
Sedna

Release 0.4.1

Kubeedge

Jun 29, 2023

GUIDE

1	Guide	3
2	Quick Start	5
3	Prerequisites	11
4	Deploy All In One Sedna	13
5	Deploy Local Sedna Cluster	15
6	Edge Cloud Collaborative AI Framework	17
7	Dataset and Model	21
8	Federated Learning	31
9	Incremental Learning	41
10	Joint Inference	55
11	Object Search Service	65
12	Object Tracking Service	75
13	Using Joint Inference Service in Helmet Detection Scenario	85
14	Using Incremental Learning Job in Helmet Detection Scenario	91
15	Using Federated Learning Job in Surface Defect Detection Scenario	101
16	Collaboratively Train Yolo-v5 Using MistNet on COCO128 Dataset	107
17	Using Lifelong Learning Job in Thermal Comfort Prediction Scenario	113
18	Python API Use Guide	117
19	lib.sedna	123
20	1. Install Tools	231
21	2. Clone the code	233
22	3. Set up Kubernetes/KubeEdge(optional)	235

23	4. What's Next?	237
24	Roadmap	239
25	RELATED LINKS	241
26	Indices and tables	243
	Python Module Index	245
	Index	249



Sedna is an edge-cloud synergy AI project incubated in KubeEdge SIG AI. Benefiting from the edge-cloud synergy capabilities provided by KubeEdge, Sedna can implement across edge-cloud collaborative training and collaborative inference capabilities, such as joint inference, incremental learning, federated learning, and lifelong learning. Sedna supports popular AI frameworks, such as TensorFlow, Pytorch, PaddlePaddle, MindSpore.

Sedna can simply enable edge-cloud synergy capabilities to existing training and inference scripts, bringing the benefits of reducing costs, improving model performance, and protecting data privacy.

GUIDE

- If you are new to Sedna, you can try the command step by step in [quick start](#).
- If you have played the above example, you can find more [examples](#).
- If you want to know more about sedna's architecture and component, you can find them in [sedna home](#).
- If you're looking to contribute documentation improvements, you'll specifically want to see the [kubernetes documentation style guide](#) before [filing an issue](#).
- If you're planning to contribute code changes, you'll want to read the [development preparation guide](#) next.
- If you're planning to add a new synergy feature directly, you'll want to read the [guide](#) next.

QUICK START

The following is showing how to run a joint inference job by sedna.

2.1 Quick Start

2.1.1 0. Check the Environment

For Sedna all-in-one installation, it requires you:

- 1 VM (**one machine is OK, cluster is not required**)
- 2 CPUs or more
- 2GB+ free memory, depends on node number setting
- 10GB+ free disk space
- Internet connection(docker hub, github etc.)
- Linux platform, such as ubuntu/centos
- Docker 17.06+

you can check the docker version by the following command,

```
docker -v
```

after doing that, the output will be like this, that means your version fits the bill.

```
Docker version 19.03.6, build 369ce74a3c
```

2.1.2 1. Deploy Sedna

Sedna provides three deployment methods, which can be selected according to your actual situation:

- [Install Sedna AllinOne](#). (used for development, here we use it)
- [Install Sedna local up](#).
- [Install Sedna on a cluster](#).

The [all-in-one script](#) is used to install Sedna along with a mini Kubernetes environment locally, including:

- A Kubernetes v1.21 cluster with multi worker nodes, default zero worker node.
- KubeEdge with multi edge nodes, default is latest KubeEdge and one edge node.

- Sedna, default is the latest version.

```
curl https://raw.githubusercontent.com/kubeedge/sedna/master/scripts/installation/
↪all-in-one.sh | NUM_EDGE_NODES=1 bash -
```

Then you get two nodes `sedna-mini-control-plane` and `sedna-mini-edge0`, you can get into each node by following command:

```
# get into cloud node
docker exec -it sedna-mini-control-plane bash
```

```
# get into edge node
docker exec -it sedna-mini-edge0 bash
```

2.1.3 1. Prepare Data and Model File

- step1: download `little model` to your edge node.

```
mkdir -p /data/little-model
cd /data/little-model
wget https://kubeedge.obs.cn-north-1.myhuaweicloud.com/examples/helmet-detection-
↪inference/little-model.tar.gz
tar -zxvf little-model.tar.gz
```

- step2: download `big model` to your cloud node.

```
mkdir -p /data/big-model
cd /data/big-model
wget https://kubeedge.obs.cn-north-1.myhuaweicloud.com/examples/helmet-detection-
↪inference/big-model.tar.gz
tar -zxvf big-model.tar.gz
```

2.1.4 2. Create Big Model Resource Object for Cloud

In cloud node:

```
kubectl create -f - <<EOF
apiVersion: sedna.io/v1alpha1
kind: Model
metadata:
  name: helmet-detection-inference-big-model
  namespace: default
spec:
  url: "/data/big-model/yolov3_darknet.pb"
  format: "pb"
EOF
```

2.1.5 3. Create Little Model Resource Object for Edge

In cloud node:

```
kubectl create -f - <<EOF
apiVersion: sedna.io/v1alpha1
kind: Model
metadata:
  name: helmet-detection-inference-little-model
  namespace: default
spec:
  url: "/data/little-model/yolov3_resnet18.pb"
  format: "pb"
EOF
```

2.1.6 4. Create JointInferenceService

Note the setting of the following parameters, which have to same as the script `little_model.py`:

- `hardExampleMining`: set hard example algorithm from {IBT, CrossEntropy} for inferring in edge side.
- `video_url`: set the url for video streaming.
- `all_examples_inference_output`: set your output path for the inference results.
- `hard_example_edge_inference_output`: set your output path for results of inferring hard examples in edge side.
- `hard_example_cloud_inference_output`: set your output path for results of inferring hard examples in cloud side.

Make preparation in edge node

```
mkdir -p /joint_inference/output
```

Create joint inference service

```
CLOUD_NODE="sedna-mini-control-plane"
EDGE_NODE="sedna-mini-edge0"

kubectl create -f https://raw.githubusercontent.com/jaypume/sedna/main/examples/joint_
↪ inference/helmet_detection_inference/helmet_detection_inference.yaml
```

2.1.7 5. Check Joint Inference Status

```
kubectl get jointinferenceservices.sedna.io
```

2.1.8 6. Mock Video Stream for Inference in Edge Side

- step1: install the open source video streaming server [EasyDarwin](#).
- step2: start EasyDarwin server.
- step3: download [video](#).
- step4: push a video stream to the url (e.g., `rtsp://localhost/video`) that the inference service can connect.

```
wget https://github.com/EasyDarwin/EasyDarwin/releases/download/v8.1.0/EasyDarwin-linux-
↪8.1.0-1901141151.tar.gz
tar -zxvf EasyDarwin-linux-8.1.0-1901141151.tar.gz
cd EasyDarwin-linux-8.1.0-1901141151
./start.sh

mkdir -p /data/video
cd /data/video
wget https://kubedge.obs.cn-north-1.myhuaweicloud.com/examples/helmet-detection-
↪inference/video.tar.gz
tar -zxvf video.tar.gz

ffmpeg -re -i /data/video/video.mp4 -vcodec libx264 -f rtsp rtsp://localhost/video
```

Check Inference Result

You can check the inference results in the output path (e.g. `/joint_inference/output`) defined in the `JointInferenceService` config.

- the result of edge inference vs the result of joint inference

`../images/inference-result.png`



2.2 API

- control-plane: Please refer to this [link](#).
- Lib: Please refer to this [link](#).

2.3 Contributing

Contributions are very welcome!

- control-plane: Please refer to this [link](#).
- Lib: Please refer to this [link](#).

2.4 Community

Sedna is an open source project and in the spirit of openness and freedom, we welcome new contributors to join us. You can get in touch with the community according to the ways:

- [Github Issues](#)
- [Regular Community Meeting](#)
- [slack channel](#)

This guide covers how to install Sedna on an existing Kubernetes environment.

For interested readers, Sedna also has two important components that would be mentioned below, i.e., [GM\(GlobalManager\)](#) and [LC\(LocalController\)](#) for workload generation and maintenance.

If you don't have an existing Kubernetes, you can: 1) Install Kubernetes by following the [Kubernetes website](#). 2) Or follow [quick start](#) for other options.

PREREQUISITES

- [Kubectl](#) with right kubeconfig
- [Kubernetes](#) 1.16+ cluster running
- [KubeEdge](#) v1.8+ along with **EdgeMesh** running

3.1 Deploy Sedna

Currently GM is deployed as a [deployment](#), and LC is deployed as a [daemonset](#).

Run the one liner:

```
curl https://raw.githubusercontent.com/kubeedge/sedna/main/scripts/installation/install.  
↪ sh | SEDNA_ACTION=create bash -
```

It requires the network to access github since it will download the sedna [crd yamls](#). If you have unstable network to access github or existing sedna source, you can try the way:

```
# SEDNA_ROOT is the sedna git source directory or cached directory  
export SEDNA_ROOT=/opt/sedna  
curl https://raw.githubusercontent.com/kubeedge/sedna/main/scripts/installation/install.  
↪ sh | SEDNA_ACTION=create bash -
```

3.2 Debug

1. Check the GM status:

```
kubectl get deploy -n sedna gm
```

2. Check the LC status:

```
kubectl get ds lc -n sedna
```

3. Check the pod status:

```
kubectl get pod -n sedna
```

3.3 Uninstall Sedna

```
curl https://raw.githubusercontent.com/kubeedge/sedna/main/scripts/installation/install.  
↪ sh | SEDNA_ACTION=delete bash -
```


DEPLOY ALL IN ONE SEDNA

The [all-in-one script](#) is used to install Sedna along with a mini Kubernetes environment locally, including:

- A Kubernetes v1.21 cluster with multi worker nodes, default zero worker node.
- KubeEdge with multi edge nodes, default is latest KubeEdge and one edge node.
- Sedna, default is the latest version.

It requires you:

- 2 CPUs or more
- 2GB+ free memory, depends on node number setting
- 10GB+ free disk space
- Internet connection(docker hub, github etc.)
- Linux platform, such as ubuntu/centos
- Docker 17.06+

For example:

```
curl https://raw.githubusercontent.com/kubeedge/sedna/master/scripts/installation/all-in-one.sh | KUBEEDGE_VERSION=v1.8.0 NUM_EDGE_NODES=2 bash -
```

Above command installs a mini Sedna environment, including:

- A Kubernetes v1.21 cluster with only one master node.
- KubeEdge with two edge nodes.
- The latest Sedna.

You can play it online on [katacoda](#).

Advanced options: | Env Variable | Description| Default Value| — | — | — | **[NUM_EDGE_NODES | Number of KubeEdge nodes] 1 | [NUM_CLOUD_WORKER_NODES | Number of cloud ****worker**** nodes, not master node] 0 | [SEDNA_VERSION | The Sedna version to be installed. [The latest version] [KUBEEDGE_VERSION | The KubeEdge version to be installed. [The latest version] [CLUSTER_NAME | The all-in-one cluster name] sedna-mini| [FORCE_INSTALL_SEDNA | If 'true', force to reinstall Sedna|false] [NODE_IMAGE | Custom node image] kubeedge/sedna-allinone-node:v1.21.1| [REUSE_EDGE_CONTAINER | Whether reuse edge node containers or not|true]**

Clean all-in-one Sedna:

```
curl https://raw.githubusercontent.com/kubeedge/sedna/main/scripts/installation/all-in-one.sh | bash /dev/stdin clean
```


DEPLOY LOCAL SEDNA CLUSTER

The `local-up` script boots a local Kubernetes cluster, installs latest KubeEdge, and deploys Sedna based on the Sedna local repository.

5.1 Use Case

When one is contributing new features for Sedna, codes like AI algorithms under testing can be frequently changed before final deployment. When coding in that case, s/he would suffer from tortured re-installations and frequent failures of the whole complicated system. To get rid of the torments, one can use the local-up installation, embraced the single-machine simulation for agiler development and testing.

5.2 Setup

It requires:

- 2 CPUs or more
- 1GB+ free memory
- 5GB+ free disk space
- Internet connection(docker hub, github etc.)
- Linux platform, such as ubuntu/centos
- Docker 17.06+
- A local Sedna code repository

Then you can enter Sedna local code repository, and create a local Sedna cluster with:

```
bash hack/local-up.sh
```

In more details, this local-up script uses `kind` to create a local K8S cluster with one master node, and joins the K8S cluster by running KubeEdge.

In another terminal, you can see them by using `kubectl get nodes -o wide`:

NAME	STATUS	ROLES	AGE	VERSION	
↪ INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	KERNEL-VERSION	CONTAINER-	
↪ RUNTIME					
edge-node	Ready	agent, edge	3d21h	v1.19.3-kubeedge-v1.6.1	↪
↪ 192.168.0.233	<none>	Ubuntu 18.04.5 LTS	4.15.0-128-generic	docker://20.10.	

(continues on next page)

(continued from previous page)

```
↪ 2
sedna-control-plane   Ready    control-plane,master   3d21h   v1.20.2
↪ 172.18.0.2          <none>    Ubuntu 20.10           4.15.0-128-generic    containerd://1.
↪ 5.0-beta.3-24-g95513021e
```

You can login the master node with:

```
docker exec -it --detach-keys=ctrl-@ sedna-control-plane bash
# since the master node just uses containerd CRI runtime, you can alias the CRI cli
↪ 'crictl' as 'docker'
alias docker=crictl
```

After you have done developing, built worker image and want to run your worker into master node, your worker image should be loaded into the cluster nodes with:

```
kind load docker-image --name sedna <your-custom-worker-image>
```

EDGE CLOUD COLLABORATIVE AI FRAMEWORK

6.1 Motivation

Currently, “Edge AI” in the industry is at an early stage of training on the cloud and inference on the edge. However, the future trend has emerged, and related research and practice are booming, bringing new value growth points for edge computing and AI. Also, edge AI applications have much room for optimization in terms of cost, model effect, and privacy protection. For example:

This proposal provides a basic framework for edge-cloud collaborative training and inference, so that AI applications running at the edge can benefit from cost reduction, model performance improvement, and data privacy protection.

6.1.1 Goals

For AI applications running at the edge, the goals of edge cloud collaborative framework are:

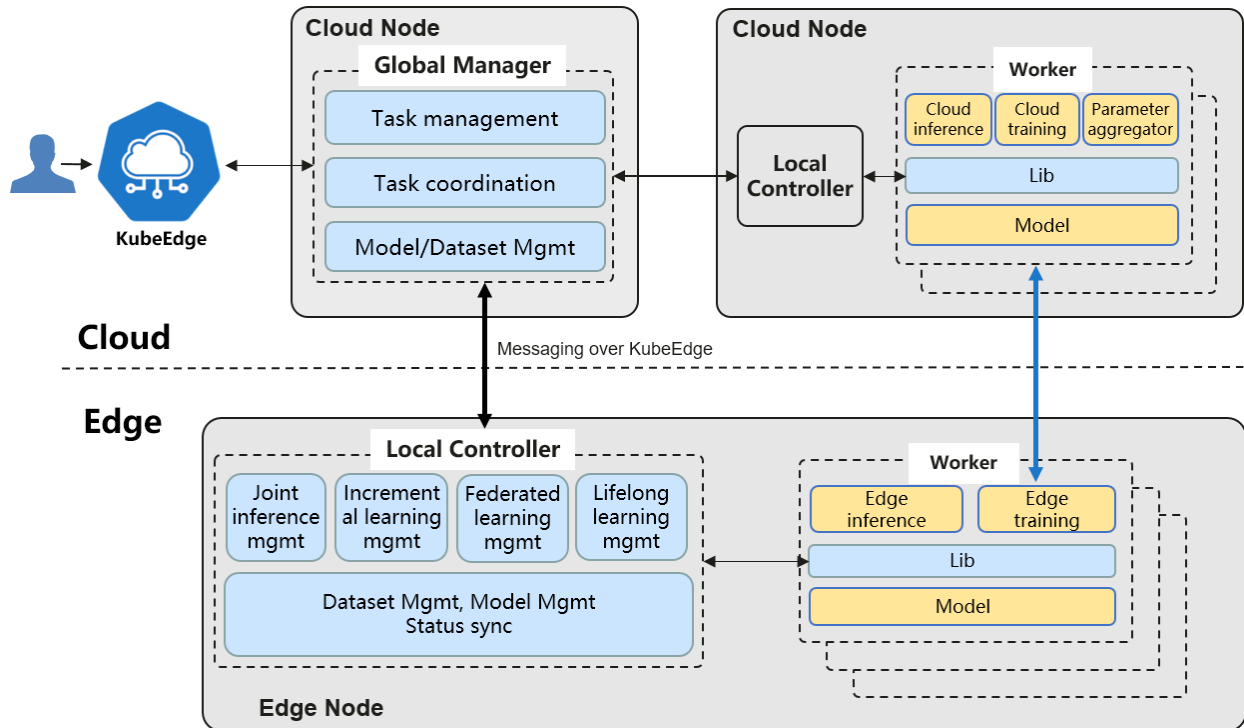
- reducing resource cost on the edge
- improving model performance
- protecting data privacy

6.2 Proposal

- What we propose:
 - an edge-cloud collaborative AI framework based on KubeEdge
 - with embed collaborative training and joint inferencing algorithm
 - working with existing AI framework like Tensorflow, etc
- 3 Features
 - joint inference
 - incremental learning
 - federated learning
- Targeting Users
 - Domain-specific AI Developers: build and publish edge-cloud collaborative AI services/functions easily
 - Application Developers: use edge-cloud collaborative AI capabilities.
- We are NOT:

- to re-invent existing ML framework, i.e., tensorflow, pytorch, mindspore, etc.
- to re-invent existing edge platform, i.e., kubeedge, etc.
- to offer domain/application-specific algorithms, i.e., facial recognition, text classification, etc.

6.2.1 Architecture



- **GlobalManager:** implements the Edge AI features controllers based on the [k8s operator pattern](#)
 - Federated Learning Controller: Implements the federated learning feature based on user created CRDs
 - Incremental Learning Controller: Implements the incremental learning feature based on user created CRDs
 - Joint Inference Controller: Implements the joint inference feature based on user created CRDs
- **LocalController:** manages the Edge AI features, the extra dataset/model resources on the edge nodes
- **Workers:** includes the training/evaluation/inference/aggregator
 - do inference or training, based on existing ML framework
 - launch on demand, imagine they are docker containers
 - different workers for different features
 - could run on edge or cloud
- **Lib:** exposes the Edge AI features to applications, i.e. training or inference programs
- **Dataset and Model**
 - **Motivation**
 - * **Goals**
 - * **Non-goals**

- Proposal
 - * Use Cases
- Design Details
 - * CRD API Group and Version
 - * CRDs
 - * Type definition
 - * Crd sample
- Controller Design

DATASET AND MODEL

7.1 Motivation

Currently, the Edge AI features depend on the object `dataset` and `model`.

This proposal provides the definitions of `dataset` and `model` as the first class of k8s resources.

7.1.1 Goals

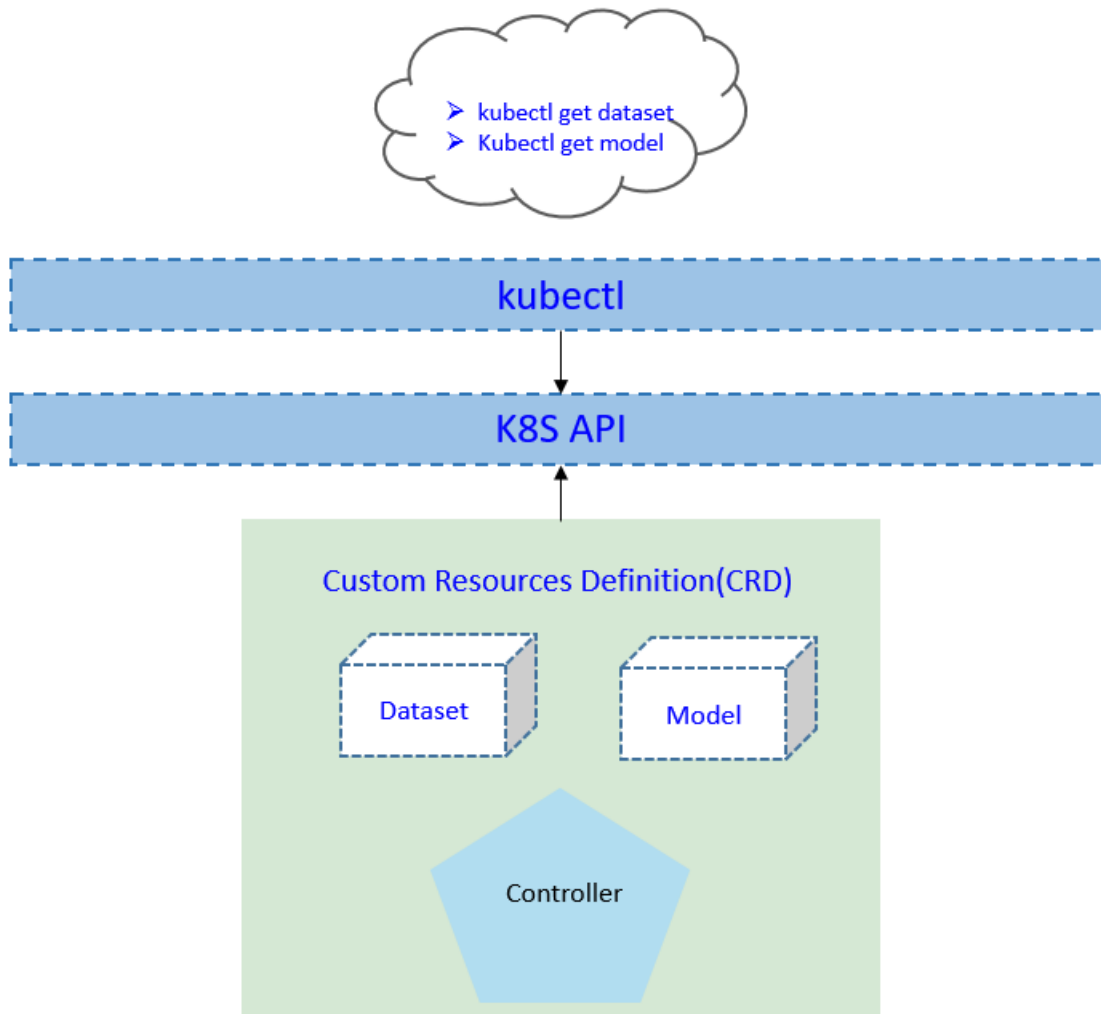
- Metadata of `dataset` and `model` objects.
- Used by the Edge AI features

7.1.2 Non-goals

- The truly format of the AI `dataset`, such as `imagenet`, `coco` or `tf-record` etc.
- The truly format of the AI `model`, such as `ckpt`, `saved_model` of tensorflow etc.
- The truly operations of the AI `dataset`, such as `shuffle`, `crop` etc.
- The truly operations of the AI `model`, such as `train`, `inference` etc.

7.2 Proposal

We propose using Kubernetes Custom Resource Definitions (CRDs) to describe the `dataset/model` specification/status and a controller to synchronize these updates between edge and cloud.



7.2.1 Use Cases

- Users can create the dataset resource, by providing the `dataset url`, `format` and the `nodeName` which owns the dataset.
- Users can create the model resource by providing the `model url` and `format`.
- Users can show the information of dataset/model.
- Users can delete the dataset/model.

7.3 Design Details

7.3.1 CRD API Group and Version

The Dataset and Model CRDs will be namespace-scoped. The tables below summarize the group, kind and API version details for the CRDs.

