.jobLister获取LifelongLearningJob的资源对象。

通过transitJobState来分析当前job应该进入到训练、评估、部署阶段了。

如果LifelongLearningJob的Status更新了，那么需要通过c.updateJobStatus()写回k8s资源对象中，这样通过kubectl查询到的就是最新的状态了，比如说当前在评估阶段、生成的模型路径在哪里等信息。

任务失败等异常处理。

第四部分

实践案例

中国电信研究院园区ReID案例

场景描述

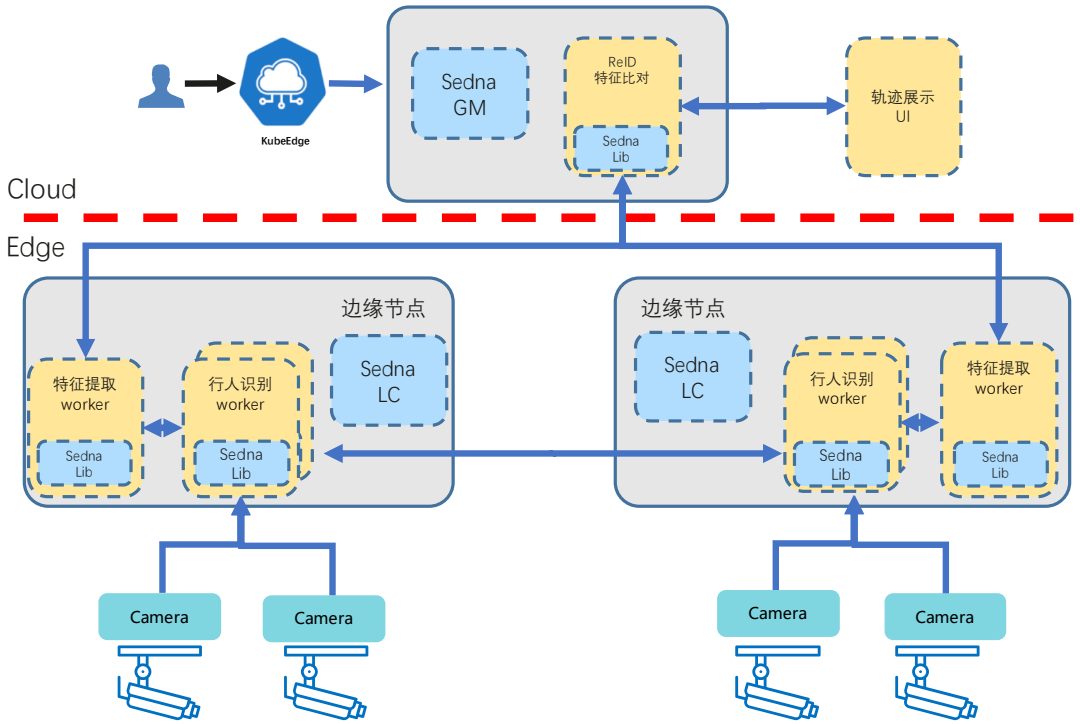
在给定视频中的第一帧和目标位置，实现目标检测、跟踪并预测其轨迹。

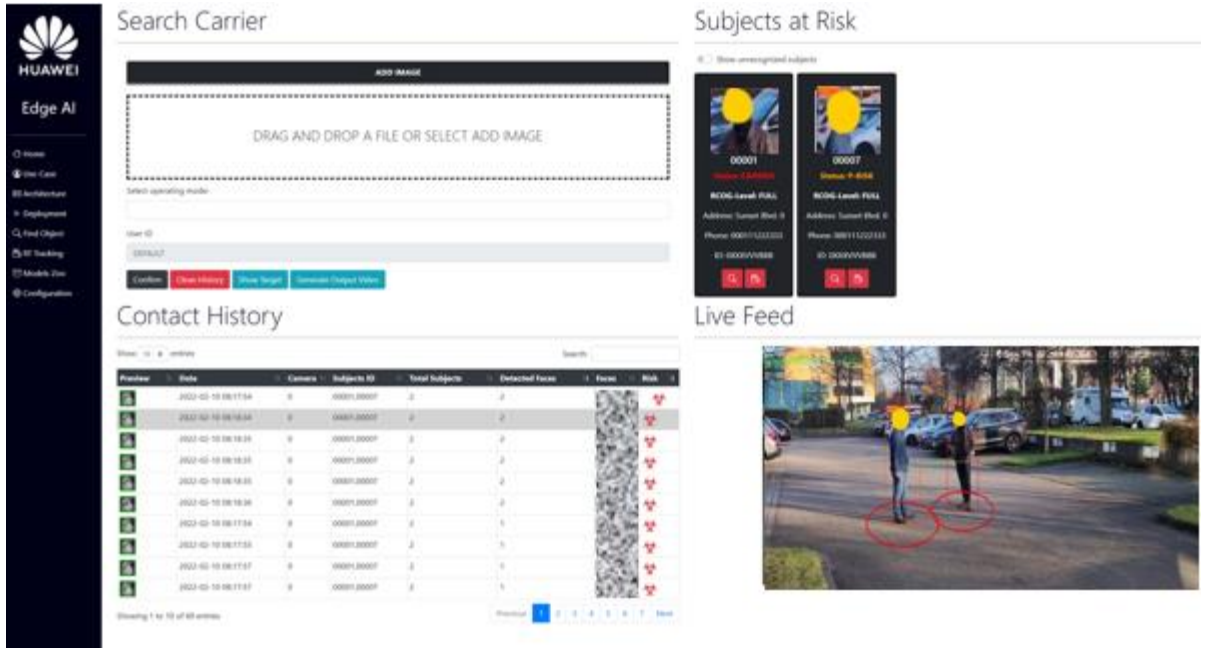
技术挑战

跨摄像头数据异构：摄像头拍摄的场景复杂，包括光照变化、遮挡严重、追踪目标数量多等，单点的目标跟踪算法无法得到较好的效果。

业务收益

- 多目标跟踪准确度（MOTA）从70.6%提升到87.0%
- 推理时延从21.2ms降低到18.5ms





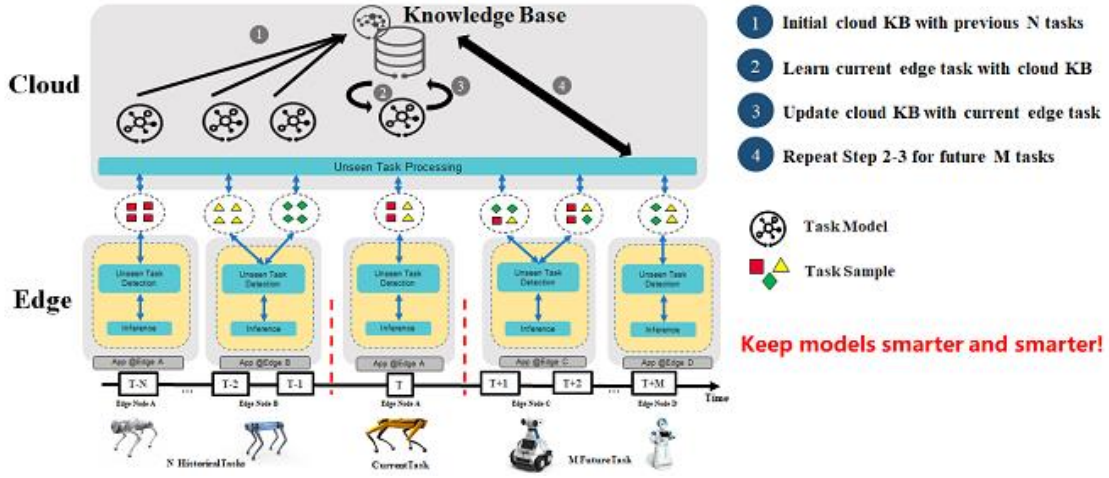
云机器人园区室外递送终身学习案例

场景描述

- 部署Sedna终身学习，实现机器人室外递送过程中的边侧智能环境感知任务；
- 基于视觉的语义分割，帮助机器人识别出低矮障碍，比如马路牙子和斜坡，帮助机器人做出正确的避让决策。

技术挑战

- **机器人本体资源不足。**比如无法部署GPU，导致推理速度慢，机器人来不及做避让决策；
- **数据异构问题导致AI模型失效。**比如，AI模型无法很好地识别到新环境的图片；
- **边侧数据不足，AI模型训练难以收敛或启动。**



业务收益：

- 未知任务训练提升E1-1F昏暗桥底corner case的mIoU指标达**1.78倍**；
- 启用未知任务训练帮助配送时间从**27min38s**减少到**19min53s**，节省幅度达**28%；**
- 未知任务识别模块上传未知样本，帮助减少**253张**训练样本打标量，节省幅度达**26%。**

Summary

- 了解Edge AI相关现状和趋势。
- 了解边云协同AI框架的特性和功能。
- 了解边云协同AI框架Sedna，以及背后Operator是如何打造的。
- 了解边云协同AI框架应用案例。

Thank You!