- Dataset

Field	Description
Group	sedna.io
APIVersion	v1alpha1
Kind	Dataset

- Model

Field	Description
Group	sedna.io
APIVersion	v1alpha1
Kind	Model

7.3.2 CRDs

Dataset CRD

crd source

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: datasets.sedna.io
spec:
  group: sedna.io
  names:
    kind: Dataset
    plural: datasets
    scope: Namespaced
  versions:
    - name: v1alpha1
      subresources:
        # status enables the status subresource.
        status: {}
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              required:
```

(continues on next page)

(continued from previous page)

```

    - url
    - format
  properties:
    url:
      type: string
    format:
      type: string
    nodeName:
      type: string
  status:
    type: object
    properties:
      numberOfSamples:
        type: integer
      updateTime:
        type: string
        format: datetime

  additionalPrinterColumns:
    - name: NumberOfSamples
      type: integer
      description: The number of samples in the dataset
      jsonPath: ".status.numberOfSamples"
    - name: Node
      type: string
      description: The node name of the dataset
      jsonPath: ".spec.nodeName"
    - name: spec
      type: string
      description: The spec of the dataset
      jsonPath: ".spec"

```

1. format of dataset

We use this field to report the number of samples for the dataset and do dataset splitting.

Current we support these below formats:

- txt: one nonempty line is one sample

Model CRD

crd source

```

apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: models.sedna.io
spec:
  group: sedna.io
  names:
    kind: Model

```

(continues on next page)

(continued from previous page)

```

plural: models
scope: Namespaced
versions:
  - name: v1alpha1
    subresources:
      # status enables the status subresource.
      status: {}
    served: true
    storage: true
    schema:
      openAPIV3Schema:
        type: object
        properties:
          spec:
            type: object
            required:
              - url
              - format
            properties:
              url:
                type: string
              format:
                type: string
          status:
            type: object
            properties:
              updateTime:
                type: string
                format: datetime
          metrics:
            type: array
            items:
              type: object
              properties:
                key:
                  type: string
                value:
                  type: string

additionalPrinterColumns:
  - name: updateAGE
    type: date
    description: The update age
    jsonPath: ".status.updateTime"
  - name: metrics
    type: string
    description: The metrics
    jsonPath: ".status.metrics"

```

7.3.3 CRD type definition

- Dataset

go source

```
package v1alpha1

import (
    metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
)

// +genclient
// +k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object

// Dataset describes the data that a dataset resource should have
type Dataset struct {
    metav1.TypeMeta `json:",inline"`

    metav1.ObjectMeta `json:"metadata,omitempty"`

    Spec   DatasetSpec   `json:"spec"`
    Status DatasetStatus `json:"status"`
}

// DatasetSpec is a description of a dataset
type DatasetSpec struct {
    URL      string `json:"url"`
    Format   string `json:"format"`
    NodeName string `json:"nodeName"`
}

// DatasetStatus represents information about the status of a dataset
// including the time a dataset updated, and number of samples in a dataset
type DatasetStatus struct {
    UpdateTime      *metav1.Time `json:"updateTime,omitempty" protobuf:"bytes,1,opt,
↪name=updateTime"`
    NumberOfSamples int           `json:"numberOfSamples"`
}

// +k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object

// DatasetList is a list of Datasets
type DatasetList struct {
    metav1.TypeMeta `json:",inline"`
    metav1.ListMeta `json:"metadata"`

    Items []Dataset `json:"items"`
}
```

- Model

go source

```

package v1alpha1

import (
    metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
)

// +genclient
// +k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object

// Model describes the data that a model resource should have
type Model struct {
    metav1.TypeMeta `json:",inline"`

    metav1.ObjectMeta `json:"metadata,omitempty"`

    Spec   ModelSpec   `json:"spec"`
    Status ModelStatus `json:"status"`
}

// ModelSpec is a description of a model
type ModelSpec struct {
    URL string `json:"url"`
    Format string `json:"format"`
}

// ModelStatus represents information about the status of a model
// including the time a model updated, and metrics in a model
type ModelStatus struct {
    UpdateTime *metav1.Time `json:"updateTime,omitempty" protobuf:"bytes,1,opt,
↪name=updateTime"`
    Metrics []Metric `json:"metrics,omitempty" protobuf:"bytes,2,rep,name=metrics
↪"`
}

// +k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object

// ModelList is a list of Models
type ModelList struct {
    metav1.TypeMeta `json:",inline"`
    metav1.ListMeta `json:"metadata"`

    Items []Model `json:"items"`
}

```

7.3.4 Crd samples

- Dataset

```
apiVersion: sedna.io/v1alpha1
kind: Dataset
metadata:
  name: "dataset-examp"
spec:
  url: "/code/data"
  format: "txt"
  nodeName: "edge0"
```

- Model

```
apiVersion: sedna.io/v1alpha1
kind: Model
metadata:
  name: model-examp
spec:
  url: "/model/frozen.pb"
  format: pb
```

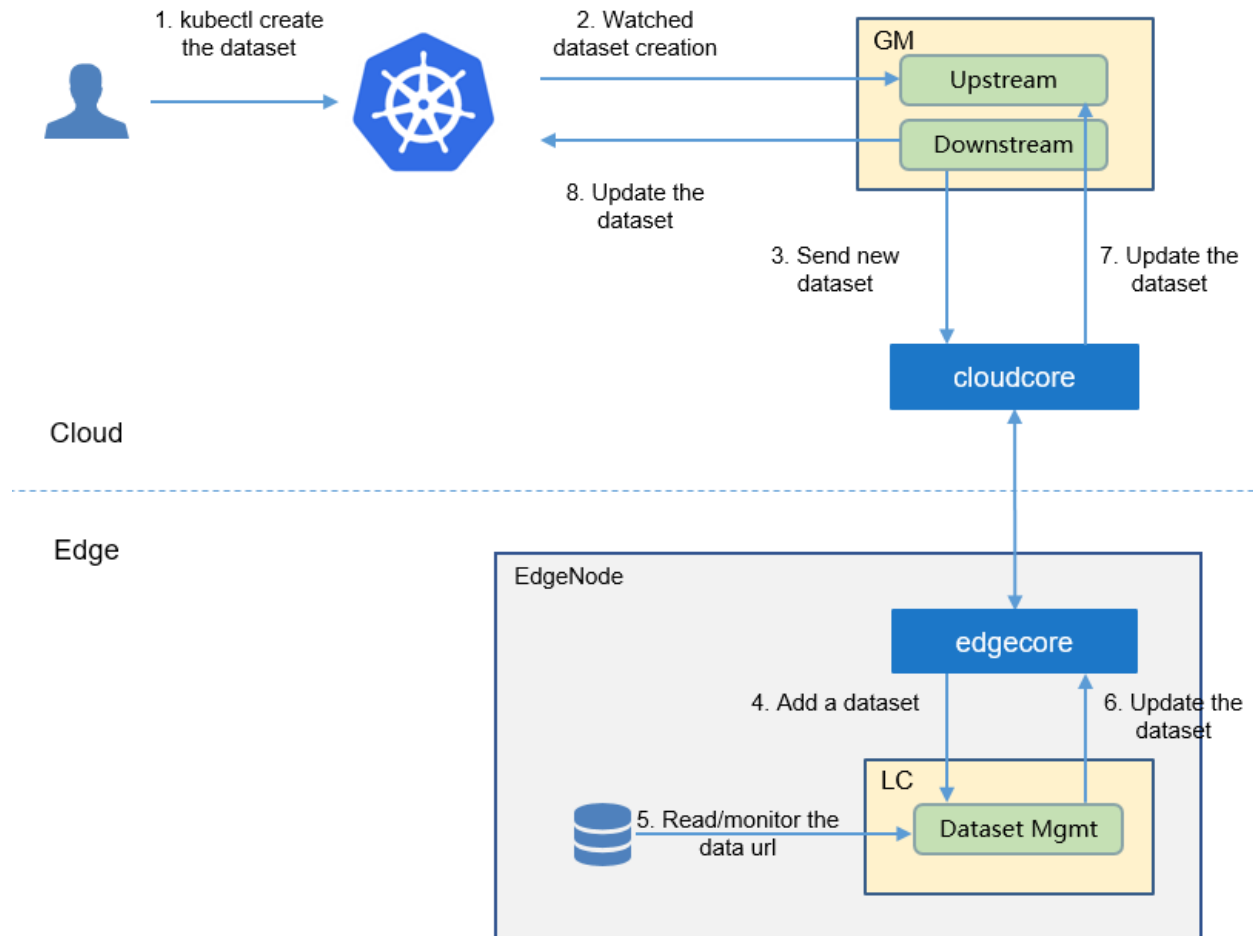
7.4 Controller Design

In the current design there is downstream/upstream controller for `dataset`, no downstream/upstream controller for `model`.

The dataset controller synchronizes the dataset between the cloud and edge.

- downstream: synchronize the dataset info from the cloud to the edge node.
- upstream: synchronize the dataset status from the edge to the cloud node, such as the information how many samples the dataset has.

Here is the flow of the dataset creation:



For the model:

1. Model's info will be synced when sync the federated-task etc which uses the model.
2. Model's status will be updated when the corresponding training/inference work has completed.

- Federated Learning

- Motivation

- * Goals

- * Non-goals

- Proposal

- * Use Cases

- Design Details

- * CRD API Group and Version

- * Federated learning CRD

- * Federated learning type definition

- * Federated learning sample

- * Validation

- Controller Design
 - * Federated Learning Controller
 - * Downstream Controller
 - * Upstream Controller
 - * Details of api between GM(cloud) and LC(edge)
- Workers Communication

FEDERATED LEARNING

8.1 Motivation

For edge AI, data is naturally generated at the edge. based on these assumptions:

- Users are unwilling to upload raw data to the cloud because of data privacy.
- Users do not want to purchase new devices for centralized training at the edge.
- The sample size at the edge is usually small, and it is often difficult to train a good model at a single edge node.

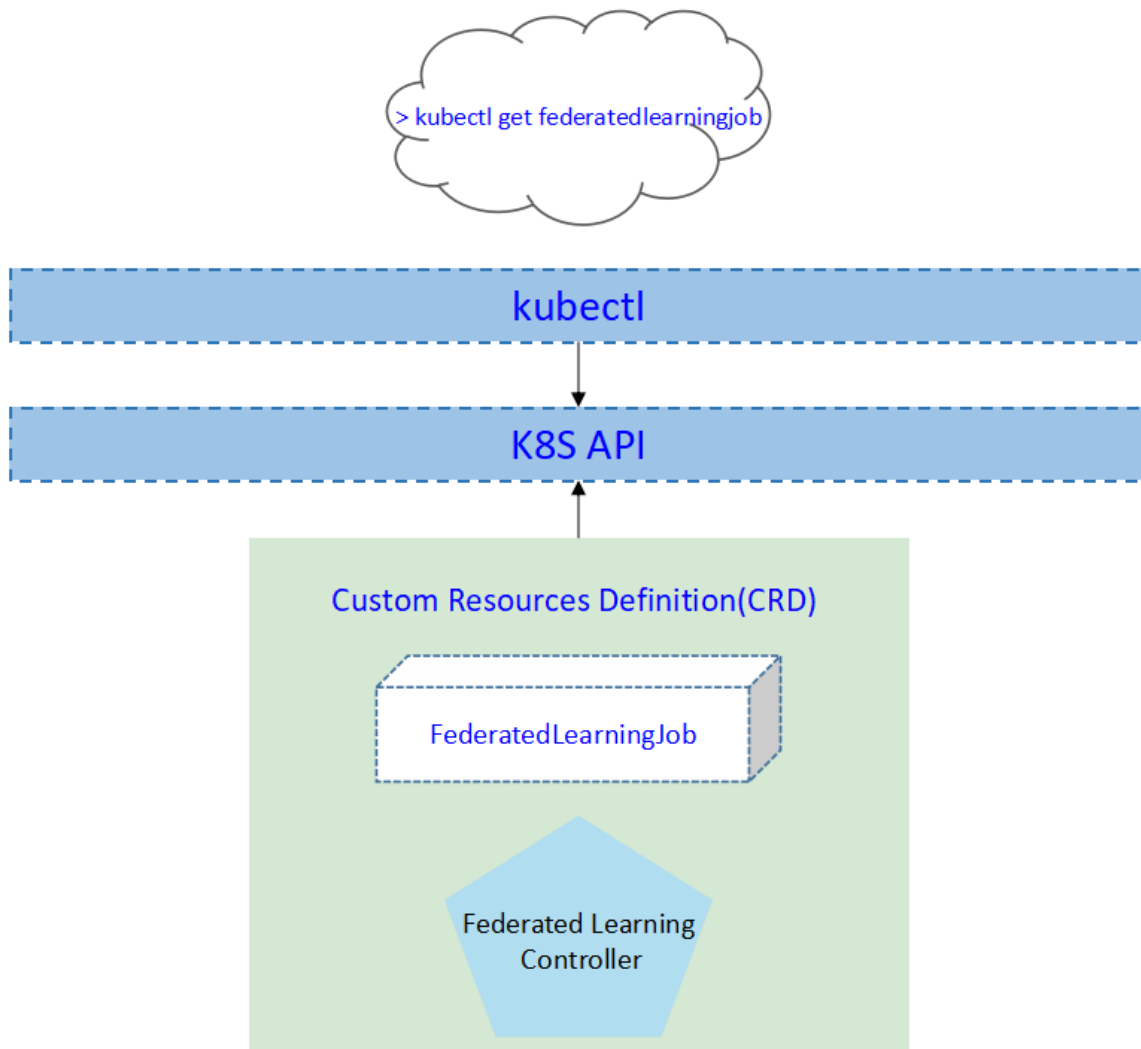
Therefore, we propose a edge cloud federated learning framework to help to train a model **without uploading raw data**, and **higher precision** and **less convergence time** are also benefits.

8.1.1 Goals

- The framework can combine data on multiple edge nodes to complete training.
- The framework provides the functions of querying the training status and result.
- The framework integrates some common aggregation algorithms, FedAvg and so on.
- The framework integrates some common weight/gradient compression algorithm to reduce the cloud-edge traffic required for aggregation operations.
- The framework integrates some common multi-job migration algorithms to resolve the problem of low precision caused by small size samples.

8.2 Proposal

We propose using Kubernetes Custom Resource Definitions (CRDs) to describe the federated learning specification/status and a controller to synchronize these updates between edge and cloud.



8.2.1 Use Cases

- User can create a federated learning job, with providing a training script, specifying the aggregation algorithm, configuring training hyperparameters, configuring training datasets.
- Users can get the federated learning status, including the nodes participating in training, current training status, samples size of each node, current iteration times, and current aggregation times.
- Users can get the saved aggregated model. The model file can be stored on the cloud or edge node.

8.3 Design Details

8.3.1 CRD API Group and Version

The `FederatedLearningJob` CRD will be namespace-scoped. The tables below summarize the group, kind and API version details for the CRD.

- `FederatedLearningJob`

Field	Description
Group	sedna.io
APIVersion	v1alpha1
Kind	FederatedLearningJob

8.3.2 Federated learning CRD

Below is the CustomResourceDefinition yaml for `FederatedLearningJob`: [crd source](#)

8.3.3 Federated learning type definition

[go source](#)

Validation

[Open API v3 Schema based validation](#) can be used to guard against bad requests. Invalid values for fields (example string value for a boolean field etc) can be validated using this.

Here is a list of validations we need to support :

1. The `dataset` specified in the crd should exist in k8s.
2. The `model` specified in the crd should exist in k8s.
3. The `edgenode` name specified in the crd should exist in k8s.

8.3.4 federated learning sample

see [sample source](#)

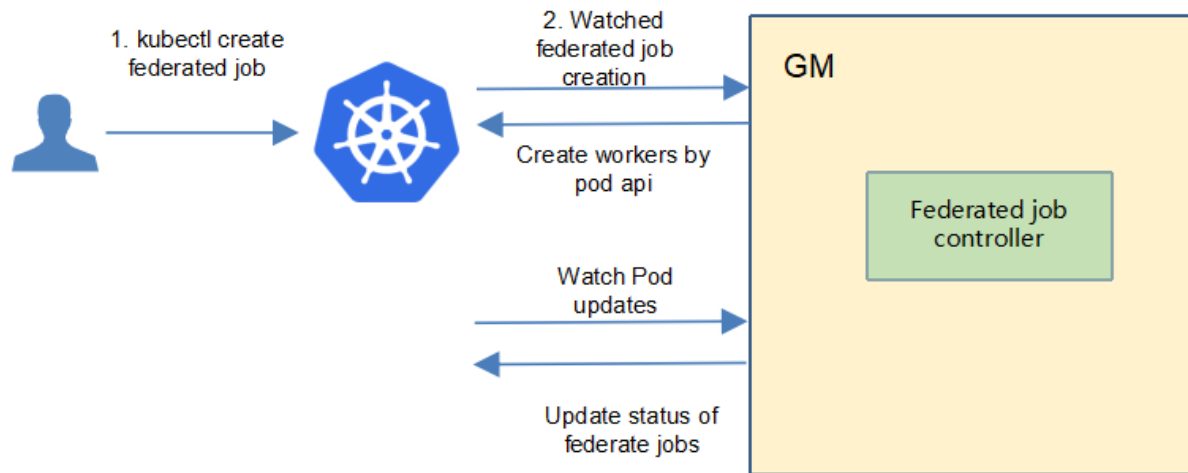
8.3.5 Creation of the federated learning job

8.4 Controller Design

The federated learning controller starts three separate goroutines called `upstream`, `downstream` and `federated-learningcontroller`. These are not separate controllers as such but named here for clarity.

- `federated learning`: watch the updates of `federated-learning-job` crds, and create the workers to complete the job.
- `downstream`: synchronize the federated-learning updates from the cloud to the edge node.
- `upstream`: synchronize the federated-learning updates from the edge to the cloud node.

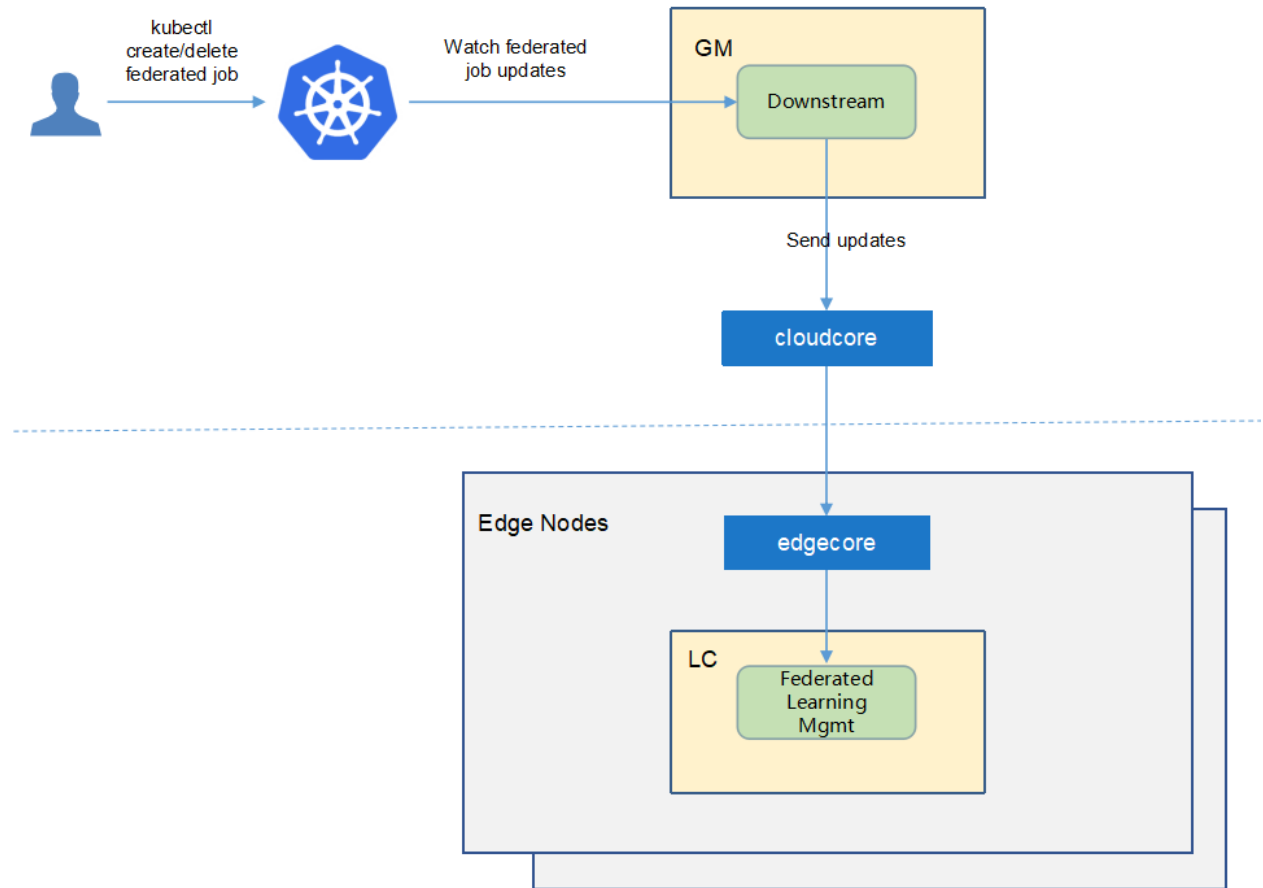
8.4.1 Federated Learning Controller



The federated-learning controller watches for the updates of federated-learning jobs and the corresponding pods against the K8S API server. Updates are categorized below along with the possible actions:

Update Type	Action
New Federated-learning-job Created	Create the aggregation worker and these local-training workers
Federated-learning-job Deleted	NA. These workers will be deleted by <code>k8s gc</code> .
The corresponding pod created/running/completed/failed	Update the status of federated-learning job.

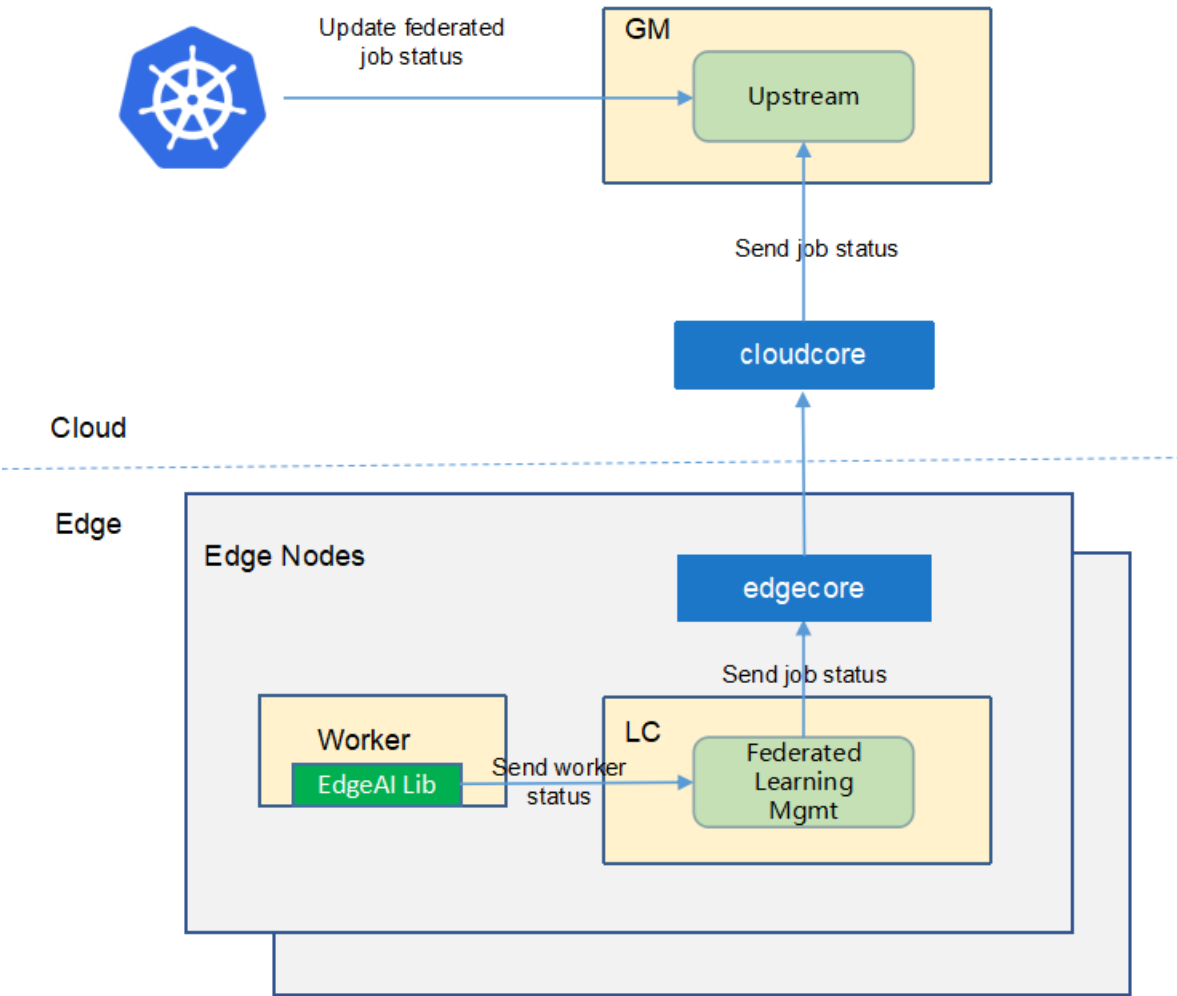
8.4.2 Downstream Controller



The downstream controller watches for federated-learning updates against the K8S API server. Updates are categorized below along with the possible actions that the downstream controller can take:

Update Type	Action
New Federated-learning-job Created	Sends the job information to LCs.
Federated-learning-job Deleted	The controller sends the delete event to LCs.

8.4.3 Upstream Controller



The upstream controller watches for federated-learning-job updates from the edge node and applies these updates against the API server in the cloud. Updates are categorized below along with the possible actions that the upstream controller can take:

Update Type	Action
Federated-learning-job Reported State Updated	The controller appends the reported status of the Federated-learning-job in the cloud.

8.4.4 Details of api between GM(cloud) and LC(edge)

1. GM(downstream controller) syncs the job info to LC:

```
// POST <namespace>/federatedlearningjobs/<job-name>
// body same to the job crd of k8s api, omitted here.
```

2. LC uploads the job status which reported by the worker to GM(upstream controller):

```
// POST <namespace>/federatedlearningjobs/<job-name>/status

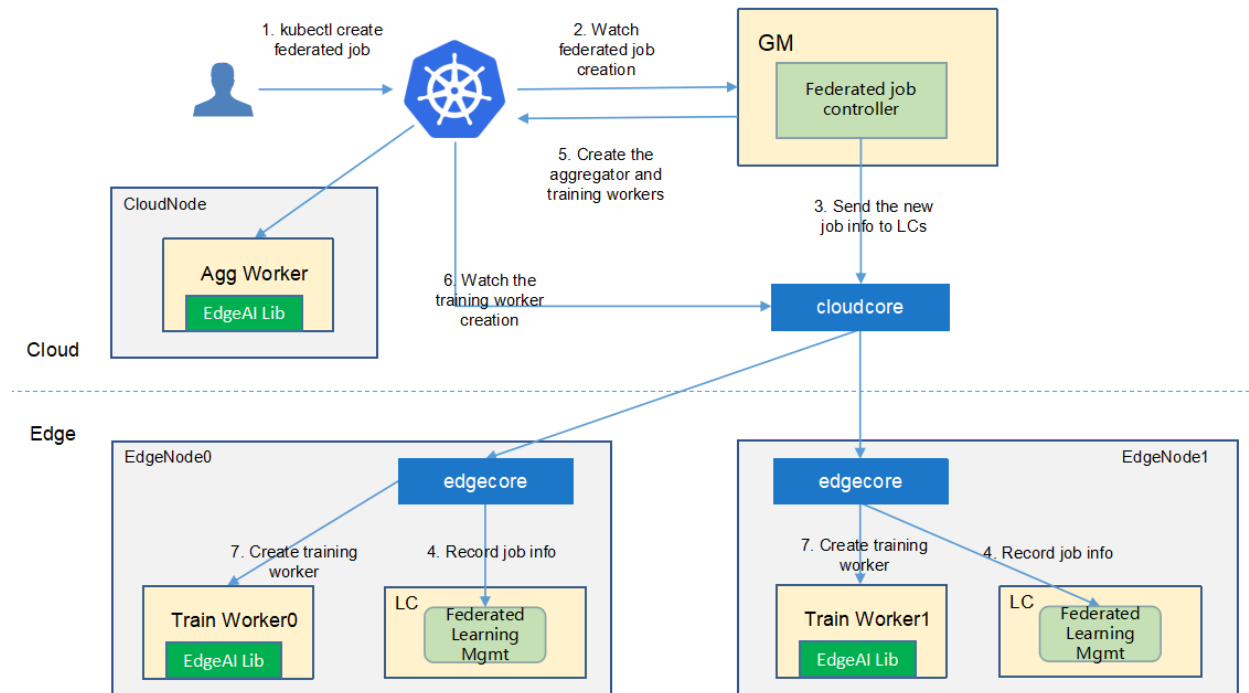
// WorkerMessage defines the message from that the training worker. It will send to GM.
↪GM.
type WorkerMessage struct {
    Phase string `json:"phase"`
    Status string `json:"status"`
    Output *WorkerOutput `json:"output"`
}

//
type WorkerOutput struct {
    Models []*Model `json:"models"`
    JobInfo *JobInfo `json:"jobInfo"`
}

// Model defines the model information
type Model struct {
    Format string `json:"format"`
    URL string `json:"url"`
    // Including the metrics, e.g. precision/recall
    Metrics map[string]float64 `json:"metrics"`
}

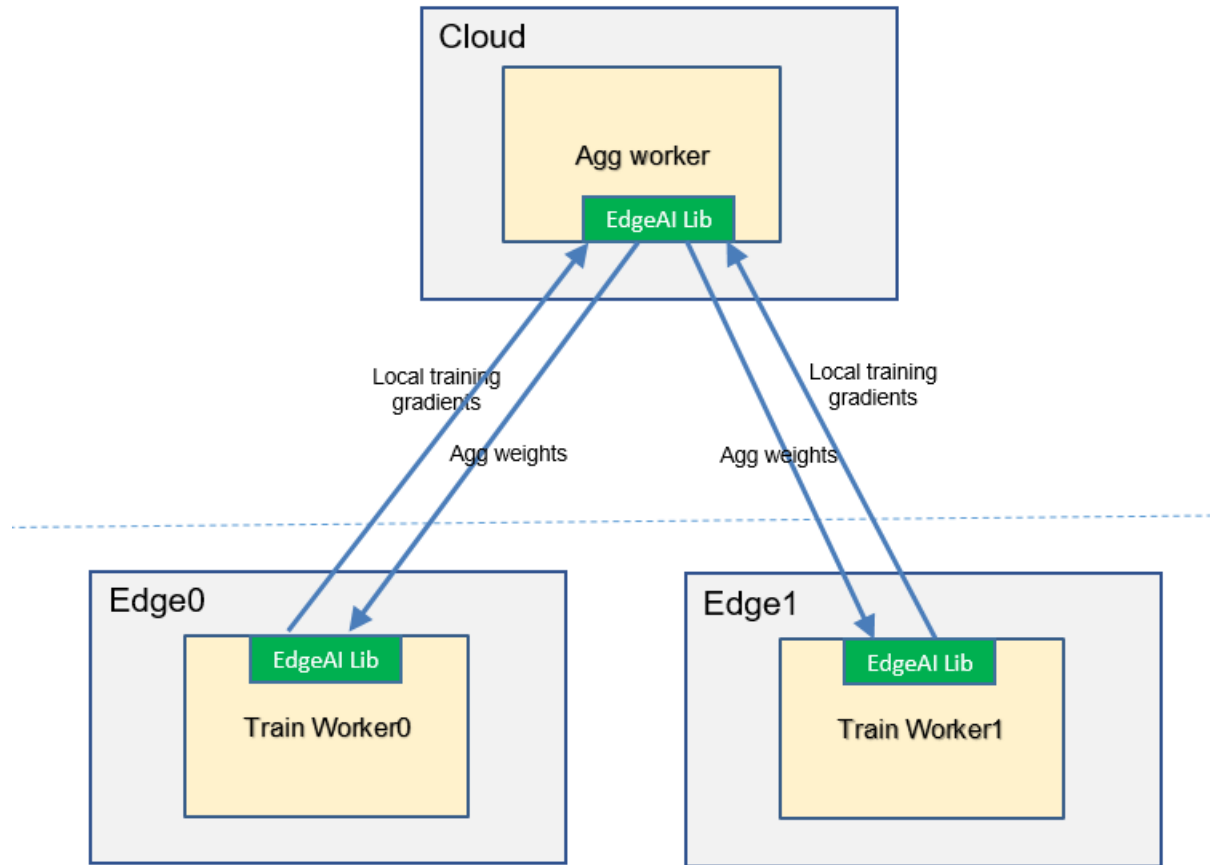
// JobInfo defines the job information
type JobInfo struct {
    // Current training round
    CurrentRound int `json:"currentRound"`
    UpdateTime string `json:"updateTime"`
    SampleCount int `json:"sampleCount"`
}
```

8.4.5 The flow of federated learning job creation



The federated-learning controller watches the creation of federatedlearningjob crd in the cloud, syncs them to lc via the cloudhub-to-edgehub channel, and creates the aggregator worker on the cloud nodes and the training workers on the edge nodes specified by the user. The aggregator worker is started by the native k8s at the cloud nodes. These training workers are started by the kubeedge at the edge nodes.

8.5 Workers Communication



- Incremental Learning
 - Motivation
 - * Goals
 - * Non-goals
 - Proposal
 - * Use Cases
 - Design Details
 - * CRD API Group and Version
 - * Incremental learning CRD
 - * Incremental learning type definition
 - * Incremental learning sample
 - * Validation

- Controller Design
 - * Incremental Learning Controller
 - * Downstream Controller
 - * Upstream Controller
 - * Details of api between GM(cloud) and LC(edge)
- Workers Communication

INCREMENTAL LEARNING

9.1 Motivation

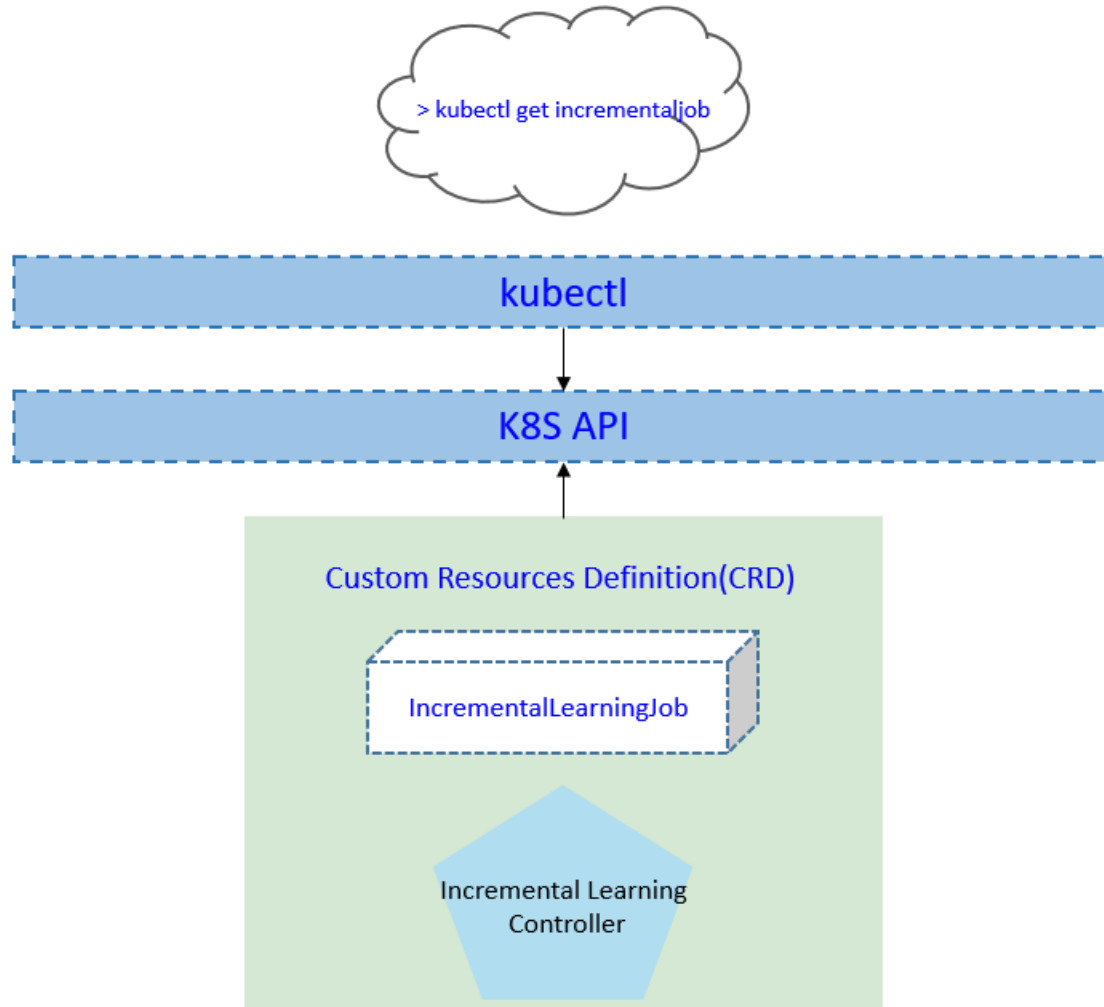
Data is continuously generated on the edge side. Traditionally, the data is collected manually and periodically retrained on the cloud to improve the model effect. This method wastes a lot of human resources, and the model update frequency is slow. Incremental learning allows users to continuously monitor the newly generated data and by configuring some triggering rules to determine whether to start training, evaluation, and deployment automatically, and continuously improve the model performance.

9.1.1 Goals

- Automatically retrains, evaluates, and updates models based on the data generated at the edge.
- Support time trigger, sample size trigger, and precision-based trigger.
- Support manual triggering of training, evaluation, and model update.
- support hard sample discovering of unlabeled data, for reducing the manual labeling workload.
- Support lifelong learning that reserves historical knowledge to avoid frequent re-training/ re-fine-tuning, and tackles samples uncovered in historical knowledge base.

9.2 Proposal

We propose using Kubernetes Custom Resource Definitions (CRDs) to describe the incremental learning specification/status and a controller to synchronize these updates between edge and cloud.



9.2.1 Use Cases

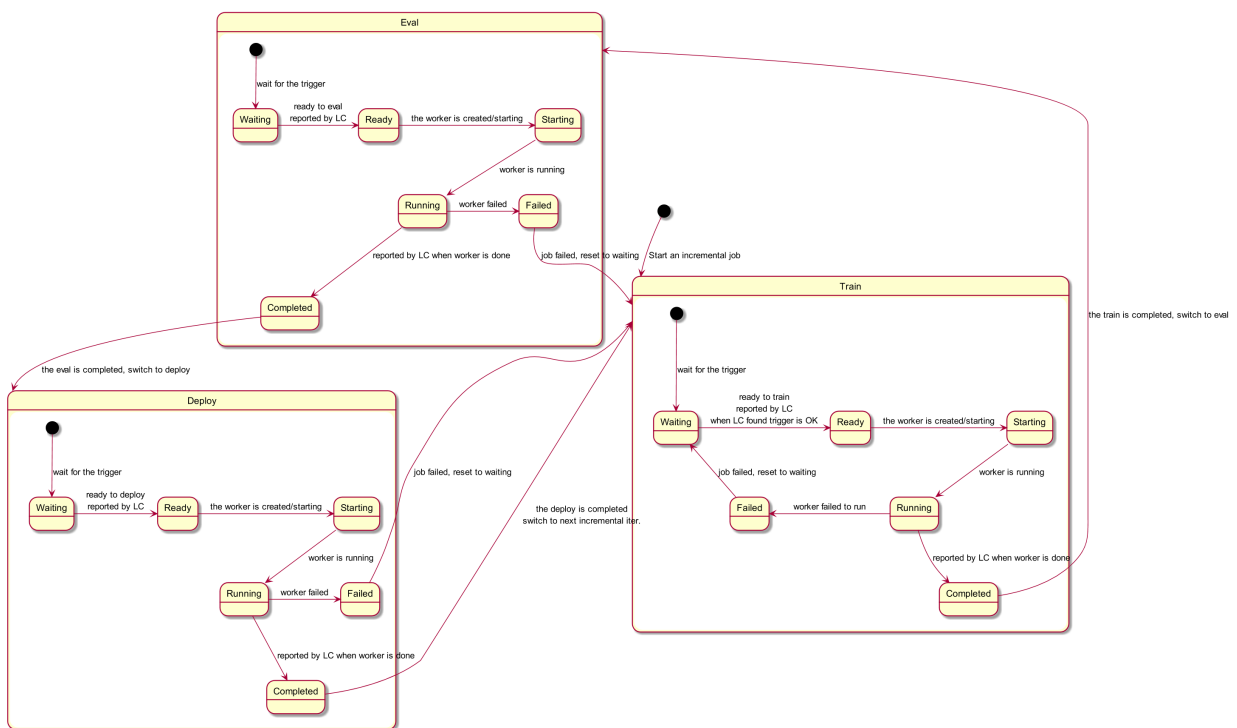
- Users can create the incremental learning jobs, by providing training scripts, configuring training hyperparameters, providing training datasets, configuring training and deployment triggers.

9.3 Design Details

There are three stages in a incremental learning job: train/eval/deploy.

Each stage contains these below states:

1. Waiting: wait to trigger satisfied, i.e. wait to train/eval/deploy
2. Ready: the corresponding trigger satisfied, now ready to train/eval/deploy
3. Starting: the corresponding stage is starting
4. Running: the corresponding stage is running
5. Failed: the corresponding stage failed
6. Completed: the corresponding stage completed



9.3.1 CRD API Group and Version

The `IncrementalLearningJob` CRD will be namespace-scoped. The tables below summarize the group, kind and API version details for the CRD.

- `IncrementalLearningJob`

Field	Description
Group	sedna.io
APIVersion	v1alpha1
Kind	IncrementalLearningJob

9.3.2 Incremental learning CRD

See the [crd source](#) for details.

9.3.3 Incremental learning job type definition

See the [golang source](#) for details.

Validation

[Open API v3 Schema based validation](#) can be used to guard against bad requests. Invalid values for fields (example string value for a boolean field etc) can be validated using this.

Here is a list of validations we need to support :

1. The `dataset` specified in the `crd` should exist in k8s.
2. The `model` specified in the `crd` should exist in k8s.
3. The `edgenode` name specified in the `crd` should exist in k8s.

9.3.4 Incremental learning job sample

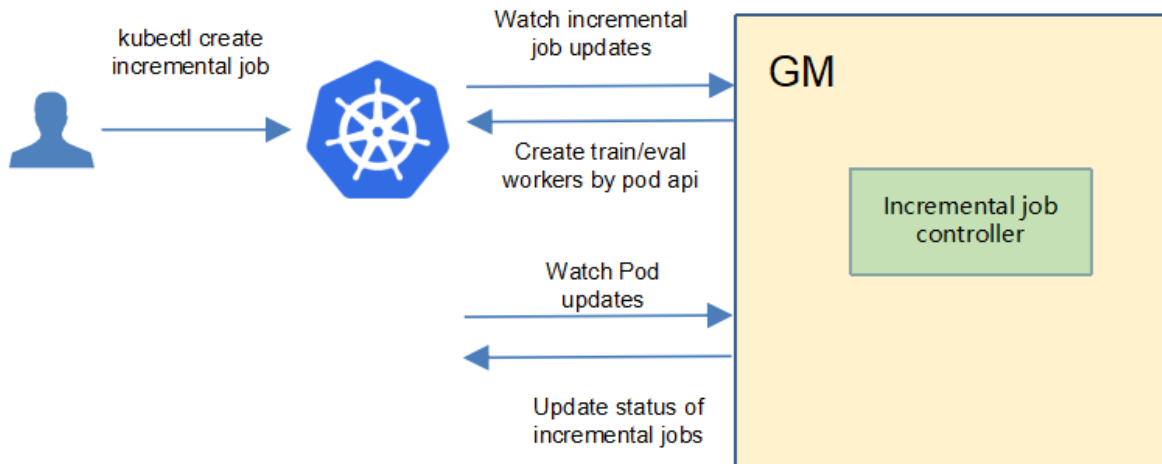
See the [source](#) for an example.

9.4 Controller Design

The incremental learning controller starts three separate goroutines called `upstream`, `downstream` and `incrementallearningjobcontroller`. These are not separate controllers as such but named here for clarity.

- `incremental learning`: watch the updates of incremental-learning job crds, and create the workers depending on the state machine.
- `downstream`: synchronize the incremental-learning-job updates from the cloud to the edge node.
- `upstream`: synchronize the incremental-learning-job updates from the edge to the cloud node.

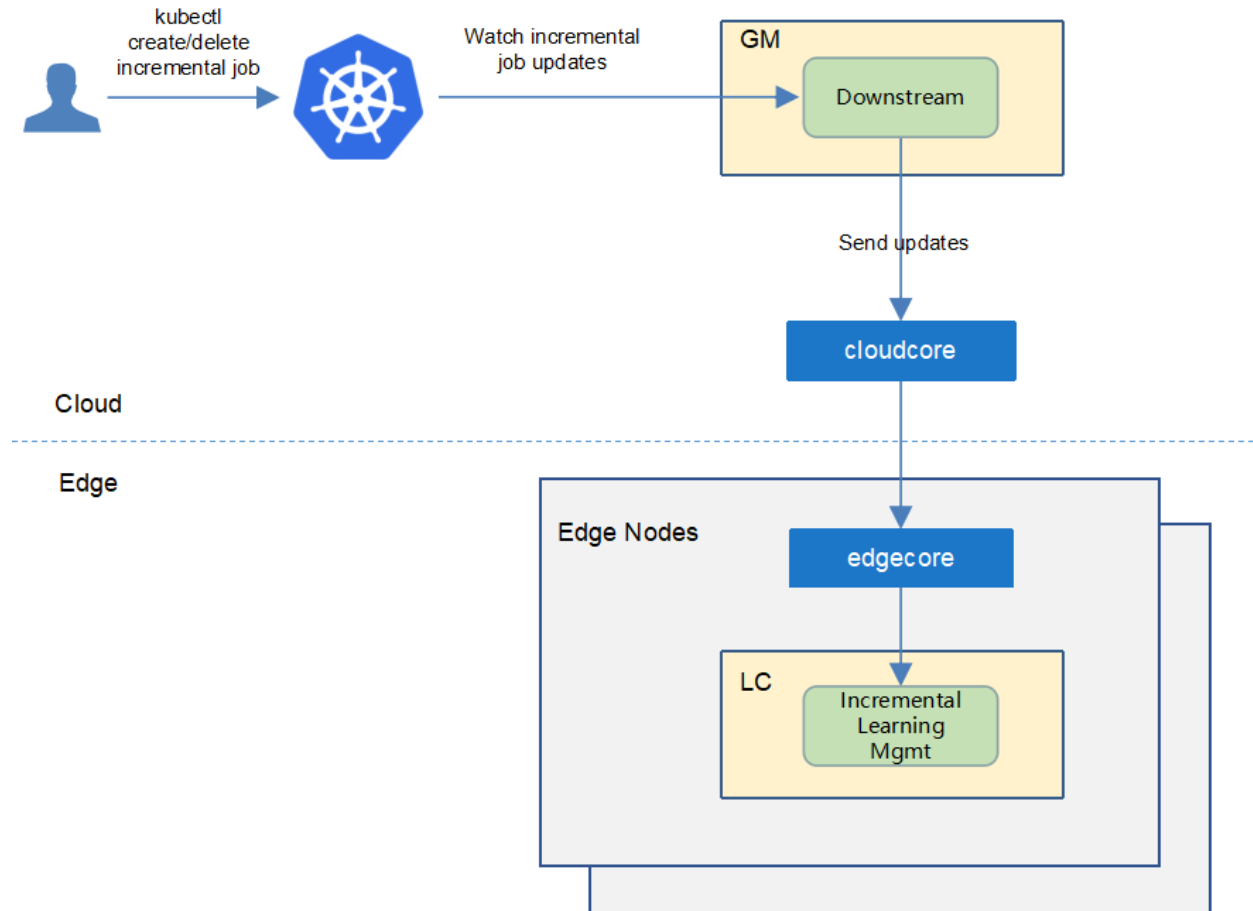
9.4.1 Incremental Learning Controller



The incremental-learning controller watches for the updates of incremental-learning jobs and the corresponding pods against the K8S API server. Updates are categorized below along with the possible actions:

Update Type	Action
New Incremental-learning-job Created	Wait to train trigger satisfied
Incremental-learning-job Deleted	NA. These workers will be deleted by <code>k8s gc</code> .
The Status of Incremental-learning-job Updated	Create the train/eval worker if it's ready.
The corresponding pod created/running/completed/failed	Update the status of incremental-learning job.

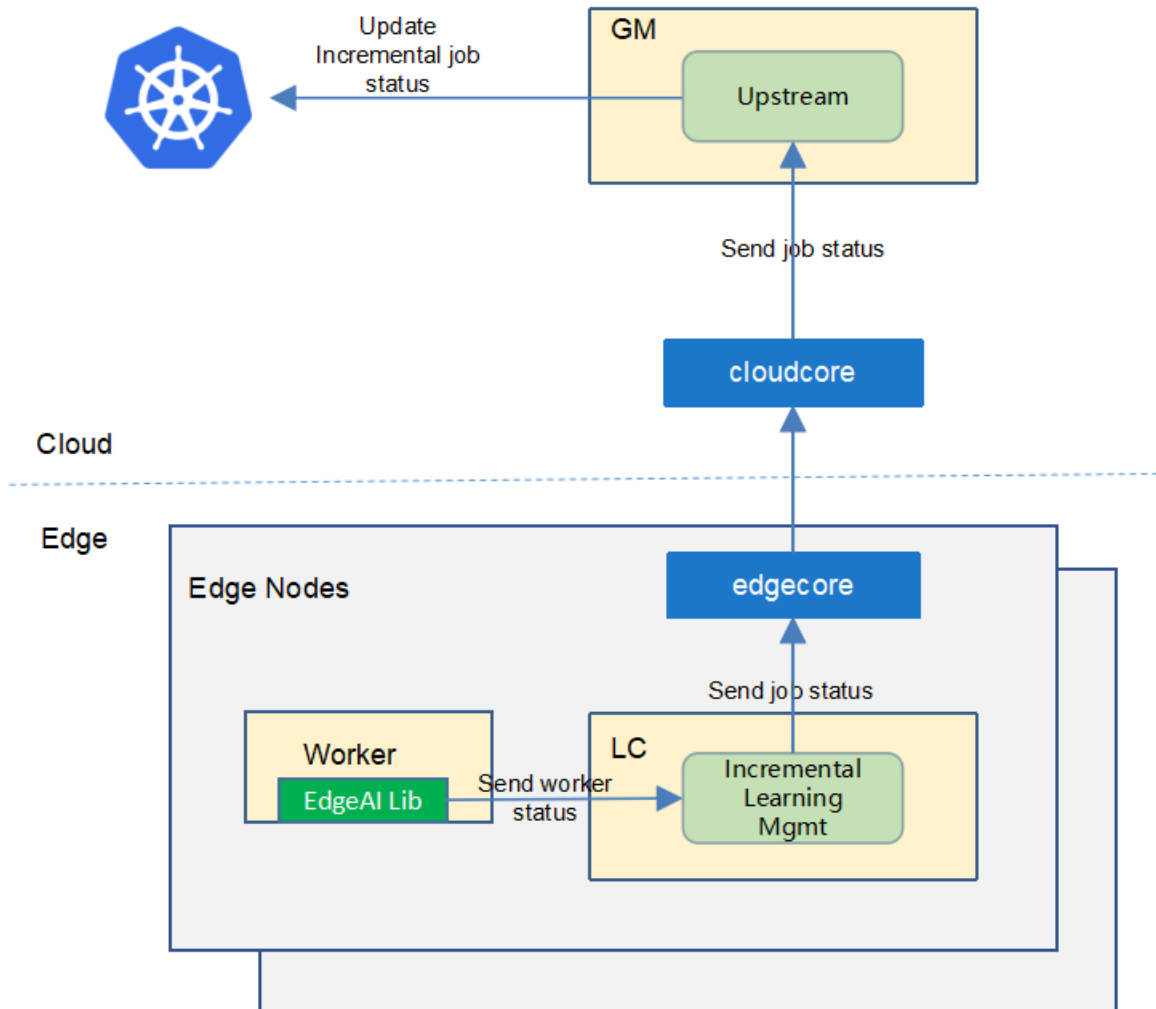
9.4.2 Downstream Controller



The downstream controller watches for the incremental-learning job updates against the K8S API server. Updates are categorized below along with the possible actions that the downstream controller can take:

Update Type	Action
New Incremental-learning-job Created	Sends the job information to LCs.
Incremental-learning-job Deleted	The controller sends the delete event to LCs.

9.4.3 Upstream Controller



The upstream controller watches for the incremental-learning job updates from the edge node and applies these updates against the API server in the cloud. Updates are categorized below along with the possible actions that the upstream controller can take:

Update Type	Action
Incremental-learning-job Reported State Up-dated	The controller appends the reported status of the job by LC in the cloud.

9.4.4 Details of api between GM(cloud) and LC(edge)

1. GM(downstream controller) syncs the job info to LC:

```
// POST <namespace>/incrementallearningjobs/<job-name>
// body same to the job crd of k8s api, omitted here.
```

2. LC uploads the job status which reported by the worker to GM(upstream controller):

```
// POST <namespace>/incrementallearningjobs/<job-name>/status

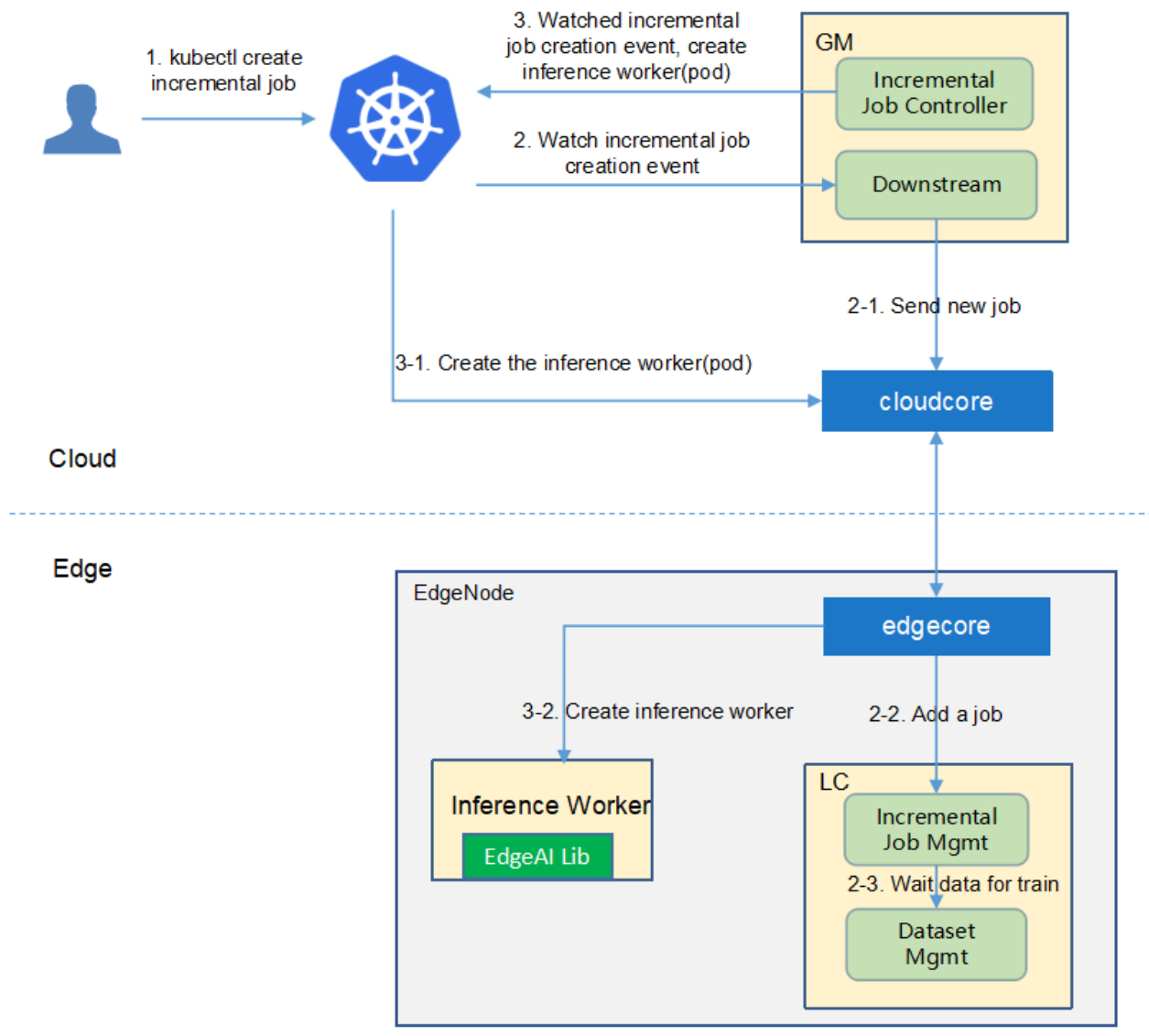
// WorkerMessage defines the message from that the training worker. It will send to GM.
↪GM.
type WorkerMessage struct {
    Phase string `json:"phase"`
    Status string `json:"status"`
    Output *WorkerOutput `json:"output"`
}
//
type WorkerOutput struct {
    Models []*Model `json:"models"`
    OwnerInfo *OwnerInfo `json:"ownerInfo"`
}

// Model defines the model information
type Model struct {
    Format string `json:"format"`
    URL string `json:"url"`
    // Including the metrics, e.g. precision/recall
    Metrics map[string]float64 `json:"metrics"`
}

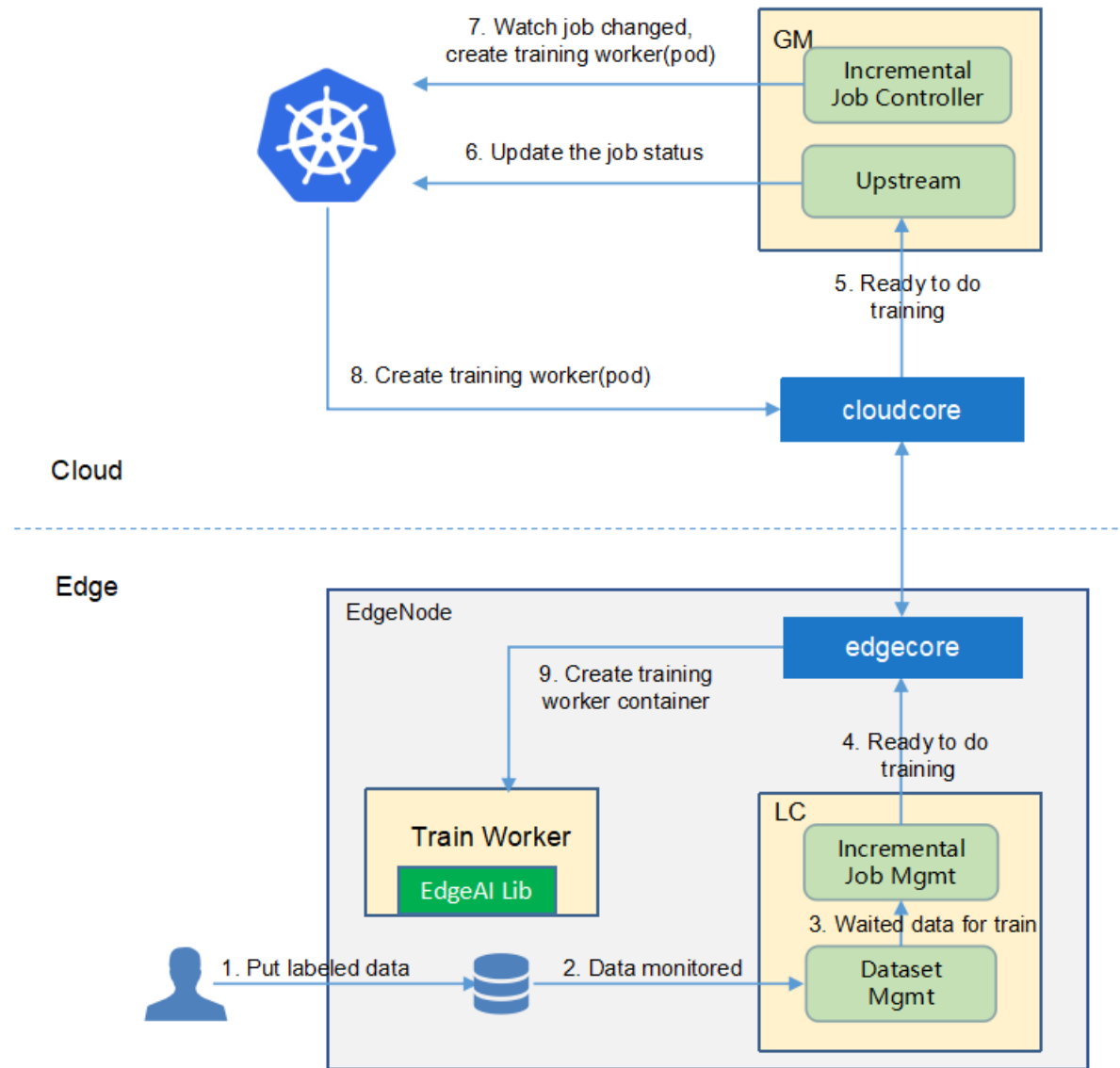
// TaskInfo defines the task information
type TaskInfo struct {
    // Current training round
    CurrentRound int `json:"currentRound"`
    UpdateTime string `json:"updateTime"`
}
```

9.4.5 The flows of incremental learning job

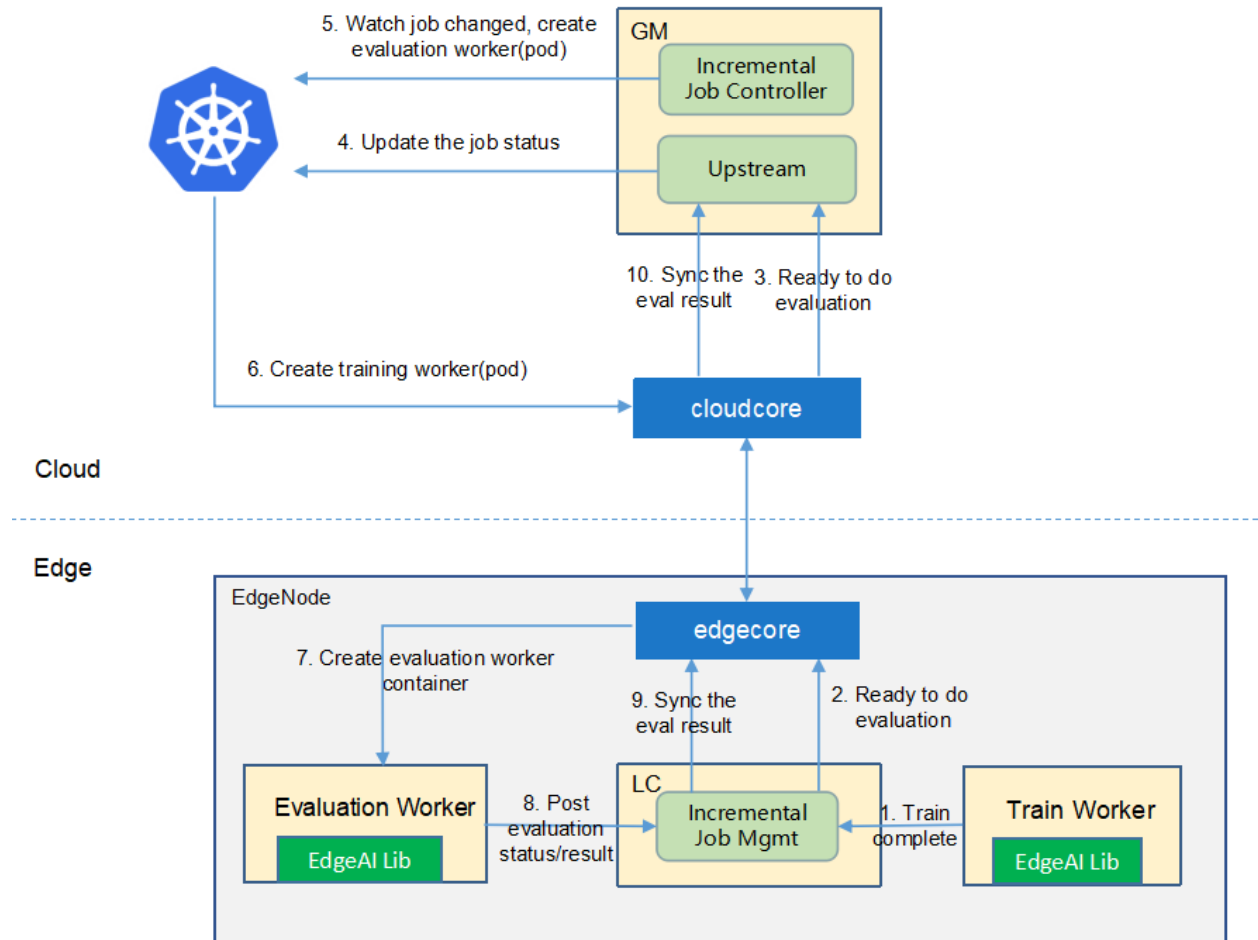
- Flow of the job creation:



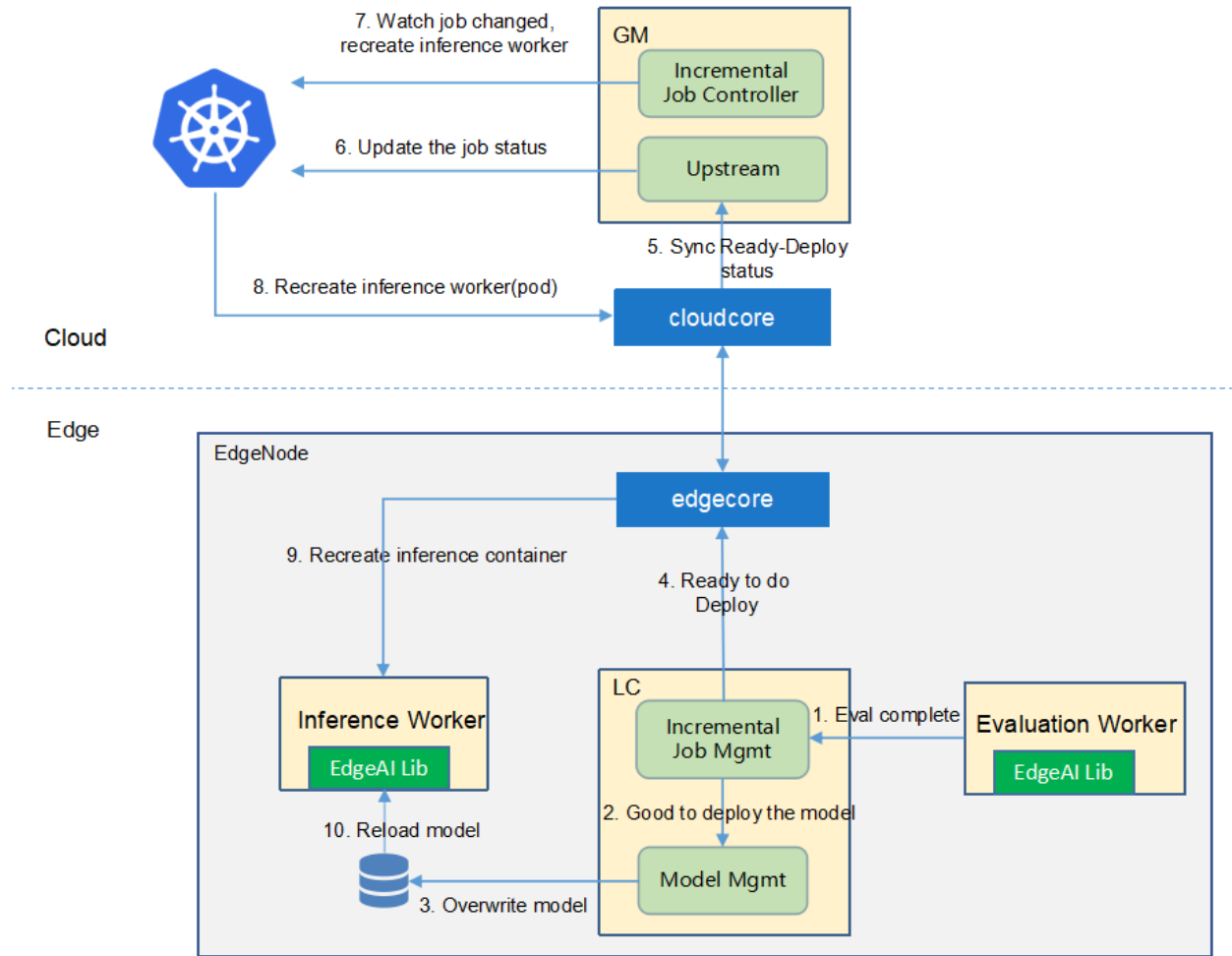
- Flow of the train stage:



- Flow of the eval stage:



- Flow of the deploy stage:



9.5 Workers Communication

No need to communicate between workers.

- Joint Inference
 - Motivation
 - * Goals
 - * Non-goals
 - Proposal
 - * Use Cases
 - Design Details
 - * CRD API Group and Version
 - * Joint inference CRD
 - * Joint inference type definition
 - * Joint inference sample

- * Validation
- Controller Design
 - * Joint Inference Controller
 - * Downstream Controller
 - * Upstream Controller
 - * Details of api between GM(cloud) and LC(edge)
 - * Details of api between Worker(edge) and LC(edge)
 - * Flow of Joint Inference
- Workers Communication

JOINT INFERENCE

10.1 Motivation

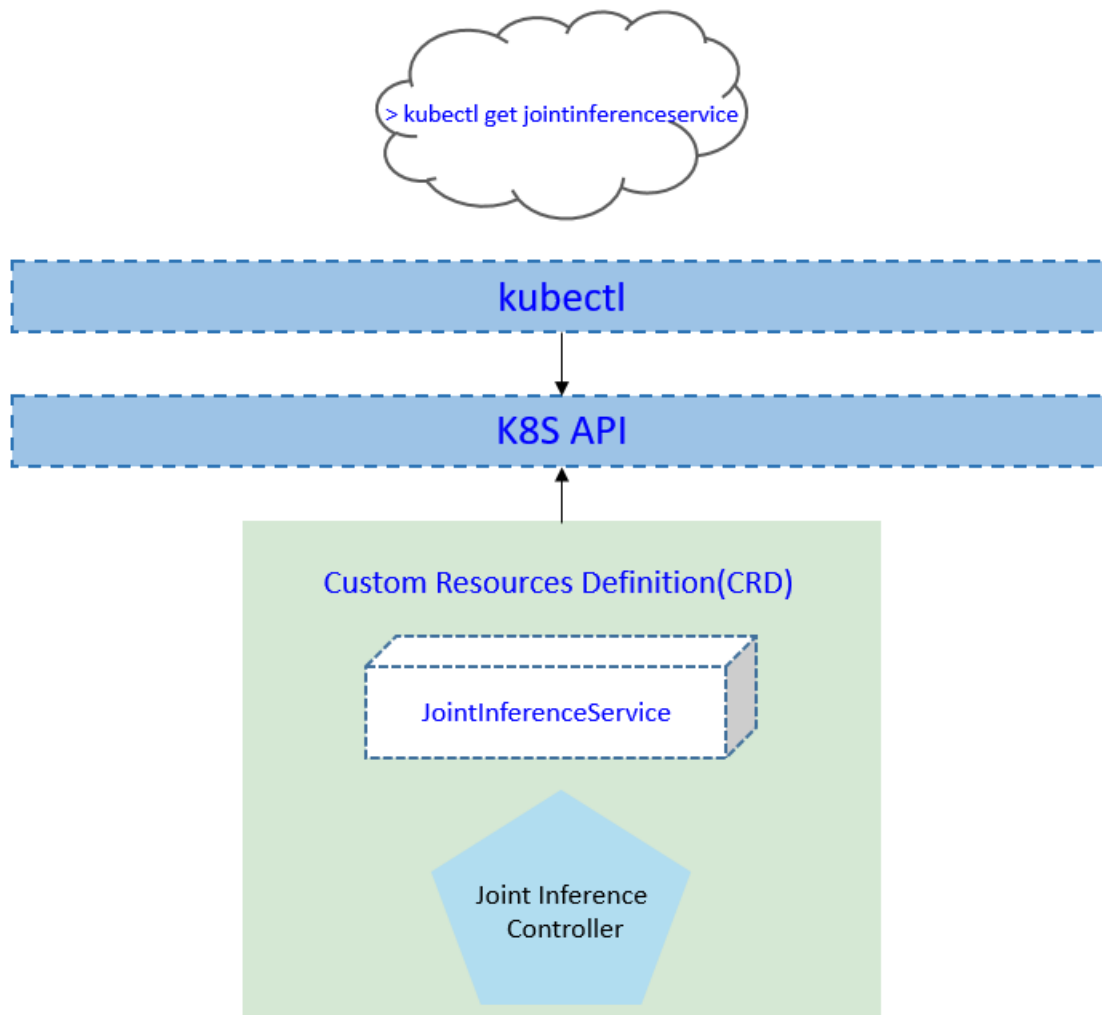
Inference on the edge can get a shorter latency and a higher throughput, and inference on the cloud can get better inference precision. The collaborative inference technology detects hard samples on the edge and sends them to the cloud for inference. **In this way, simple samples inference on the edge ensures latency and throughput, while hard samples inference on the cloud improves the overall precision.**

10.1.1 Goals

- Joint inference improves the inference precision without significantly reducing the time and throughput.

10.2 Proposal

We propose using Kubernetes Custom Resource Definitions (CRDs) to describe the joint inference specification/status and a controller to synchronize these updates between edge and cloud.



10.2.1 Use Cases

- User can create a joint inference service with providing a training script, specifying the aggregation algorithm, configuring training hyper parameters, configuring training datasets.
- Users can get the joint inference status, including the counts of inference at the edge/cloud.

10.3 Design Details

10.3.1 CRD API Group and Version

The `JointInferenceService` CRD will be namespace-scoped. The tables below summarize the group, kind and API version details for the CRD.

- `JointInferenceService`

Field	Description
Group	sedna.io
APIVersion	v1alpha1
Kind	JointInferenceService

10.3.2 Joint inference CRD

see [crd source](#)

10.3.3 Joint inference type definition

see [go source](#)

Validation

[Open API v3 Schema based validation](#) can be used to guard against bad requests. Invalid values for fields (example string value for a boolean field etc) can be validated using this.

Here is a list of validations we need to support :

1. The `dataset` specified in the crd should exist in k8s.
2. The `model` specified in the crd should exist in k8s.
3. The `edgenode` name specified in the crd should exist in k8s.

10.3.4 joint inference sample

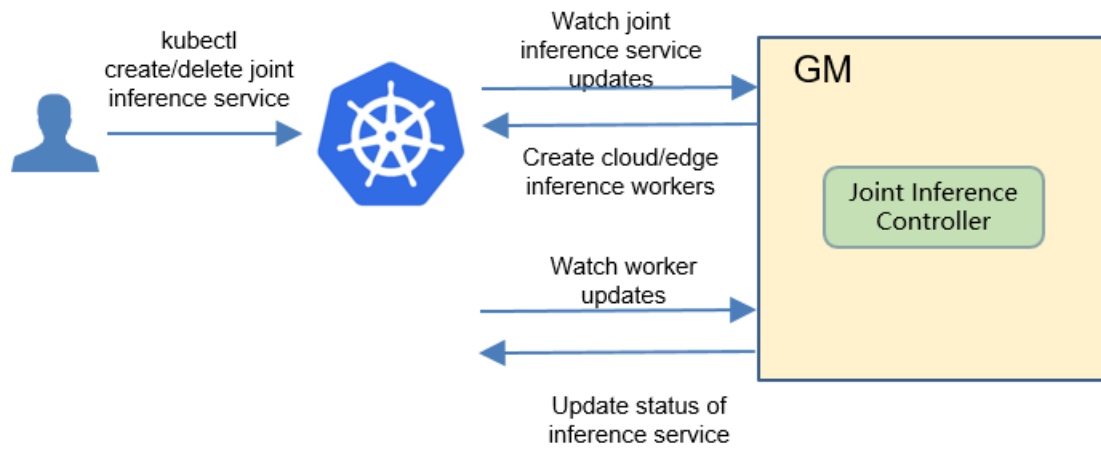
see [sample source](#)

10.4 Controller Design

The joint inference controller starts three separate goroutines called `upstream`, `downstream` and `joint-inferencecontroller`. These are not separate controllers as such but named here for clarity.

- `joint inference`: watch the updates of `joint-inference-task` crds, and create the workers to complete the task.
- `downstream`: synchronize the `joint-inference` updates from the cloud to the edge node.
- `upstream`: synchronize the `joint-inference` updates from the edge to the cloud node.

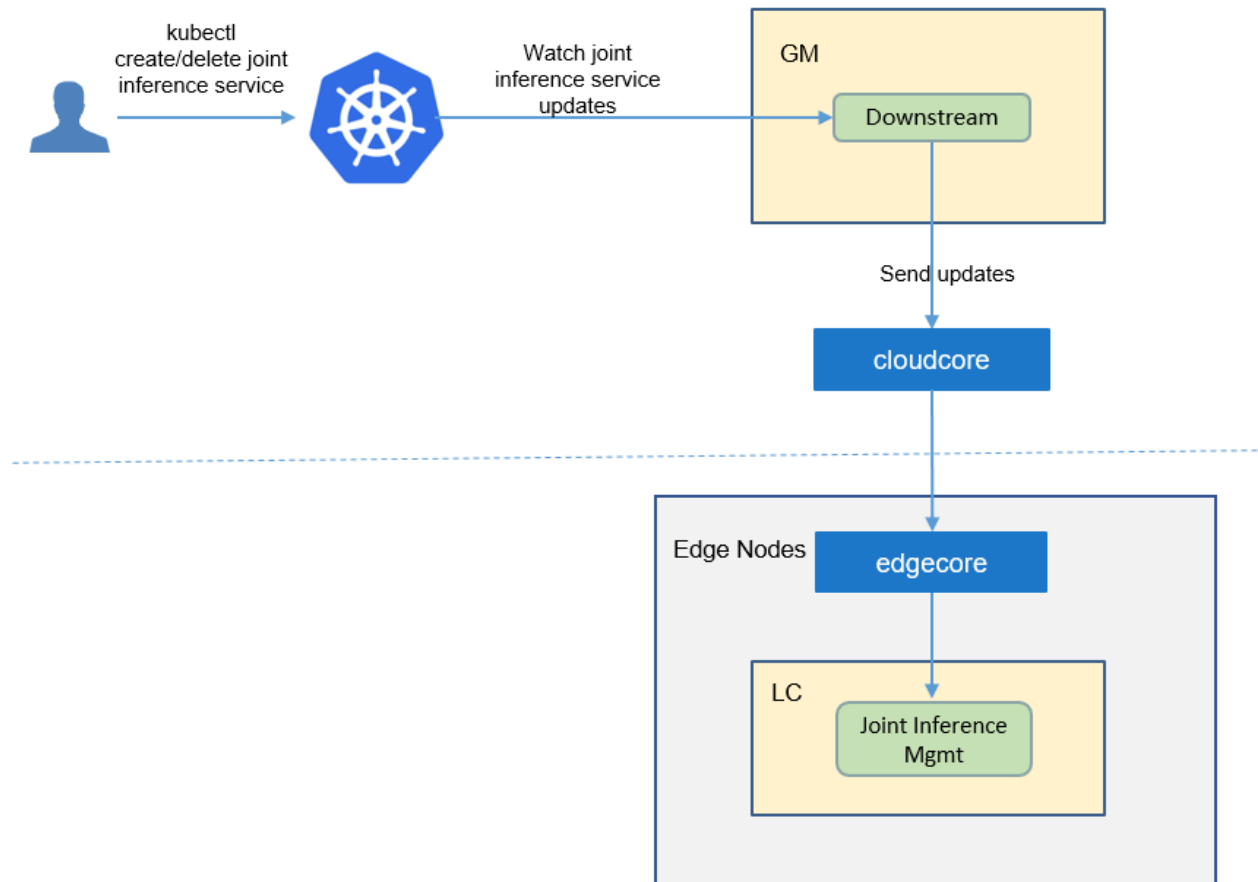
10.4.1 Joint Inference Controller



The joint-inference controller watches for the updates of joint-inference tasks and the corresponding pods against the K8S API server. Updates are categorized below along with the possible actions:

Update Type	Action
New Joint-inference-service Created	Create the cloud/edge worker
Joint-inference-service Deleted	NA. These workers will be deleted by GM.
The corresponding pod created/running/completed/failed	Update the status of joint-inference task.

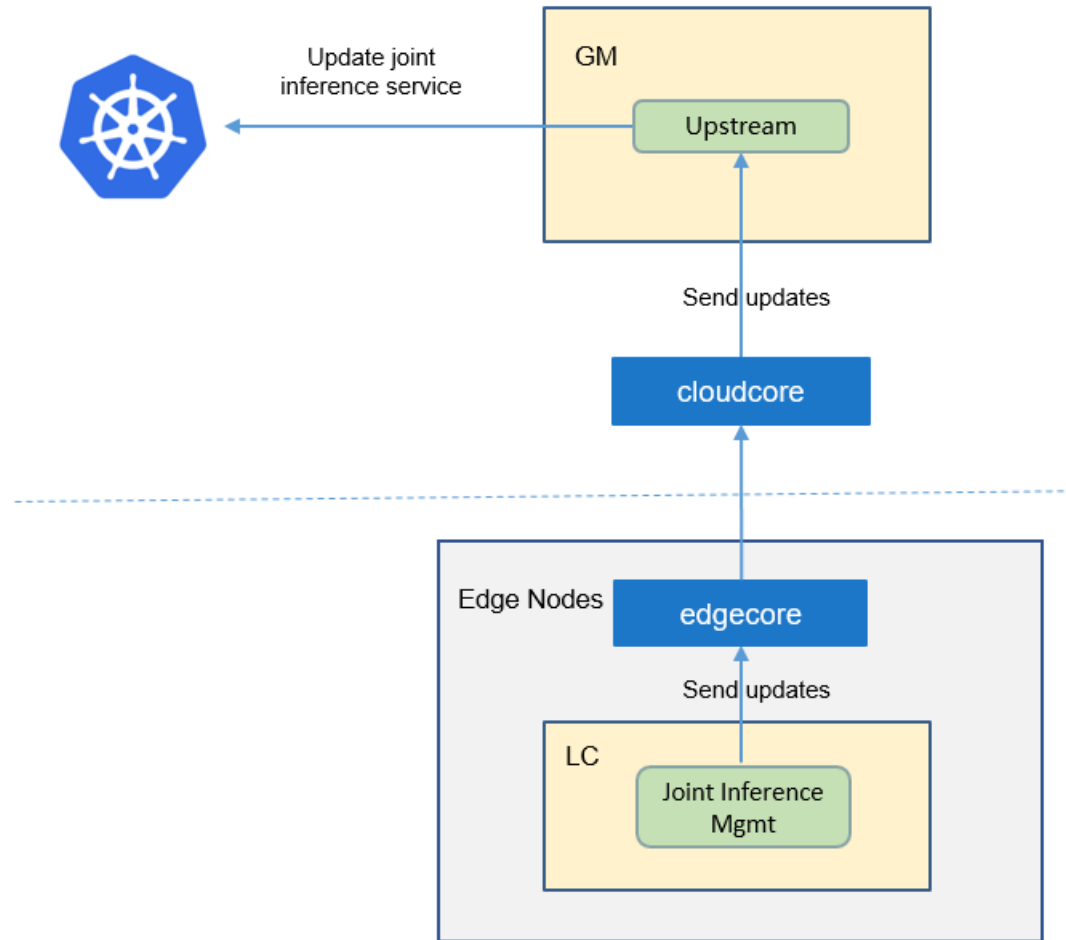
10.4.2 Downstream Controller



The downstream controller watches for joint-inference updates against the K8S API server. Updates are categorized below along with the possible actions that the downstream controller can take:

Update Type	Action
New Joint-inference-service Created	Sends the task information to LCs.
Joint-inference-service Deleted	The controller sends the delete event to LCs.

10.4.3 Upstream Controller



The upstream controller watches for joint-inference-task updates from the edge node and applies these updates against the API server in the cloud. Updates are categorized below along with the possible actions that the upstream controller can take:

Update Type	Action
Joint-inference-service Reported State Updated	The controller appends the reported status of the Joint-inference-service in the cloud.

10.4.4 Details of api between GM(cloud) and LC(edge)

1. GM(downstream controller) syncs the task info to LC:

```
// POST <namespace>/sedna/downstream/jointinferenceservices/<name>/insert
// body same to the task crd of k8s api, omitted here.
```

2. LC uploads the task status which reported by the worker to GM(upstream controller):


```
// POST <namespace>/sedna/upstream/jointinferenceservices/<name>/status

// JoinInferenceServiceStatus defines status that send to GlobalManager
type JoinInferenceServiceStatus struct {
    Phase string `json:"phase"`
    Status string `json:"status"`
    Output *Output `json:"output"`
}

// Output defines task output information
type Output struct {
    Models []Model `json:"models"`
    TaskInfo *TaskInfo `json:"taskInfo"`
}

// Model defines the model information
type Model struct {
    Format string `json:"format"`
    URL string `json:"url"`
}

// TaskInfo defines the task information
type TaskInfo struct {
    InferenceNumber int `json:"inferenceNumber"`
    HardExampleNumber int `json:"hardExampleNumber"`
    UploadCloudRatio float64 `json:"uploadCloudRatio"`
    StartTime string `json:"startTime"`
    CurrentTime string `json:"currentTime"`
}
```

10.4.5 Details of api between Worker(edge) and LC(edge)

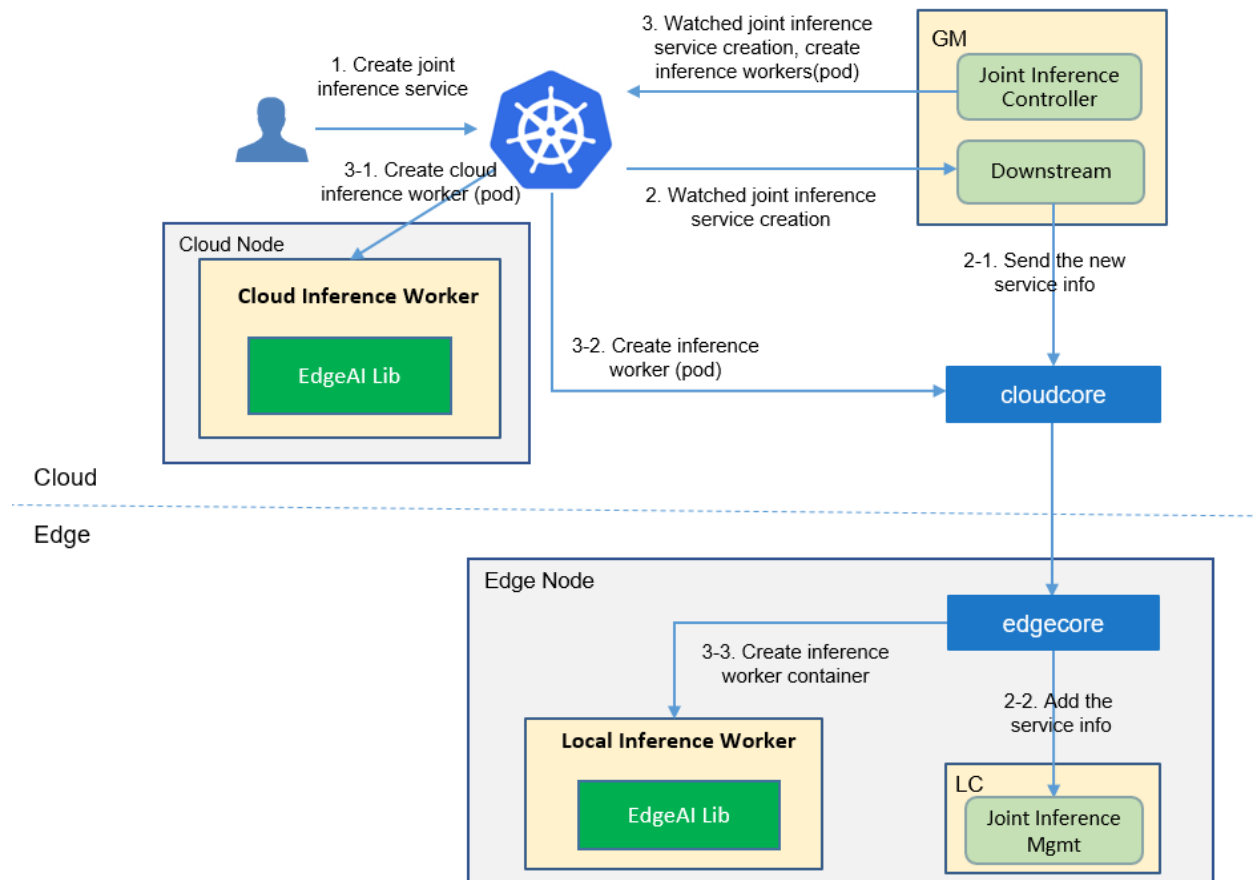
1. Worker sends inference info to LC in same edge node:

```
// POST /sedna/workers/<worker-name>/info
```

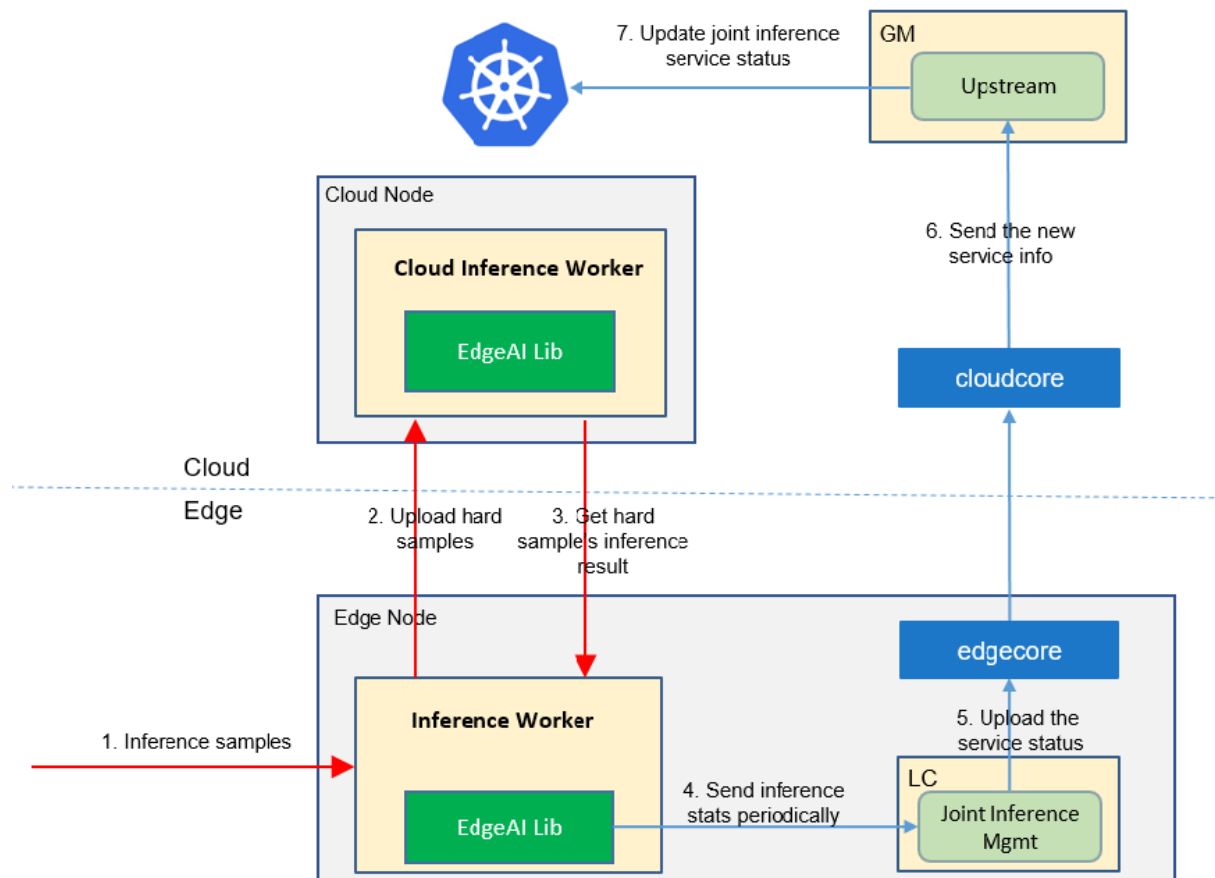
```
{
  "name": "worker-name",
  "namespace": "default",
  "ownerName": "jointinferenceservice-name",
  "ownerKind": "jointinferenceservice",
  "kind": "inference",
  "status": "completed/failed/running",
  "taskInfo": {
    "inferenceNumber": 1000,
    "hardExampleNumber": 100,
    "uploadCloudRatio": 0.1,
    "startTime": "2020-11-03T08:39:22.517Z",
    "updateTime": "2020-11-03T08:50:22.517Z"
  }
}
```

10.4.6 Flow of Joint Inference

- The flow of joint inference service creation:



10.5 Workers Communication



- Object Search Service
 - Motivation
 - * Goals
 - Proposal
 - * Use Cases
 - Design Details
 - * CRD API Group and Version
 - * Object search service type definition
 - Validation
 - * Object search service sample
 - Controller Design
 - * Object search service Controller
 - * Downstream Controller
 - * Upstream Controller

- * Details of api between GM(cloud) and LC(edge)
- * Flow of object search service creation
- Workers Communication

OBJECT SEARCH SERVICE

11.1 Motivation

Object search is an important technology in the field of computer vision, which is widely used in security monitoring, intelligent transportation, etc. Generally, online object search applications have stringent latency constraints, which cannot be met by cloud computing schemes. Object search schemes based on edge computing have the characteristics of low latency and data privacy security, and are the mainstream technology trend.

However, the amount of feature matching computation increases exponentially with the expansion of the search object scale, and a single edge computing node is difficult to support large-scale object search. Multi-edge collaborative computing can accelerate large-scale object search applications and improve the search accuracy, which is the future development trend.

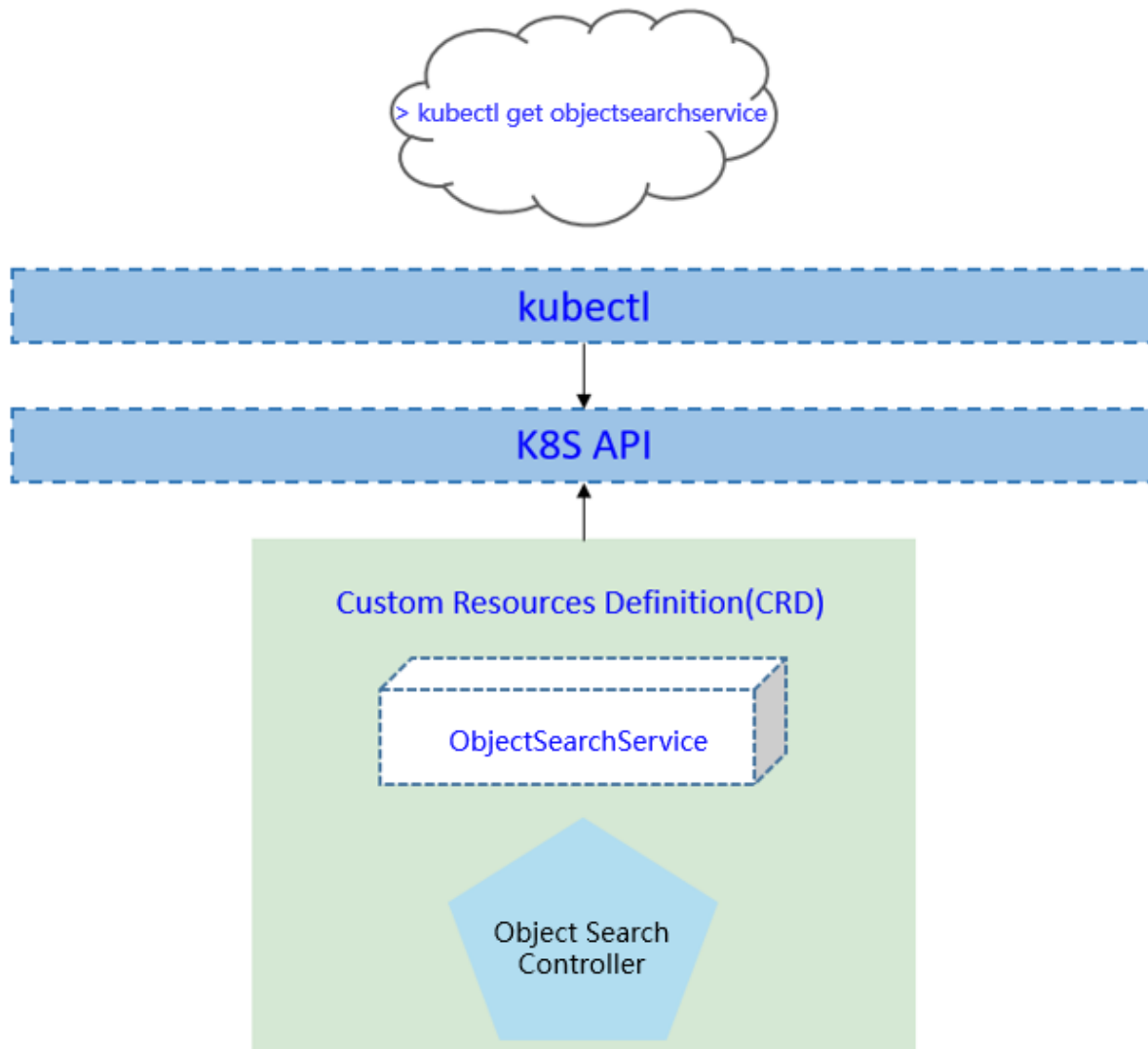
We propose the first open source end-to-end multi-edge collaborative object search solution. Based on KubeEdge's cloud-edge collaboration and resource management capabilities, we utilize multiple edge computing nodes to execute the AI inference tasks of object search in parallel. Our solution can not only reduce delay and improve throughput, but also will bring accuracy promotion. In addition, our solution will also support efficient offline object search.

11.1.1 Goals

- Support single/multi-object search
- Support across-camera object search
- Support parallel object re-identification(ReID)
- Support historical video data object search
- Support multi-camera data joint analysis and decision making
- Provide a user entry for submitting search tasks and result queries

11.2 Proposal

We propose using Kubernetes Custom Resource Definitions (CRDs) to describe the object search service specification/status and a controller to synchronize these updates between edge and cloud.



11.2.1 Use Cases

- User can create typical multi-edge collaborative object search applications with providing AI models.

11.3 Design Details

11.3.1 CRD API Group and Version

The ObjectSearchService CRD will be namespace-scoped. The tables below summarize the group, kind and API version details for the CRD.

- ObjectSearchService

Field	Description
Group	sedna.io
APIVersion	v1alpha1
Kind	ObjectSearchService

11.3.2 Object search service type definition

[go source](#)

Validation

[Open API v3 Schema based validation](#) can be used to guard against bad requests. Invalid values for fields (example string value for a boolean field etc) can be validated using this.

11.3.3 Object search service sample

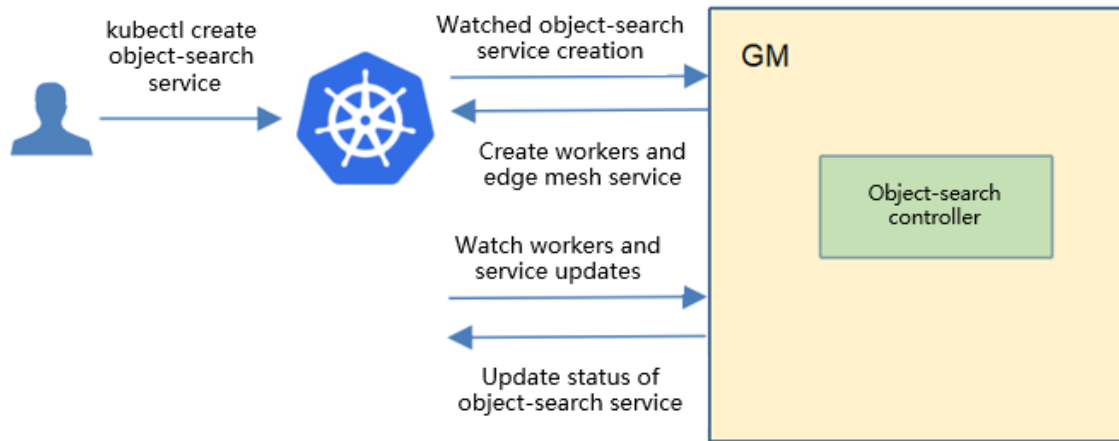
See the [source](#) for an example.

11.4 Controller Design

The object search service controller starts three separate goroutines called `upstream`, `downstream` and `object-search-service` controller. These are not separate controllers as such but named here for clarity.

- `object-search-service`: watch the updates of object-search-service task crds, and create the workers to complete the task.
- `downstream`: synchronize the object-search-service updates from the cloud to the edge node.
- `upstream`: synchronize the object-search-service updates from the edge to the cloud node.

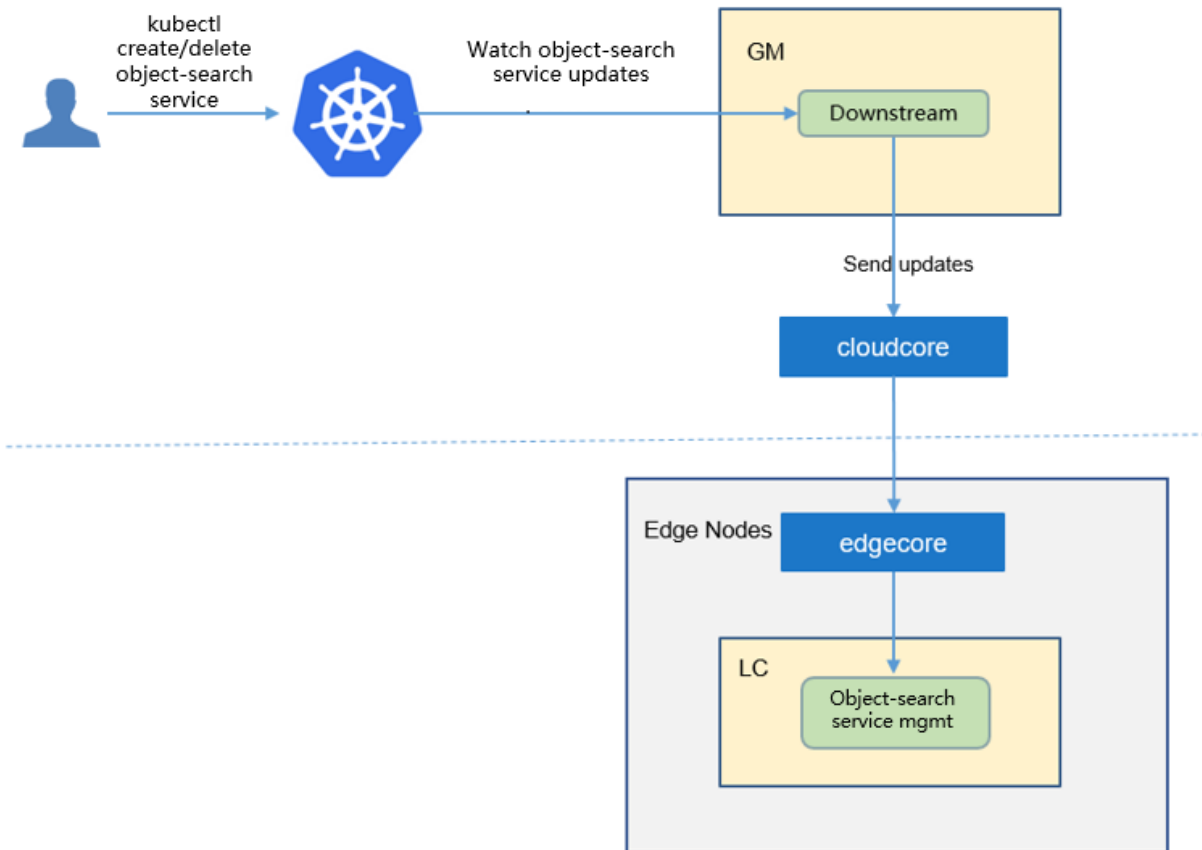
11.4.1 Object search service Controller



The object-search-service controller watches for the updates of object-search-service tasks and the corresponding pods against the K8S API server. Updates are categorized below along with the possible actions:

Update Type	Action
New Object-search-service Created	Create the cloud/edge worker
Object-search-service Deleted	NA. These workers will be deleted by GM.
The corresponding pod created/running/completed/failed	Update the status of object-search-service task.

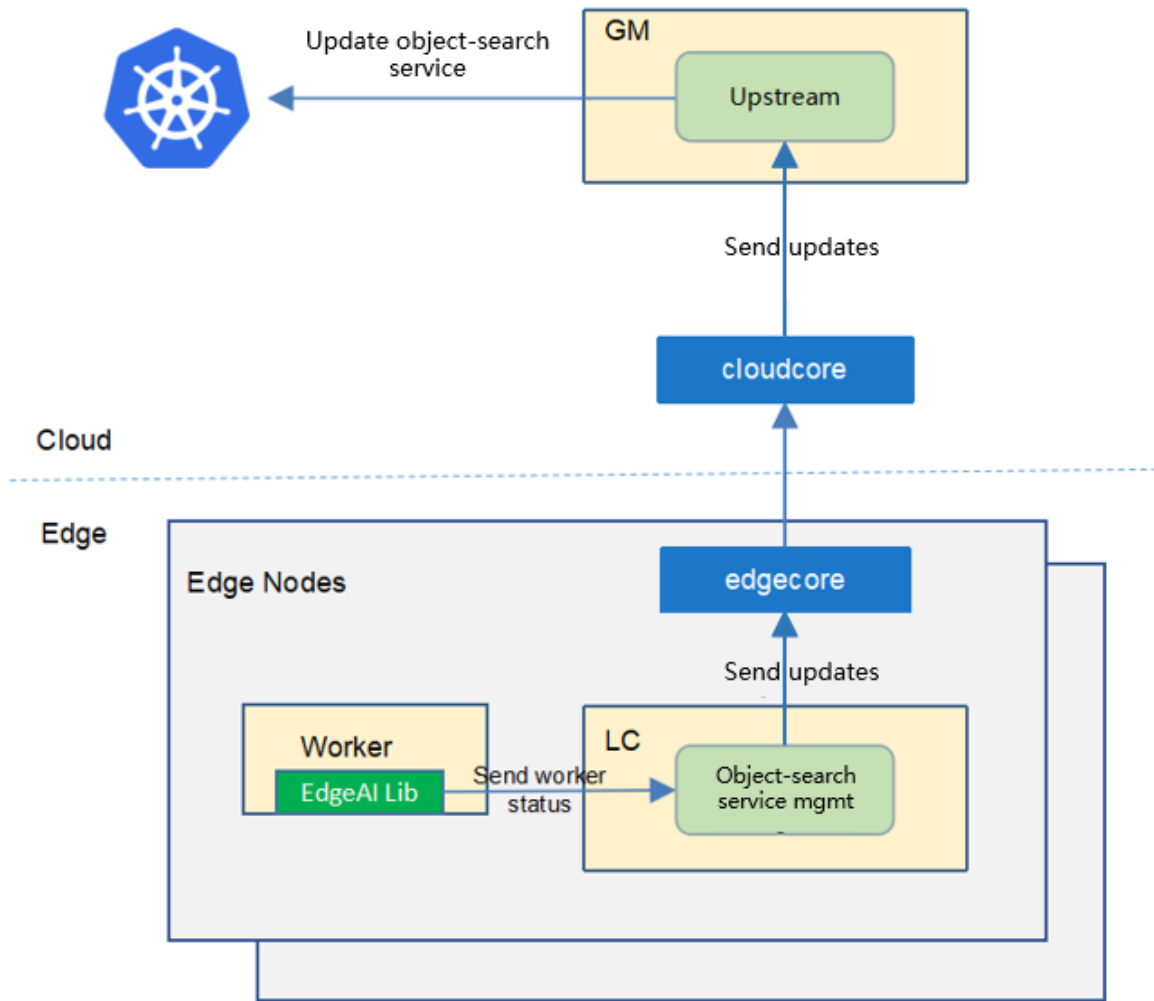
11.4.2 Downstream Controller



The downstream controller watches for object-search-service updates against the K8S API server. Updates are categorized below along with the possible actions that the downstream controller can take:

Update Type	Action
New Object-search-service Created	Sends the task information to LCs.
Object-search-service Deleted	The controller sends the delete event to LCs.

11.4.3 Upstream Controller



The upstream controller watches for object-search-service task updates from the edge node and applies these updates against the API server in the cloud. Updates are categorized below along with the possible actions that the upstream controller can take:

Update Type			Action
Object-search-service Updated	Reported	State	The controller appends the reported status of the object-search-service in the cloud.

11.4.4 Details of api between GM(cloud) and LC(edge)

1. GM(downstream controller) syncs the task info to LC:

```
// POST <namespace>/sedna/downstream/objectsearchservices/<name>/insert
// body same to the task crd of k8s api, omitted here.
```

2. LC uploads the task status which reported by the worker to GM(upstream controller):

```
// POST <namespace>/sedna/upstream/objectsearchservices/<name>/status

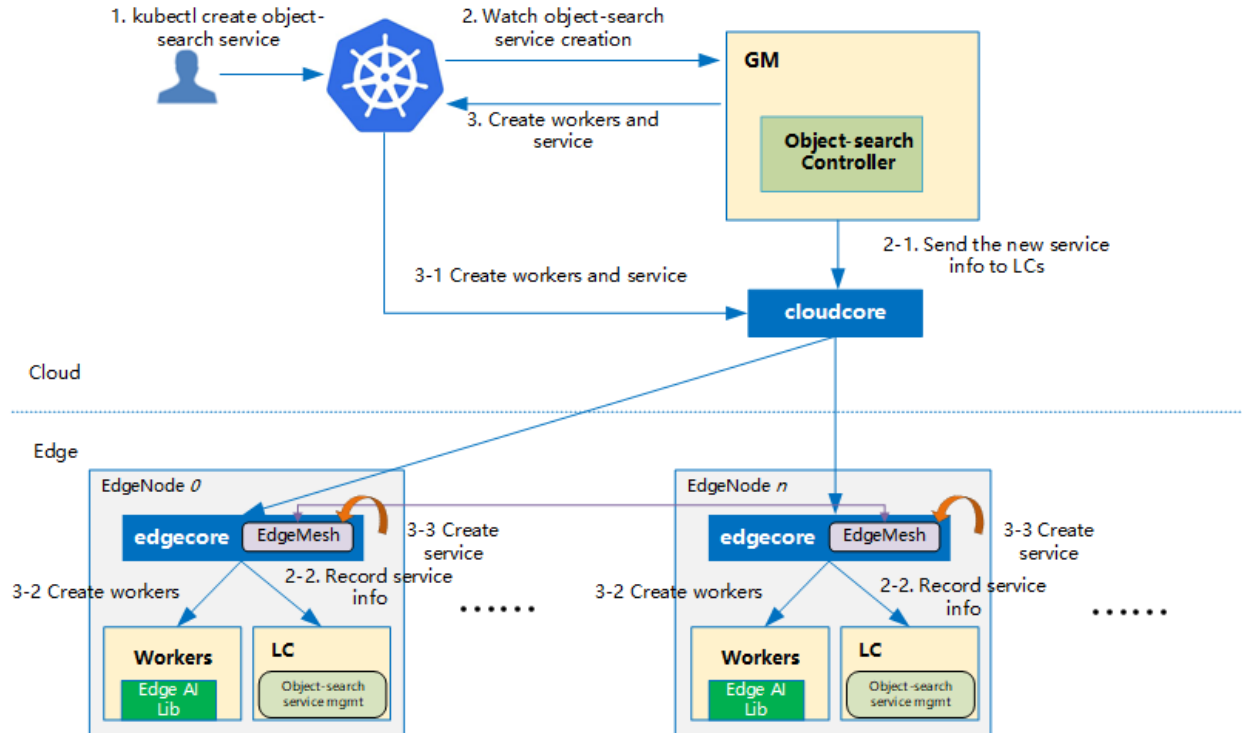
// ObjectSearchServiceStatus defines status that send to GlobalManager
type ObjectSearchServiceStatus struct {
    Phase string `json:"phase"`
    Status string `json:"status"`
    Output *Output `json:"output"`
}

// Output defines task output information
type Output struct {
    TaskInfo *TaskInfo `json:"taskInfo"`
}

// TaskInfo defines the task information
type TaskInfo struct {
    SearchingObjectNumber int    `json:"searchingObjectNumber"`
    SearchedObjectNumber int    `json:"searchedObjectNumber"`
    StartTime             string `json:"startTime"`
    CurrentTime           string `json:"currentTime"`
}
```

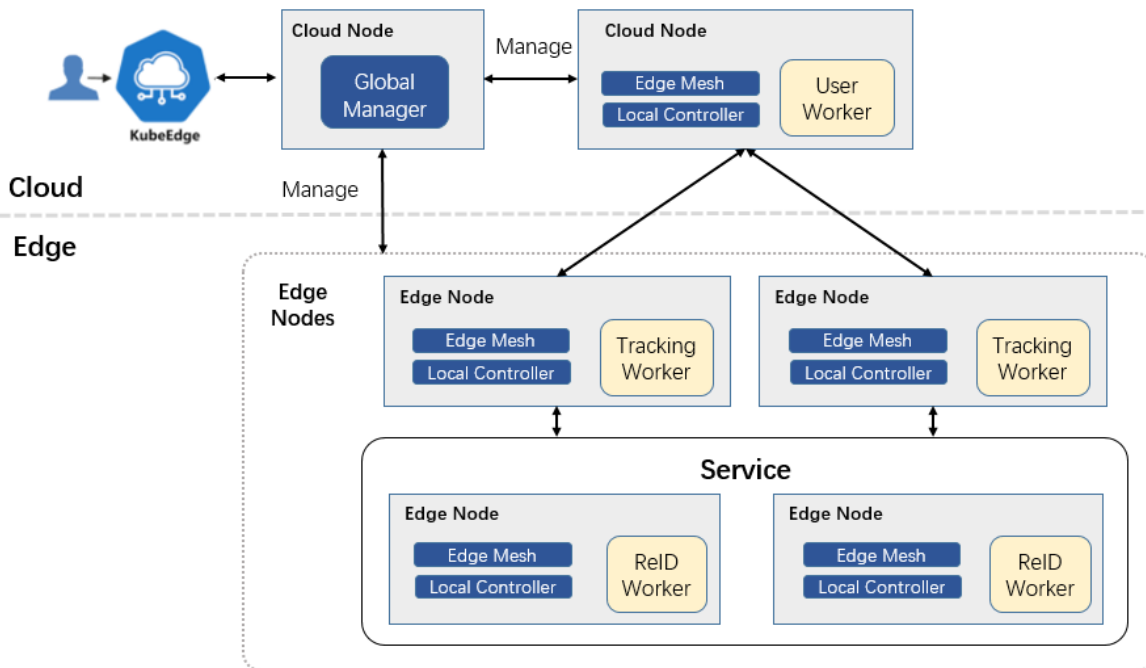
11.4.5 Flow of object search service creation

- The flow of object search service creation:



The object search service controller watches the creation of object search service crd in the cloud, syncs them to lc via the cloudhub-to-edgehub channel, and creates the workers on the edge nodes specified by the user.

- The components of object search service:

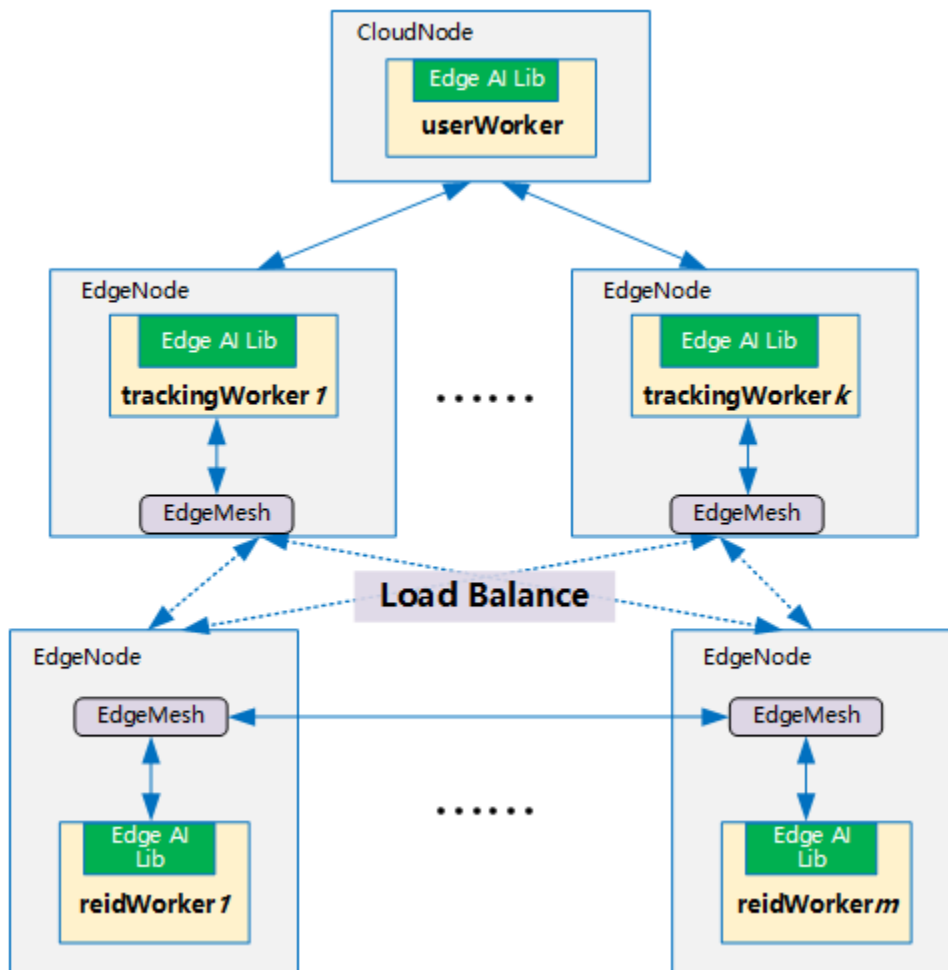


The object search service includes three types of workers: 1) User worker; 2) Tracking worker; 3) ReID worker. A user worker is used to provide API interface to users, and users can submit the object images to be searched through the API interface. There are usually multiple tracking workers and ReID workers, which can perform inference tasks.

of object search in parallel. Tracking workers are used to read camera data and perform object detection. Different tracking workers read data from different cameras. ReID worker is used for object feature extraction and matching to determine whether the detected object is the object to be searched.

The user worker, tracking workers, and ReID workers are started by the kubeedge at the edge nodes.

11.5 Workers Communication



- Object Tracking Service
 - Motivation
 - * Goals
 - Proposal
 - * Use Cases
 - Design Details
 - * CRD API Group and Version
 - * Object tracking service type definition
 - Validation

- * Object tracking service sample
- Controller Design
 - * Object tracking service Controller
 - * Downstream Controller
 - * Upstream Controller
 - * Details of api between GM(cloud) and LC(edge)
 - * Flow of object tracking service creation
- Workers Communication

OBJECT TRACKING SERVICE

12.1 Motivation

Object tracking is an important technology in the field of computer vision, which is widely used in security monitoring, intelligent transportation, etc. Generally, object tracking applications have high latency requirements, which cannot be met by cloud computing schemes. The object tracking schemes based on edge computing have the characteristics of low latency and data privacy security, and are the mainstream technology trend.

However, it is difficult for a single edge computing node to provide high-quality object tracking services. On the one hand, the coordination of multiple edge computing nodes is often required for across-camera object tracking. On the other hand, when there are large-scale tracking objects, a single computing node is difficult to handle. Multi-edge collaborative computing can accelerate large-scale object tracking applications and improve the tracking accuracy, which is the future development trend.

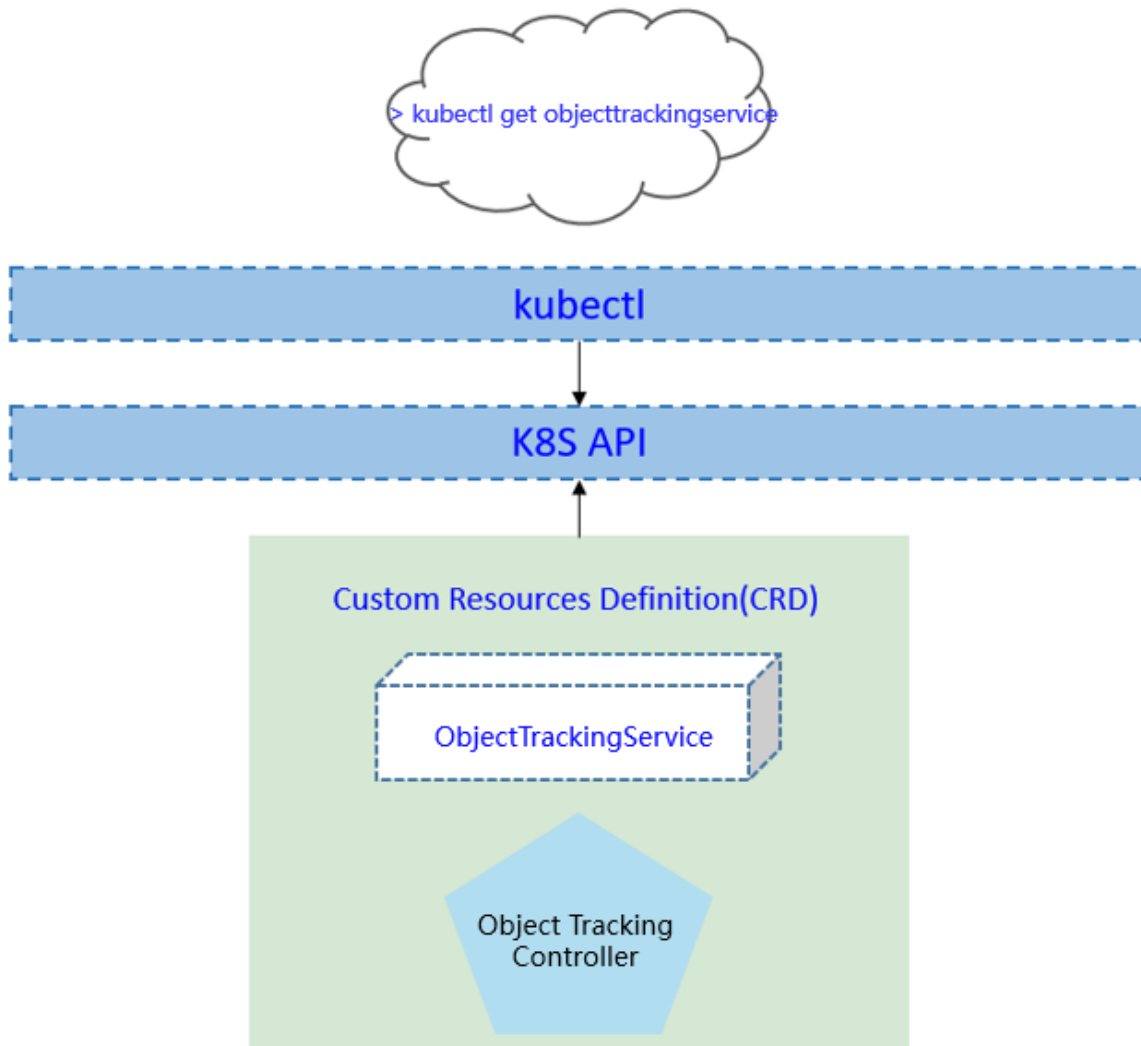
We propose the first open source end-to-end multi-edge collaborative object tracking solution. Based on KubeEdge's cloud-edge collaboration and resource management capabilities, we use multiple edge computing nodes to execute the AI inference tasks of object tracking in parallel. Our solution can not only reduce delay and improve throughput, but also will bring accuracy promotion.

12.1.1 Goals

- Support single/multi-object tracking
- Support across-camera object tracking
- Support parallel object re-identification(ReID)
- Support multi-camera data joint analysis and decision making

12.2 Proposal

We propose using Kubernetes Custom Resource Definitions (CRDs) to describe the object tracking service specification/status and a controller to synchronize these updates between edge and cloud.



12.2.1 Use Cases

- User can create typical multi-edge collaborative object tracking applications with providing AI models.

12.3 Design Details

12.3.1 CRD API Group and Version

The `ObjectTrackingService` CRD will be namespace-scoped. The tables below summarize the group, kind and API version details for the CRD.

- `ObjectTrackingService`

Field	Description
Group	sedna.io
APIVersion	v1alpha1
Kind	ObjectTrackingService

12.3.2 Object tracking service type definition

[go source](#)

Validation

[Open API v3 Schema based validation](#) can be used to guard against bad requests. Invalid values for fields (example string value for a boolean field etc) can be validated using this.

12.3.3 Object tracking service sample

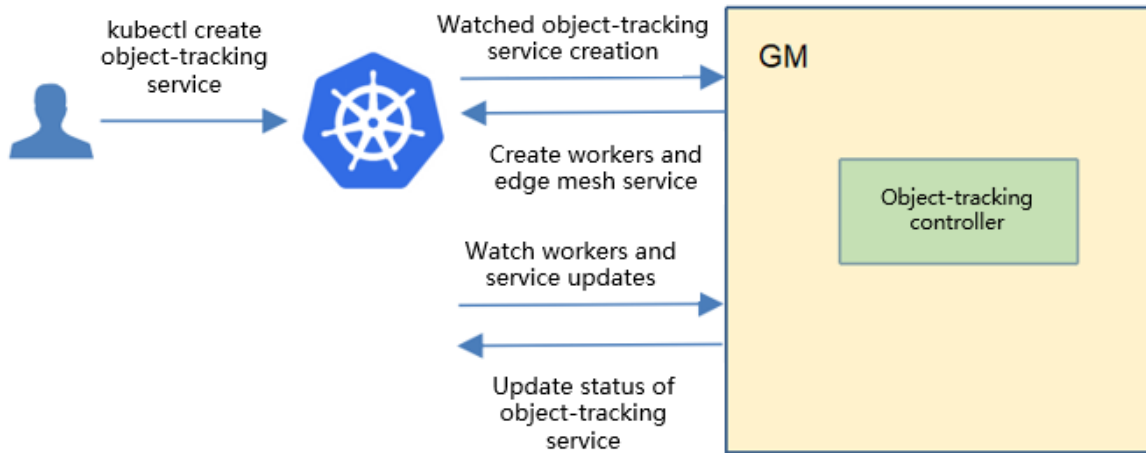
See the [source](#) for an example.

12.4 Controller Design

The object tracking service controller starts three separate goroutines called `upstream`, `downstream` and `object-tracking-service` controller. These are not separate controllers as such but named here for clarity.

- `object-tracking-service`: watch the updates of `object-tracking-service` task crds, and create the workers to complete the task.
- `downstream`: synchronize the `object-tracking-service` updates from the cloud to the edge node.
- `upstream`: synchronize the `object-tracking-service` updates from the edge to the cloud node.

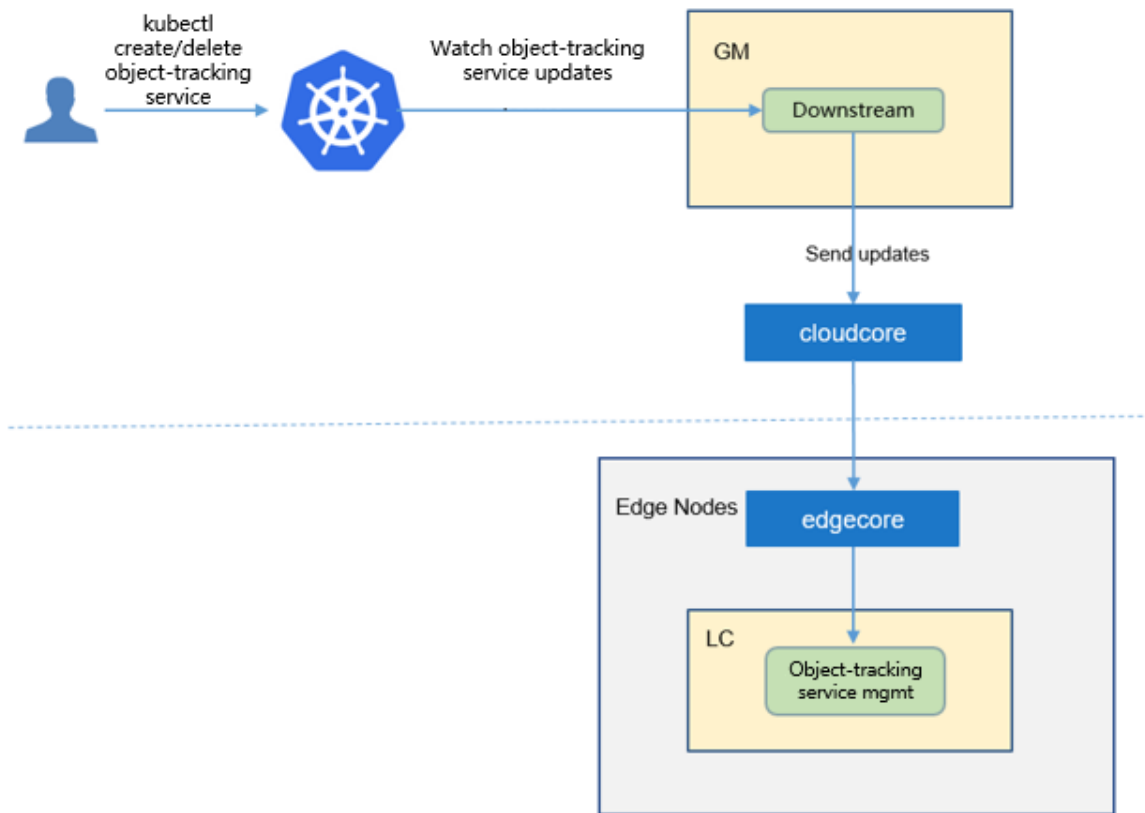
12.4.1 Object tracking service Controller



The object-tracking-service controller watches for the updates of object-tracking-service tasks and the corresponding pods against the K8S API server. Updates are categorized below along with the possible actions:

Update Type	Action
New Object-tracking-service Created	Create the cloud/edge worker
Object-tracking-service Deleted	NA. These workers will be deleted by GM.
The corresponding pod created/running/completed/failed	Update the status of object-tracking-service task.

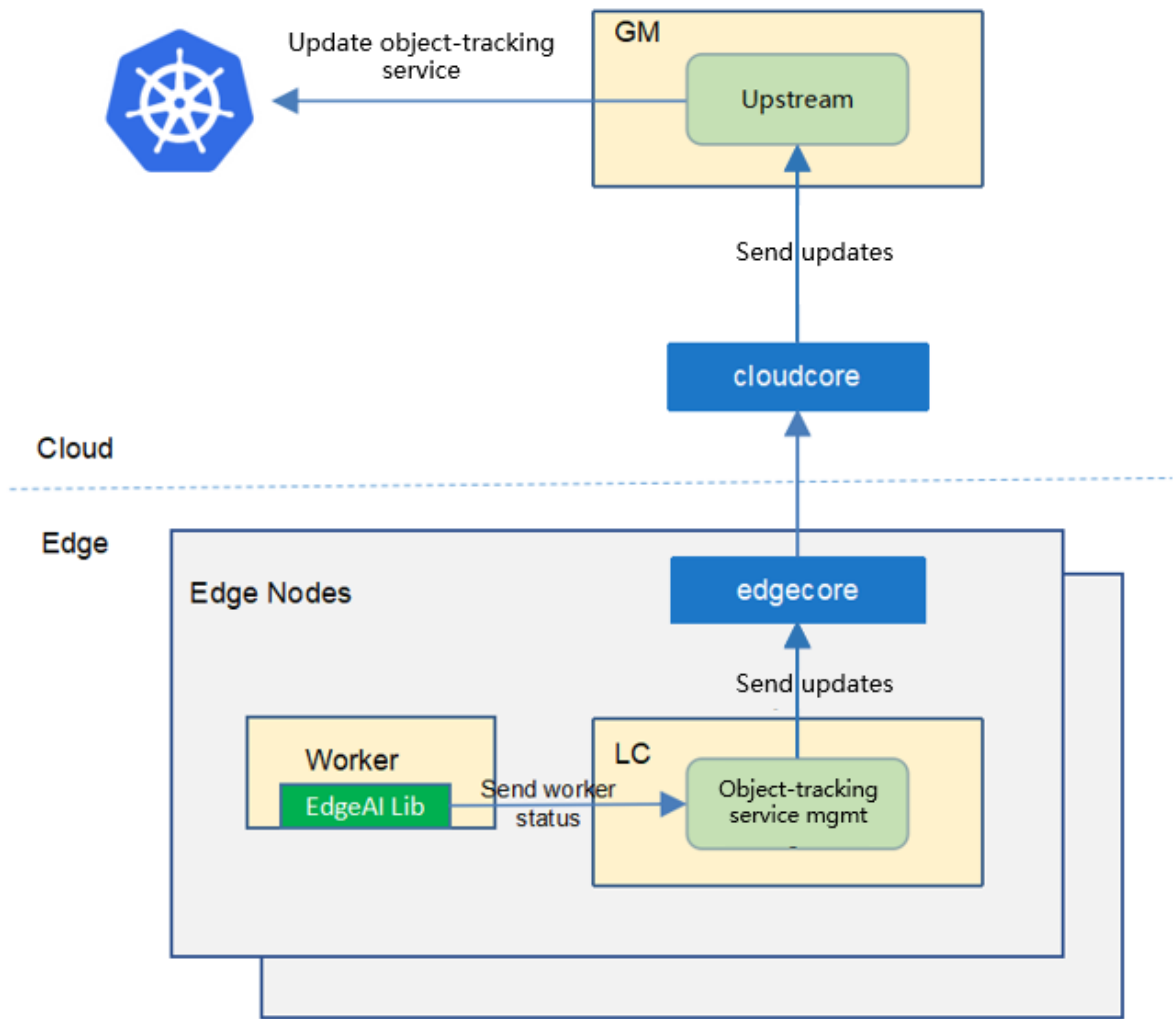
12.4.2 Downstream Controller



The downstream controller watches for object-tracking-service updates against the K8S API server. Updates are categorized below along with the possible actions that the downstream controller can take:

Update Type	Action
New Object-tracking-service Created	Sends the task information to LCs.
Object-tracking-service Deleted	The controller sends the delete event to LCs.

12.4.3 Upstream Controller



The upstream controller watches for object-tracking-service task updates from the edge node and applies these updates against the API server in the cloud. Updates are categorized below along with the possible actions that the upstream controller can take:

Update Type	Action
Object-tracking-service Reported State Updated	The controller appends the reported status of the object-tracking-service in the cloud.

12.4.4 Details of api between GM(cloud) and LC(edge)

1. GM(downstream controller) syncs the task info to LC:

```
// POST <namespace>/sedna/downstream/objecttrackingservices/<name>/insert
// body same to the task crd of k8s api, omitted here.
```

2. LC uploads the task status which reported by the worker to GM(upstream controller):

```
// POST <namespace>/sedna/upstream/objecttrackingservices/<name>/status

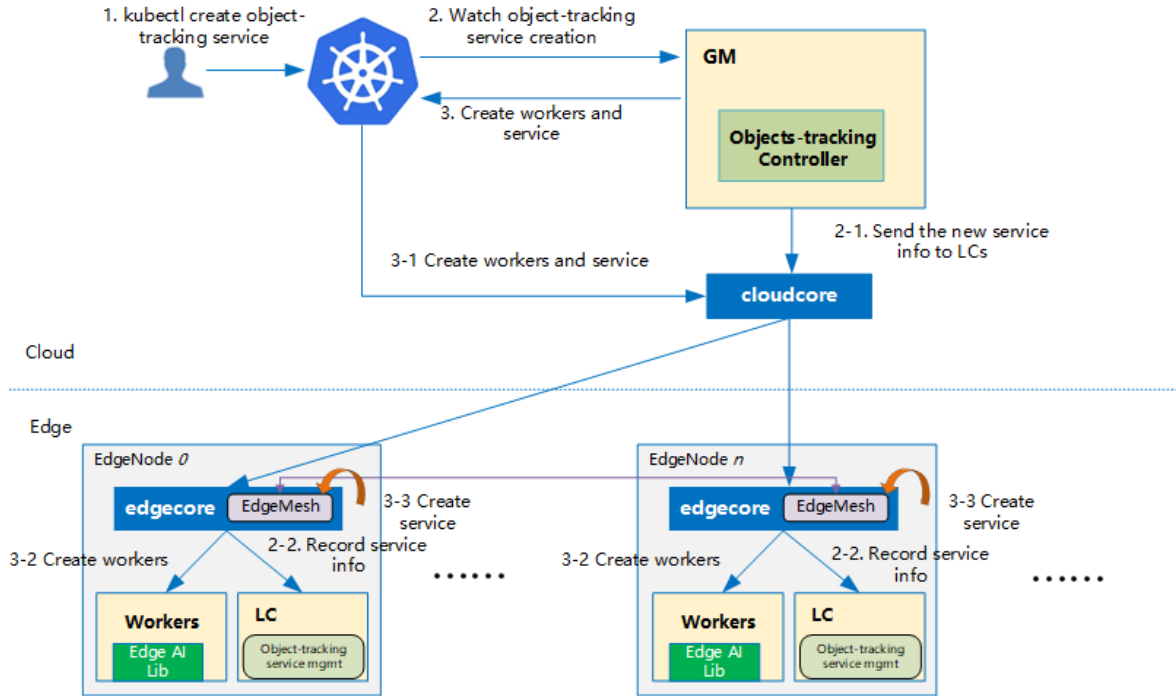
// ObjectTrackingServiceStatus defines status that send to GlobalManager
type ObjectTrackingServiceStatus struct {
    Phase string `json:"phase"`
    Status string `json:"status"`
    Output *Output `json:"output"`
}

// Output defines task output information
type Output struct {
    TaskInfo *TaskInfo `json:"taskInfo"`
}

// TaskInfo defines the task information
type TaskInfo struct {
    TrackingObjectNumber int    `json:"trackingObjectNumber"`
    FindUnknownObject    bool   `json:"findUnkownObject"`
    StartTime             string `json:"startTime"`
    CurrentTime           string `json:"currentTime"`
}
```

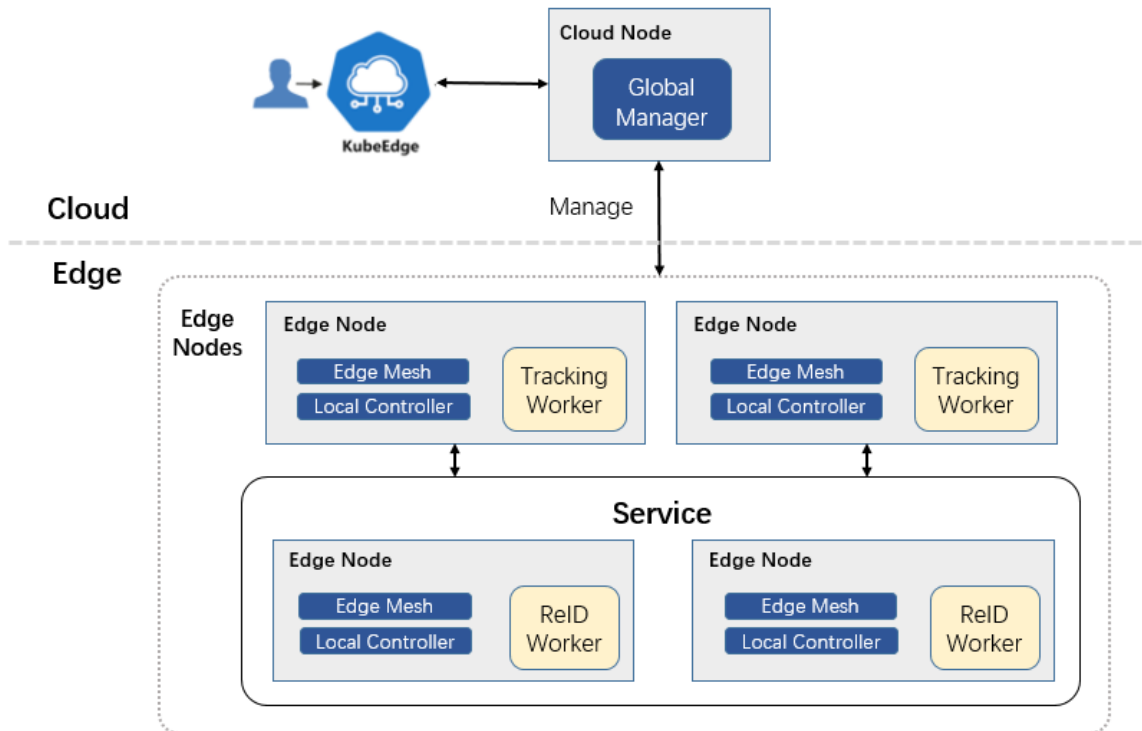
12.4.5 Flow of object tracking service creation

- The flow of object tracking service creation:



The object tracking service controller watches the creation of object tracking service crd in the cloud, syncs them to lc via the cloudhub-to-edgehub channel, and creates the inference workers on the edge nodes specified by the user.

- The components of object tracking service:

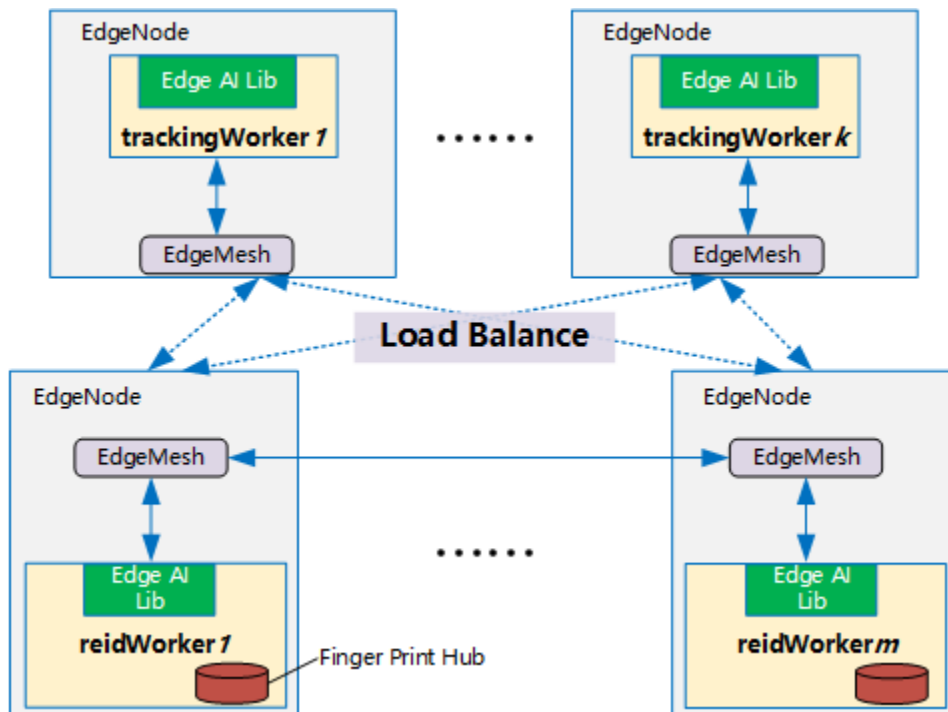


The object tracking service includes two types of workers: 1) Tracking worker; 2) ReID worker. There are usually multiple tracking workers and ReID workers, which can perform inference tasks of object tracking in parallel. Tracking

workers are used to read camera data and perform object detection and tracking. Different tracking workers read data from different cameras. ReID worker is used for object feature extraction and matching to determine the identity of the objects.

The tracking workers, and ReID workers are started by the kubeedge at the edge nodes.

12.5 Workers Communication



USING JOINT INFERENCE SERVICE IN HELMET DETECTION SCENARIO

This case introduces how to use joint inference service in helmet detection scenario. In the safety helmet detection scenario, the helmet detection shows lower performance due to limited resources in edge. However, the joint inference service can improve overall performance, which uploads hard examples that identified by the hard example mining algorithm to the cloud and infers them. The data used in the experiment is a video of workers wearing safety helmets. The joint inference service requires to detect the wearing of safety helmets in the video.

13.1 Helmet Detection Experiment

13.1.1 Install Sedna

Follow the [Sedna installation document](#) to install Sedna.

13.1.2 Prepare Data and Model

- step1: download [little model](#) to your edge node.

```
mkdir -p /data/little-model
cd /data/little-model
wget https://kubedge.obs.cn-north-1.myhuaweicloud.com/examples/helmet-detection-
↪inference/little-model.tar.gz
tar -zxvf little-model.tar.gz
```

- step2: download [big model](#) to your cloud node.

```
mkdir -p /data/big-model
cd /data/big-model
wget https://kubedge.obs.cn-north-1.myhuaweicloud.com/examples/helmet-detection-
↪inference/big-model.tar.gz
tar -zxvf big-model.tar.gz
```

13.1.3 Prepare Images

This example uses these images:

1. little model inference worker: `kubeedge/sedna-example-joint-inference-helmet-detection-little:v0.3.0`
2. big model inference worker: `kubeedge/sedna-example-joint-inference-helmet-detection-big:v0.3.0`

These images are generated by the script `build_images.sh`.

13.1.4 Create Joint Inference Service

Create Big Model Resource Object for Cloud

```
kubectl create -f - <<EOF
apiVersion: sedna.io/v1alpha1
kind: Model
metadata:
  name: helmet-detection-inference-big-model
  namespace: default
spec:
  url: "/data/big-model/yolov3_darknet.pb"
  format: "pb"
EOF
```

Create Little Model Resource Object for Edge

```
kubectl create -f - <<EOF
apiVersion: sedna.io/v1alpha1
kind: Model
metadata:
  name: helmet-detection-inference-little-model
  namespace: default
spec:
  url: "/data/little-model/yolov3_resnet18.pb"
  format: "pb"
EOF
```

Create JointInferenceService

Note the setting of the following parameters, which have to same as the script `little_model.py`:

- `hardExampleMining`: set hard example algorithm from {IBT, CrossEntropy} for inferring in edge side.
- `video_url`: set the url for video streaming.
- `all_examples_inference_output`: set your output path for the inference results.
- `hard_example_edge_inference_output`: set your output path for results of inferring hard examples in edge side.
- `hard_example_cloud_inference_output`: set your output path for results of inferring hard examples in cloud side.

Make preparation in edge node

```
mkdir -p /joint_inference/output
```

Create joint inference service

```
CLOUD_NODE="cloud-node-name"
EDGE_NODE="edge-node-name"

kubect1 create -f - <<EOF
apiVersion: sedna.io/v1alpha1
kind: JointInferenceService
metadata:
  name: helmet-detection-inference-example
  namespace: default
spec:
  edgeWorker:
    model:
      name: "helmet-detection-inference-little-model"
    hardExampleMining:
      name: "IBT"
      parameters:
        - key: "threshold_img"
          value: "0.9"
        - key: "threshold_box"
          value: "0.9"
    template:
      spec:
        nodeName: $EDGE_NODE
        containers:
          - image: kubeedge/sedna-example-joint-inference-helmet-detection-little:v0.3.0
            imagePullPolicy: IfNotPresent
            name: little-model
            env: # user defined environments
            - name: input_shape
              value: "416,736"
            - name: "video_url"
              value: "rtsp://localhost/video"
            - name: "all_examples_inference_output"
              value: "/data/output"
            - name: "hard_example_cloud_inference_output"
              value: "/data/hard_example_cloud_inference_output"
            - name: "hard_example_edge_inference_output"
              value: "/data/hard_example_edge_inference_output"
            resources: # user defined resources
              requests:
                memory: 64M
                cpu: 100m
              limits:
                memory: 2Gi
            volumeMounts:
              - name: outputdir
                mountPath: /data/
```

(continues on next page)

(continued from previous page)

```

    volumes:  # user defined volumes
      - name: outputdir
        hostPath:
          # user must create the directory in host
          path: /joint_inference/output
          type: Directory

cloudWorker:
  model:
    name: "helmet-detection-inference-big-model"
  template:
    spec:
      nodeName: $CLOUD_NODE
      containers:
        - image: kubeedge/sedna-example-joint-inference-helmet-detection-big:v0.3.0
          name: big-model
          imagePullPolicy: IfNotPresent
          env: # user defined environments
            - name: "input_shape"
              value: "544,544"
          resources: # user defined resources
            requests:
              memory: 2Gi
EOF

```

13.1.5 Check Joint Inference Status

```
kubectl get jointinferenceservices.sedna.io
```

13.1.6 Mock Video Stream for Inference in Edge Side

- step1: install the open source video streaming server [EasyDarwin](#).
- step2: start EasyDarwin server.
- step3: download [video](#).
- step4: push a video stream to the url (e.g., `rtsp://localhost/video`) that the inference service can connect.

```

wget https://github.com/EasyDarwin/EasyDarwin/releases/download/v8.1.0/EasyDarwin-linux-
↪8.1.0-1901141151.tar.gz
tar -zxvf EasyDarwin-linux-8.1.0-1901141151.tar.gz
cd EasyDarwin-linux-8.1.0-1901141151
./start.sh

mkdir -p /data/video
cd /data/video
wget https://kubeedge.obs.cn-north-1.myhuaweicloud.com/examples/helmet-detection-
↪inference/video.tar.gz
tar -zxvf video.tar.gz

```

(continues on next page)

(continued from previous page)

```
ffmpeg -re -i /data/video/video.mp4 -vcodec libx264 -f rtsp rtsp://localhost/video
```

13.1.7 Check Inference Result

You can check the inference results in the output path (e.g. /joint_inference/output) defined in the JointInferenceService config.

- the result of edge inference vs the result of joint inference



USING INCREMENTAL LEARNING JOB IN HELMET DETECTION SCENARIO

This document introduces how to use incremental learning job in helmet detection scenario. Using the incremental learning job, our application can automatically retrains, evaluates, and updates models based on the data generated at the edge.

14.1 Helmet Detection Experiment

14.1.1 Install Sedna

Follow the [Sedna installation document](#) to install Sedna.

14.1.2 Prepare Model

In this example, we need to prepare base model and deploy model in advance. download [models](#), including base model and deploy model.

```
cd /  
wget https://kubedge.obs.cn-north-1.myhuaweicloud.com/examples/helmet-detection/models.  
→tar.gz  
tar -zxvf models.tar.gz
```

14.1.3 Prepare for Inference Worker

In this example, we simulate a inference worker for helmet detection, the worker will upload hard examples to HE_SAVED_URL, while it inferences data from local video. We need to make following preparations:

- make sure following localdirs exist

```
mkdir -p /incremental_learning/video/  
mkdir -p /incremental_learning/he/  
mkdir -p /data/helmet_detection  
mkdir /output
```

- download [video](#), unzip video.tar.gz, and put it into /incremental_learning/video/

```
cd /incremental_learning/video/
wget https://kubedeege.obs.cn-north-1.myhuaweicloud.com/examples/helmet-detection/video.
tar.gz
tar -zxvf video.tar.gz
```

14.1.4 Prepare Image

This example uses the image:

```
kubedeege/sedna-example-incremental-learning-helmet-detection:v0.4.0
```

This image is generated by the script [build_images.sh](#), used for creating training, eval and inference worker.

14.1.5 Create Incremental Job

In this example, \$WORKER_NODE is a custom node, you can fill it which you actually run.

```
WORKER_NODE="edge-node"
```

Create Dataset

```
kubect1 create -f - <<EOF
apiVersion: sedna.io/v1alpha1
kind: Dataset
metadata:
  name: incremental-dataset
spec:
  url: "/data/helmet_detection/train_data/train_data.txt"
  format: "txt"
  nodeName: $WORKER_NODE
EOF
```

Create Initial Model to simulate the initial model in incremental learning scenario.

```
kubect1 create -f - <<EOF
apiVersion: sedna.io/v1alpha1
kind: Model
metadata:
  name: initial-model
spec:
  url : "/models/base_model"
  format: "ckpt"
EOF
```

Create Deploy Model

```
kubect1 create -f - <<EOF
apiVersion: sedna.io/v1alpha1
kind: Model
metadata:
  name: deploy-model
```

(continues on next page)

(continued from previous page)

```
spec:
  url : "/models/deploy_model/saved_model.pb"
  format: "pb"
EOF
```

Start The Incremental Learning Job

- incremental learning supports hot model updates and cold model updates. Job support cold model updates default. If you want to use hot model updates, please to add the following fields:

```
deploySpec:
  model:
    hotUpdateEnabled: true
    pollPeriodSeconds: 60 # default value is 60
```

- create the job:

```
IMAGE=kubeedge/sedna-example-incremental-learning-helmet-detection:v0.4.0

kubectrl create -f - <<EOF
apiVersion: sedna.io/v1alpha1
kind: IncrementalLearningJob
metadata:
  name: helmet-detection-demo
spec:
  initialModel:
    name: "initial-model"
  dataset:
    name: "incremental-dataset"
    trainProb: 0.8
  trainSpec:
    template:
      spec:
        nodeName: $WORKER_NODE
        containers:
          - image: $IMAGE
            name: train-worker
            imagePullPolicy: IfNotPresent
            args: ["train.py"]
            env:
              - name: "batch_size"
                value: "32"
              - name: "epochs"
                value: "1"
              - name: "input_shape"
                value: "352,640"
              - name: "class_names"
                value: "person,helmet,helmet-on,helmet-off"
              - name: "nms_threshold"
                value: "0.4"
              - name: "obj_threshold"
                value: "0.3"
      trigger:
```

(continues on next page)

(continued from previous page)

```
checkPeriodSeconds: 60
timer:
  start: 02:00
  end: 20:00
condition:
  operator: ">"
  threshold: 500
  metric: num_of_samples
evalSpec:
  template:
    spec:
      nodeName: $WORKER_NODE
      containers:
        - image: $IMAGE
          name: eval-worker
          imagePullPolicy: IfNotPresent
          args: ["eval.py"]
          env:
            - name: "input_shape"
              value: "352,640"
            - name: "class_names"
              value: "person,helmet,helmet-on,helmet-off"
deploySpec:
  model:
    name: "deploy-model"
    hotUpdateEnabled: true
    pollPeriodSeconds: 60
  trigger:
    condition:
      operator: ">"
      threshold: 0.1
      metric: precision_delta
hardExampleMining:
  name: "IBT"
  parameters:
    - key: "threshold_img"
      value: "0.9"
    - key: "threshold_box"
      value: "0.9"
template:
  spec:
    nodeName: $WORKER_NODE
    containers:
      - image: $IMAGE
        name: infer-worker
        imagePullPolicy: IfNotPresent
        args: ["inference.py"]
        env:
          - name: "input_shape"
            value: "352,640"
          - name: "video_url"
            value: "file://video/video.mp4"
```

(continues on next page)

(continued from previous page)

```

    - name: "HE_SAVED_URL"
      value: "/he_saved_url"
  volumeMounts:
    - name: localvideo
      mountPath: /video/
    - name: hedir
      mountPath: /he_saved_url
  resources: # user defined resources
    limits:
      memory: 2Gi
  volumes: # user defined volumes
    - name: localvideo
      hostPath:
        path: /incremental_learning/video/
        type: DirectoryOrCreate
    - name: hedir
      hostPath:
        path: /incremental_learning/he/
        type: DirectoryOrCreate
  outputDir: "/output"
EOF

```

1. The Dataset describes data with labels and HE_SAVED_URL indicates the address of the deploy container for uploading hard examples. Users will mark label for the hard examples in the address.
2. Ensure that the path of outputDir in the YAML file exists on your node. This path will be directly mounted to the container.

14.1.6 Check Incremental Learning Job

Query the service status:

```
kubectl get incrementallearningjob helmet-detection-demo
```

In the IncrementalLearningJob resource helmet-detection-demo, the following trigger is configured:

```

trigger:
  checkPeriodSeconds: 60
  timer:
    start: 02:00
    end: 20:00
  condition:
    operator: ">"
    threshold: 500
    metric: num_of_samples

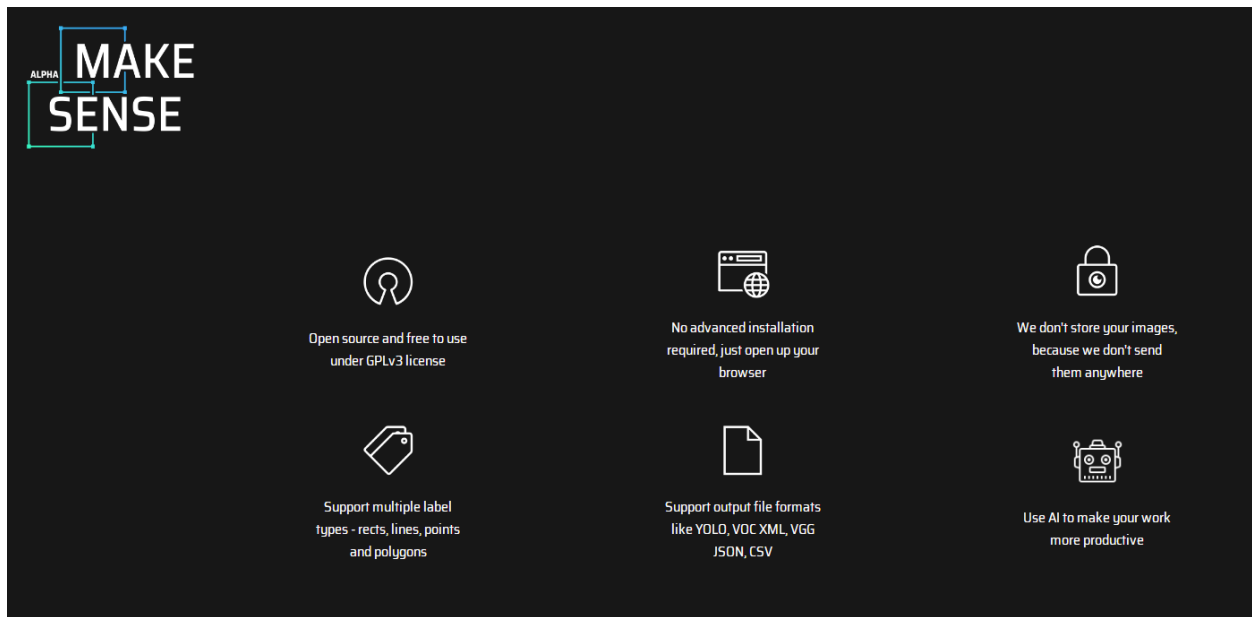
```

14.1.7 Hard Example Labeling

In a real word, we need to label the hard examples in `HE_SAVED_URL` with annotation tools and then put the examples to Dataset's url.

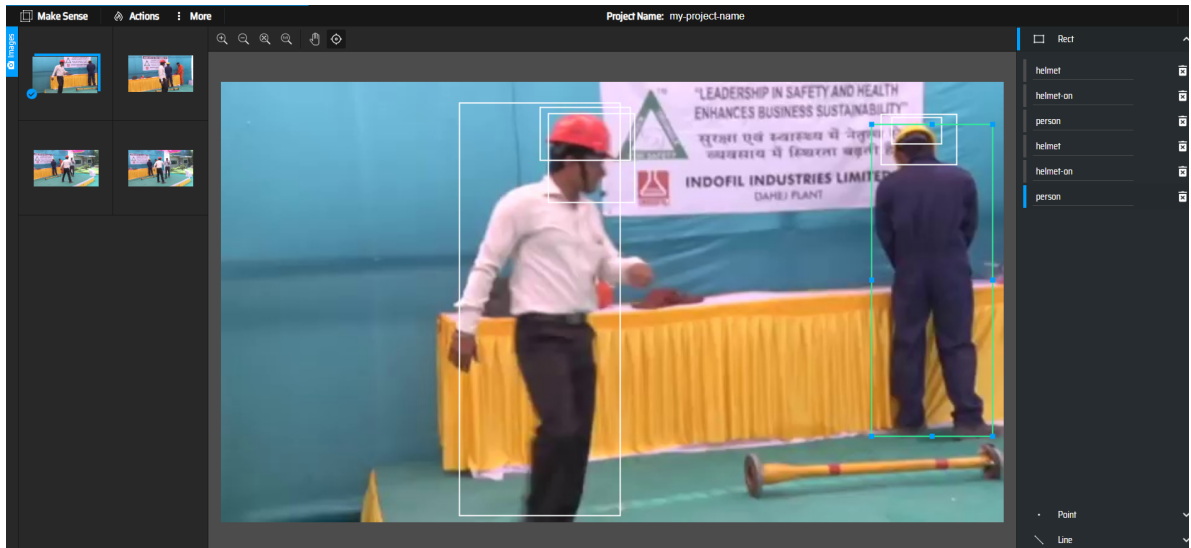
You can use Open-Source annotation tools to label hard examples, such as [MAKE SENSE](#), which has following main advantages:

- Open source and free to use under GPLv3 license
- Support outputfile formats like YOLO, VOC XML, VGG JSON, CSV
- No advanced installation required, just open up your browser
- Use AI to make your work more productive
- Offline running as a container, ensuring data security

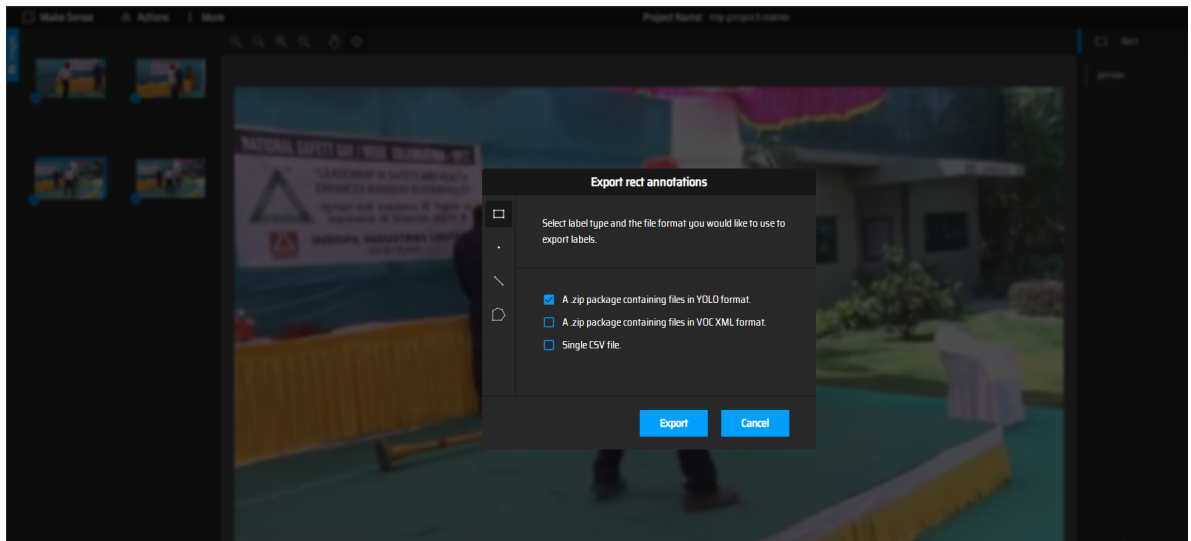


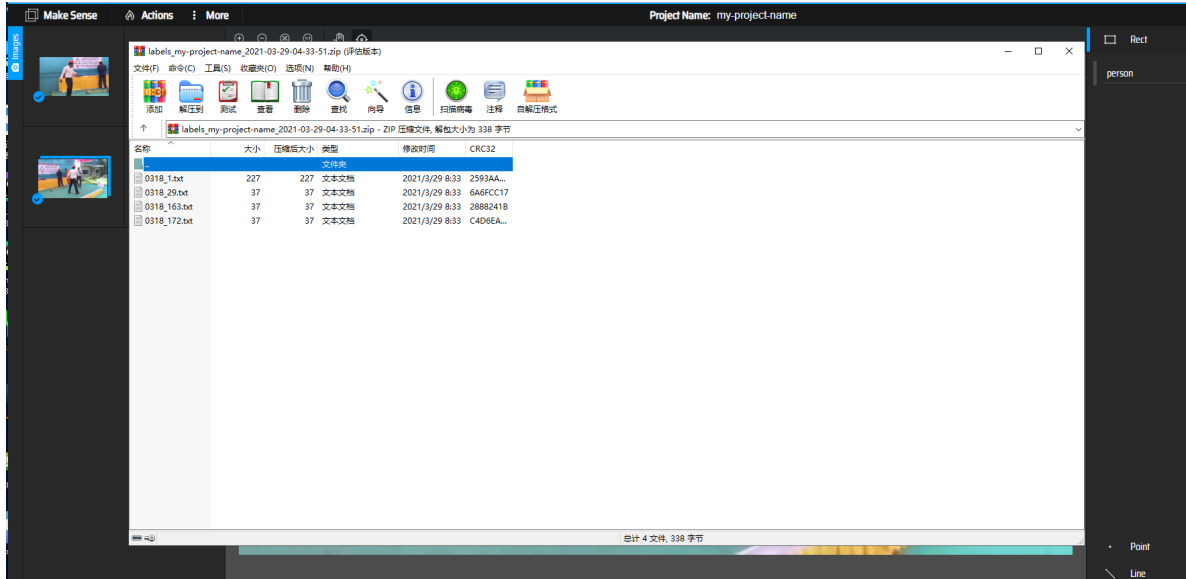
the details labeling are not described here, main steps in this demo are as follows:

- import unlabeled hard example to annotation tools



- label and export annotations





- you will get YOLO format annotations, so you need convert them to the type which can be used by your own training code. in this example, the following scripts are provided for reference:

```
import os

annotation_dir_path = "C:/Users/Administrator/Desktop/labeled_data"
save_path = "C:/Users/Administrator/Desktop/labeled_data/save_label.txt"

def convert_single_line(line):
    line_list = []
    line = line.split(" ")
    for i in range(1, len(line)):
        line[i] = float(line[i])
        line[i] = line[i] * 1000
        line_list.append(str(int(line[i])))
    line_list.append(line[0])
    return ",".join(line_list)

if __name__ == '__main__':
    results = []
    g = os.walk(annotation_dir_path)
    for path, dir_list, file_list in g:
        for file_name in file_list:
            file_path = os.path.join(path, file_name)
            file_name = file_name.split(".txt")
            file_name = file_name[0] + '.jpg'
            single_label_string = file_name
            f = open(file_path)
            lines = f.readlines()
            for line in lines:
                line = line.strip('\n')
                single_label_string = single_label_string + " " + convert_single_
                line(line)
                results.append(single_label_string)
            save_file = open(save_path, "w")
```

(continues on next page)

(continued from previous page)






```

for result in results:
    save_file.write(result + "\n")
save_file.close()

```

How to use: `annotation_dir_path`: location for labeled annotations from MAKESENSE `save_path`: location for label txt which converted from annotations

- run above script, you can get a txt which includes all label information
- put the text with examples in the same dir
- you will get labeled examples which meet training requirements

 0318_1.jpg	2021/3/29 16:00	JPG 文件	52 KB
 0318_29.jpg	2021/3/29 16:00	JPG 文件	57 KB
 0318_163.jpg	2021/3/29 16:00	JPG 文件	69 KB
 0318_172.jpg	2021/3/29 16:00	JPG 文件	69 KB
 save_label.txt	2021/3/30 11:52	文本文档	1 KB

- put these examples and annotations above to Dataset's url

Without annotation tools, we can simulate the condition of `num_of_samples` in the following ways: Download `dataset` to `$WORKER_NODE`.

```

cd /data/helmet_detection
wget https://kubedeege.obs.cn-north-1.myhuaweicloud.com/examples/helmet-detection/
dataset.tar.gz
tar -zxvf dataset.tar.gz

```

The LocalController component will check the number of the sample, realize trigger conditions are met and notice the GlobalManager Component to start train worker. When the train worker finish, we can view the updated model in the `/output` directory in `$WORKER_NODE` node. Then the eval worker will start to evaluate the model that train worker generated.

If the eval result satisfy the `deploySpec`'s trigger

```

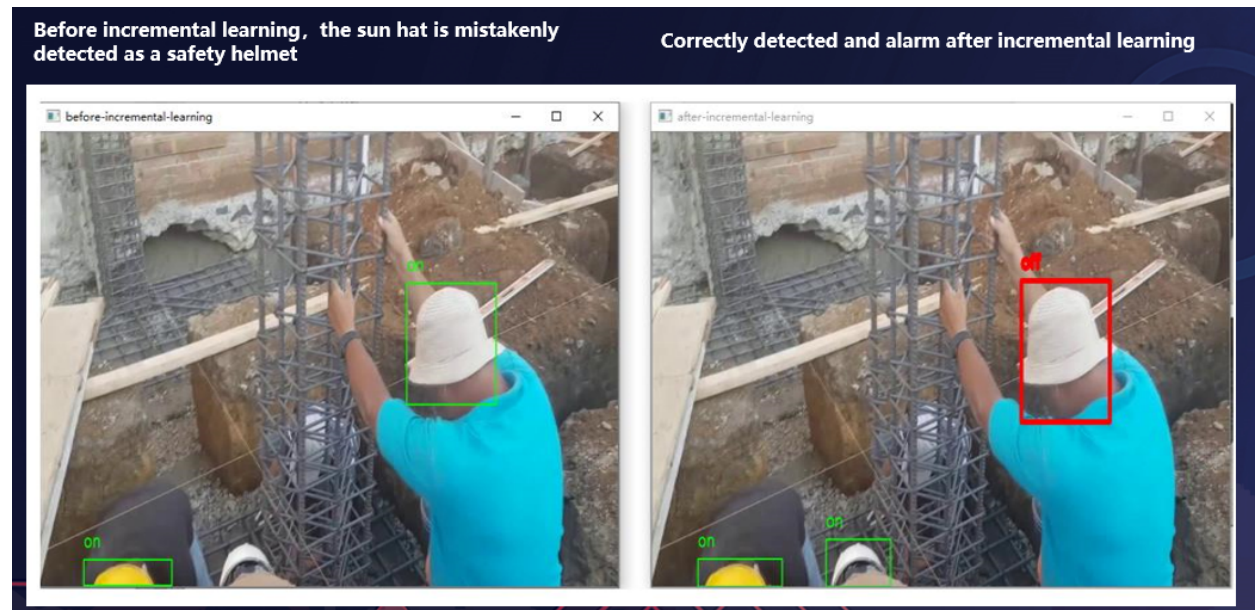
trigger:
  condition:
    operator: ">"
    threshold: 0.1
    metric: precision_delta

```

the deploy worker will load the new model and provide service.

14.1.8 Effect Display

In this example, false and failed detections occur at stage of inference before incremental learning, after incremental learning, all targets are correctly detected.



USING FEDERATED LEARNING JOB IN SURFACE DEFECT DETECTION SCENARIO

This case introduces how to use federated learning job in surface defect detection scenario. In the safety surface defect detection, data is scattered in different places (such as server node, camera or others) and cannot be aggregated due to data privacy and bandwidth. As a result, we cannot use all the data for training. Using Federated Learning, we can solve the problem. Each place uses its own data for model training ,uploads the weight to the cloud for aggregation, and obtains the aggregation result for model update.

15.1 Surface Defect Detection Experiment

Assume that there are two edge nodes and a cloud node. Data on the edge nodes cannot be migrated to the cloud due to privacy issues. Base on this scenario, we will demonstrate the surface inspection.

15.1.1 Prepare Nodes

```
CLOUD_NODE="cloud-node-name"  
EDGE1_NODE="edge1-node-name"  
EDGE2_NODE="edge2-node-name"
```

15.1.2 Install Sedna

Follow the [Sedna installation document](#) to install Sedna.

15.1.3 Prepare Dataset

Download [dataset](#) and the [label file](#) to /data of EDGE1_NODE.

```
mkdir -p /data  
cd /data  
git clone https://github.com/abin24/Magnetic-tile-defect-datasets.git Magnetic-tile-  
↪defect-datasets  
curl -o 1.txt https://raw.githubusercontent.com/kubeedge/sedna/main/examples/federated_  
↪learning/surface_defect_detection/data/1.txt
```

Download [dataset](#) and the [label file](#) to /data of EDGE2_NODE.

```
mkdir -p /data
cd /data
git clone https://github.com/abin24/Magnetic-tile-defect-datasets..git Magnetic-tile-
↪defect-datasets
curl -o 2.txt https://raw.githubusercontent.com/kubeedge/sedna/main/examples/federated_
↪learning/surface_defect_detection/data/2.txt
```

15.1.4 Prepare Images

This example uses these images:

1. aggregation worker: kubeedge/sedna-example-federated-learning-surface-defect-detection-aggregation:v0.3.0
2. train worker: kubeedge/sedna-example-federated-learning-surface-defect-detection-train:v0.3.0

These images are generated by the script [build_images.sh](#).

15.1.5 Create Federated Learning Job

Create Dataset

create dataset for \$EDGE1_NODE

```
kubectl create -f - <<EOF
apiVersion: sedna.io/v1alpha1
kind: Dataset
metadata:
  name: "edge1-surface-defect-detection-dataset"
spec:
  url: "/data/1.txt"
  format: "txt"
  nodeName: $EDGE1_NODE
EOF
```

create dataset for \$EDGE2_NODE

```
kubectl create -f - <<EOF
apiVersion: sedna.io/v1alpha1
kind: Dataset
metadata:
  name: "edge2-surface-defect-detection-dataset"
spec:
  url: "/data/2.txt"
  format: "txt"
  nodeName: $EDGE2_NODE
EOF
```

Create Model

create the directory /model in the host of \$EDGE1_NODE

```
mkdir /model
```

create the directory /model in the host of \$EDGE2_NODE

```
mkdir /model
```

create model

```
kubectl create -f - <<EOF
apiVersion: sedna.io/v1alpha1
kind: Model
metadata:
  name: "surface-defect-detection-model"
spec:
  url: "/model"
  format: "pb"
EOF
```

Start Federated Learning Job

```
kubectl create -f - <<EOF
apiVersion: sedna.io/v1alpha1
kind: FederatedLearningJob
metadata:
  name: surface-defect-detection
spec:
  aggregationWorker:
    model:
      name: "surface-defect-detection-model"
    template:
      spec:
        nodeName: $CLOUD_NODE
        containers:
          - image: kubeedge/sedna-example-federated-learning-surface-defect-detection-
↪ aggregation:v0.3.0
            name: agg-worker
            imagePullPolicy: IfNotPresent
            env: # user defined environments
              - name: "exit_round"
                value: "3"
            resources: # user defined resources
              limits:
                memory: 2Gi
  trainingWorkers:
    - dataset:
        name: "edge1-surface-defect-detection-dataset"
      template:
        spec:
```

(continues on next page)

(continued from previous page)

```

        nodeName: $EDGE1_NODE
        containers:
          - image: kubeedge/sedna-example-federated-learning-surface-defect-detection-
↪train:v0.3.0
            name: train-worker
            imagePullPolicy: IfNotPresent
            env: # user defined environments
              - name: "batch_size"
                value: "32"
              - name: "learning_rate"
                value: "0.001"
              - name: "epochs"
                value: "2"
            resources: # user defined resources
              limits:
                memory: 2Gi
    - dataset:
        name: "edge2-surface-defect-detection-dataset"
    template:
      spec:
        nodeName: $EDGE2_NODE
        containers:
          - image: kubeedge/sedna-example-federated-learning-surface-defect-detection-
↪train:v0.3.0
            name: train-worker
            imagePullPolicy: IfNotPresent
            env: # user defined environments
              - name: "batch_size"
                value: "32"
              - name: "learning_rate"
                value: "0.001"
              - name: "epochs"
                value: "2"
            resources: # user defined resources
              limits:
                memory: 2Gi
EOF

```

15.1.6 Check Federated Learning Status

```
kubect1 get federatedlearningjob surface-defect-detection
```

15.1.7 Check Federated Learning Train Result

After the job completed, you will find the model generated on the directory `/model` in `$EDGE1_NODE` and `$EDGE2_NODE`.

COLLABORATIVELY TRAIN YOLO-V5 USING MISTNET ON COCO128 DATASET

This case introduces how to train a federated learning job with an aggregation algorithm named MistNet in MNIST handwritten digit classification scenario. Data is scattered in different places (such as edge nodes, cameras, and others) and cannot be aggregated at the server due to data privacy and bandwidth. As a result, we cannot use all the data for training. In some cases, edge nodes have limited computing resources and even have no training capability. The edge cannot gain the updated weights from the training process. Therefore, traditional algorithms (e.g., federated average), which usually aggregate the updated weights trained by different edge clients, cannot work in this scenario. MistNet is proposed to address this issue.

MistNet partitions a DNN model into two parts, a lightweight feature extractor at the edge side to generate meaningful features from the raw data, and a classifier including the most model layers at the cloud to be iteratively trained for specific tasks. MistNet achieves acceptable model utility while greatly reducing privacy leakage from the released intermediate features.

16.1 Object Detection Experiment

Assume that there are two edge nodes and a cloud node. Data on the edge nodes cannot be migrated to the cloud due to privacy issues. Base on this scenario, we will demonstrate the mnist example.

16.1.1 Prepare Nodes

```
CLOUD_NODE="cloud-node-name"  
EDGE1_NODE="edge1-node-name"  
EDGE2_NODE="edge2-node-name"
```

16.1.2 Install Sedna

Follow the [Sedna installation document](#) to install Sedna.

16.1.3 Prepare Dataset

Download [dataset](#)

Create data interface for EDGE1_NODE.

```
mkdir -p /data/1
cd /data/1
wget https://github.com/ultralytics/yolov5/releases/download/v1.0/coco128.zip
unzip coco128.zip -d COCO
```

Create data interface for EDGE2_NODE.

```
mkdir -p /data/2
cd /data/2
wget https://github.com/ultralytics/yolov5/releases/download/v1.0/coco128.zip
unzip coco128.zip -d COCO
```

16.1.4 Prepare Images

This example uses these images:

1. aggregation worker: `kubeedge/sedna-example-federated-learning-mistnet-yolo-aggregator:v0.4.0`
2. train worker: `kubeedge/sedna-example-federated-learning-mistnet-yolo-client:v0.4.0`

These images are generated by the script [build_images.sh](#).

16.1.5 Create Federated Learning Job

Create Dataset

create dataset for \$EDGE1_NODE and \$EDGE2_NODE

```
kubectl create -f - <<EOF
apiVersion: sedna.io/v1alpha1
kind: Dataset
metadata:
  name: "coco-dataset-1"
spec:
  url: "/data/1/COCO"
  format: "dir"
  nodeName: $EDGE1_NODE
EOF
```

```
kubectl create -f - <<EOF
apiVersion: sedna.io/v1alpha1
kind: Dataset
metadata:
  name: "coco-dataset-2"
spec:
  url: "/data/2/COCO"
```

(continues on next page)

(continued from previous page)

```
format: "dir"
nodeName: $EDGE2_NODE
EOF
```

Create Model

create the directory /model and /pretrained in \$EDGE1_NODE and \$EDGE2_NODE.

```
mkdir -p /model
mkdir -p /pretrained
```

create the directory /model and /pretrained in the host of \$CLOUD_NODE (download links [here](#))

```
# on the cloud side
mkdir -p /model
mkdir -p /pretrained
cd /pretrained
wget https://kubedge.obs.cn-north-1.myhuaweicloud.com/examples/yolov5_coco128_mistnet/
↪ yolov5.pth
```

create model

```
kubectl create -f - <<EOF
apiVersion: sedna.io/v1alpha1
kind: Model
metadata:
  name: "yolo-v5-model"
spec:
  url: "/model/yolov5.pth"
  format: "pth"
EOF

kubectl create -f - <<EOF
apiVersion: sedna.io/v1alpha1
kind: Model
metadata:
  name: "yolo-v5-pretrained-model"
spec:
  url: "/pretrained/yolov5.pth"
  format: "pth"
EOF
```

16.1.6 Create a secret with your S3 user credential. (Optional)

```
kubect1 create -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
  annotations:
    s3-endpoint: s3.amazonaws.com
    s3-usehttps: "1"
stringData:
  ACCESS_KEY_ID: XXXX
  SECRET_ACCESS_KEY: XXXXXXXX
EOF
```

Start Federated Learning Job

```
kubect1 create -f - <<EOF
apiVersion: sedna.io/v1alpha1
kind: FederatedLearningJob
metadata:
  name: yolo-v5
spec:
  pretrainedModel: # option
    name: "yolo-v5-pretrained-model"
  transmitter: # option
    ws: { } # option, by default
    s3: # optional, but at least one
      aggDataPath: "s3://sedna/fl/aggregation_data"
      credentialName: mysecret
  aggregationWorker:
    model:
      name: "yolo-v5-model"
    template:
      spec:
        nodeName: $CLOUD_NODE
        containers:
          - image: kubeedge/sedna-example-federated-learning-mistnet-yolo-aggregator:v0.
↪ 4.0
            name: agg-worker
            imagePullPolicy: IfNotPresent
            env: # user defined environments
              - name: "cut_layer"
                value: "4"
              - name: "epsilon"
                value: "100"
              - name: "aggregation_algorithm"
                value: "mistnet"
              - name: "batch_size"
                value: "32"
              - name: "epochs"
```

(continues on next page)

(continued from previous page)

```

        value: "100"
        resources: # user defined resources
        limits:
            memory: 8Gi
trainingWorkers:
- dataset:
    name: "coco-dataset-1"
template:
  spec:
    nodeName: $EDGE1_NODE
    containers:
      - image: kubernetes/sedna-example-federated-learning-mistnet-yolo-client:v0.4.0
        name: train-worker
        imagePullPolicy: IfNotPresent
        args: [ "-i", "1" ]
        env: # user defined environments
          - name: "cut_layer"
            value: "4"
          - name: "epsilon"
            value: "100"
          - name: "aggregation_algorithm"
            value: "mistnet"
          - name: "batch_size"
            value: "32"
          - name: "learning_rate"
            value: "0.001"
          - name: "epochs"
            value: "1"
        resources: # user defined resources
        limits:
            memory: 2Gi
- dataset:
    name: "coco-dataset-2"
template:
  spec:
    nodeName: $EDGE2_NODE
    containers:
      - image: kubernetes/sedna-example-federated-learning-mistnet-yolo-client:v0.4.0
        name: train-worker
        imagePullPolicy: IfNotPresent
        args: [ "-i", "2" ]
        env: # user defined environments
          - name: "cut_layer"
            value: "4"
          - name: "epsilon"
            value: "100"
          - name: "aggregation_algorithm"
            value: "mistnet"
          - name: "batch_size"
            value: "32"
          - name: "learning_rate"
            value: "0.001"

```

(continues on next page)

(continued from previous page)

```
- name: "epochs"
  value: "1"
resources: # user defined resources
limits:
  memory: 2Gi
```

EOF

USING LIFELONG LEARNING JOB IN THERMAL COMFORT PREDICTION SCENARIO

This document introduces how to use lifelong learning job in thermal comfort prediction scenario. Using the lifelong learning job, our application can automatically retrain, evaluate, and update models based on the data generated at the edge.

17.1 Thermal Comfort Prediction Experiment

17.1.1 Install Sedna

Follow the [Sedna installation document](#) to install Sedna.

17.1.2 Prepare Dataset

In this example, you can use [ASHRAE Global Thermal Comfort Database II](#) to initial lifelong learning job.

We provide a well-processed [datasets](#), including train (`trainData.csv`), evaluation (`testData.csv`) and incremental (`trainData2.csv`) dataset.

```
cd /data
wget https://kubedge.obs.cn-north-1.myhuaweicloud.com/examples/atcii-classifier/dataset.
↪tar.gz
tar -zxvf dataset.tar.gz
```

17.1.3 Create Lifelong Job

In this example, `$WORKER_NODE` is a custom node, you can fill it which you actually run.

```
WORKER_NODE="edge-node"
```

Create Dataset

```
kubect1 create -f - <<EOF
apiVersion: sedna.io/v1alpha1
kind: Dataset
metadata:
  name: lifelong-dataset
spec:
```

(continues on next page)

(continued from previous page)

```
url: "/data/trainData.csv"
format: "csv"
nodeName: $WORKER_NODE
EOF
```

Also, you can replace `trainData.csv` with `trainData2.csv` which contained in dataset to trigger retraining.

Start The Lifelong Learning Job

```
kubect1 create -f - <<EOF
apiVersion: sedna.io/v1alpha1
kind: LifelongLearningJob
metadata:
  name: atcii-classifier-demo
spec:
  dataset:
    name: "lifelong-dataset"
    trainProb: 0.8
  trainSpec:
    template:
      spec:
        nodeName: $WORKER_NODE
        containers:
          - image: kubeedge/sedna-example-lifelong-learning-atcii-classifier:v0.3.0
            name: train-worker
            imagePullPolicy: IfNotPresent
            args: ["train.py"] # training script
            env: # Hyperparameters required for training
              - name: "early_stopping_rounds"
                value: "100"
              - name: "metric_name"
                value: "mlogloss"
      trigger:
        checkPeriodSeconds: 60
        timer:
          start: 02:00
          end: 24:00
        condition:
          operator: ">"
          threshold: 500
          metric: num_of_samples
    evalSpec:
      template:
        spec:
          nodeName: $WORKER_NODE
          containers:
            - image: kubeedge/sedna-example-lifelong-learning-atcii-classifier:v0.3.0
              name: eval-worker
              imagePullPolicy: IfNotPresent
              args: ["eval.py"]
              env:
                - name: "metrics"
                  value: "precision_score"
```

(continues on next page)

(continued from previous page)

```

      - name: "metric_param"
        value: '{"average': 'micro'}"
      - name: "model_threshold" # Threshold for filtering deploy models
        value: "0.5"
deploySpec:
  template:
    spec:
      nodeName: $WORKER_NODE
      containers:
      - image: kubernetes/sedna-example-lifelong-learning-atcii-classifier:v0.3.0
        name: infer-worker
        imagePullPolicy: IfNotPresent
        args: ["inference.py"]
        env:
        - name: "UT_SAVED_URL" # unseen tasks save path
          value: "/ut_saved_url"
        - name: "infer_dataset_url" # simulation of the inference samples
          value: "/data/testData.csv"
        volumeMounts:
        - name: utdir
          mountPath: /ut_saved_url
        - name: inferdata
          mountPath: /data/
        resources: # user defined resources
          limits:
            memory: 2Gi
        volumes: # user defined volumes
        - name: utdir
          hostPath:
            path: /lifelong/unseen_task/
            type: DirectoryOrCreate
        - name: inferdata
          hostPath:
            path: /data/
            type: DirectoryOrCreate
      outputDir: "/output"
EOF

```

Note: outputDir can be set as s3 storage url to save artifacts(model, sample, etc.) into s3, and follow [this](#) to set the credentials.

17.1.4 Check Lifelong Learning Job

query the service status

```
kubectl get lifelonglearningjob atcii-classifier-demo
```

In the lifelonglearningjob resource atcii-classifier-demo, the following trigger is configured:

```

trigger:
  checkPeriodSeconds: 60

```

(continues on next page)

(continued from previous page)

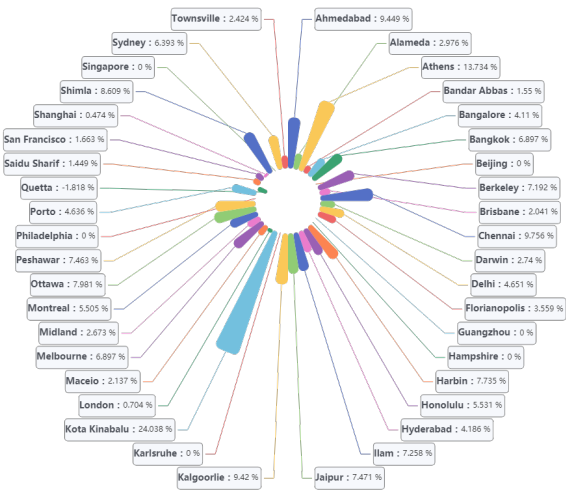
```
timer:
  start: 02:00
  end: 20:00
condition:
  operator: ">"
  threshold: 500
metric: num_of_samples
```

17.1.5 Unseen Tasks samples Labeling

In a real word, we need to label the hard examples in our unseen tasks which storage in UT_SAVED_URL with annotation tools and then put the examples to Dataset’s url.

17.1.6 Effect Display

In this example, **false** and **failed** detections occur at stage of inference before lifelong learning. After lifelong learning, the precision of the dataset have been improved by 5.12%.



PYTHON API USE GUIDE

18.1 Sedna Python SDK

The Sedna Python Software Development Kit (SDK) aims to provide developers with a convenient yet flexible tool to write the Sedna applications.

This document introduces how to obtain and call Sedna Python SDK.

18.1.1 Introduction

Expose the Edge AI features to applications, i.e. training or inference programs.

18.1.2 Requirements and Installation

The build process is tested with Python 3.6, Ubuntu 18.04.5 LTS

```
# Clone the repo
git clone --recursive https://github.com/kubeedge/sedna.git
cd sedna/lib

# Build the pip package
python setup.py bdist_wheel

# Install the pip package
pip install dist/sedna*.whl
```

Install via Setuptools

```
# Install dependence
pip install -r requirements.txt

# Install sedna
python setup.py install --user
```

18.1.3 Use Python SDK

1. (optional) Check Sedna version

```
$ python -c "import sedna; print(sedna.__version__)"
```

2. Import the required modules as follows:

```
from sedna.core.joint_inference import JointInference, BigModelService
from sedna.core.federated_learning import FederatedLearning
from sedna.core.incremental_learning import IncrementalLearning
from sedna.core.lifelong_learning import LifelongLearning
```

3. Define an Estimator:

```
import os

# Keras
import keras
from keras.layers import Dense, MaxPooling2D, Conv2D, Flatten, Dropout
from keras.models import Sequential

os.environ['BACKEND_TYPE'] = 'KERAS'

def KerasEstimator():
    model = Sequential()
    model.add(Conv2D(64, kernel_size=(3, 3),
                    activation="relu", strides=(2, 2),
                    input_shape=(128, 128, 3)))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(32, kernel_size=(3, 3), activation="relu"))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dropout(0.25))
    model.add(Dense(64, activation="relu"))
    model.add(Dense(32, activation="relu"))
    model.add(Dropout(0.5))
    model.add(Dense(2, activation="softmax"))

    model.compile(loss="categorical_crossentropy",
                  optimizer="adam",
                  metrics=["accuracy"])
    loss = keras.losses.CategoricalCrossentropy(from_logits=True)
    metrics = [keras.metrics.categorical_accuracy]
    optimizer = keras.optimizers.Adam(learning_rate=0.1)
    model.compile(loss=loss, optimizer=optimizer, metrics=metrics)
    return model
```

```
# XGBOOST

import os
import xgboost
```

(continues on next page)

(continued from previous page)

```

os.environ['BACKEND_TYPE'] = 'SKLEARN'

XGBEstimator = xgboost.XGBClassifier(
    learning_rate=0.1,
    n_estimators=600,
    max_depth=2,
    min_child_weight=1,
    gamma=0,
    subsample=0.8,
    colsample_bytree=0.8,
    objective="multi:softmax",
    num_class=3,
    nthread=4,
    seed=27
)

```

```

# Customize

class Estimator:

    def __init__(self, **kwargs):
        ...

    def load(self, model_url=""):
        ...

    def save(self, model_path=None):
        ...

    def predict(self, data, **kwargs):
        ...

    def evaluate(self, valid_data, **kwargs):
        ...

    def train(self, train_data, valid_data=None, **kwargs):
        ...

```

Notes: Estimator is a high-level API that greatly simplifies machine learning programming. Estimators encapsulate training, evaluation, prediction, and exporting for your model.

4. Initialize a Incremental Learning Job:

```

# get hard exmaple mining algorithm from config
hard_example_mining = IncrementalLearning.get_hem_algorithm_from_config(
    threshold_img=0.9
)

# create Incremental Learning infernece instance
il_job = IncrementalLearning(
    estimator=Estimator,
    hard_example_mining=hard_example_mining
)

```

(continues on next page)

(continued from previous page)

```
)
where:
```

- IncrementalLearning is the Cloud-edge job you want to access.
- Estimator is the base model for your ML job.
- hard_example_mining is the parameters of incremental learning job.

Inference

Note: The job parameters of each feature are different.

5. Running Job - training / inference / evaluation.

```
results, final_res, is_hard_example = il_job.inference(
    img_rgb,
    post_process=deal_infer_rsl,
    input_shape=input_shape
)
where:
```

- img_rgb is the sample used to inference
- deal_infer_rsl is a function used to process result after model predict
- input_shape is the parameters of Estimator in inference
- results is the result predicted by model
- final_res is the result after process by deal_infer_rsl
- is_hard_example tells if the sample is hard sample or not

18.1.4 Customize algorithm

Sedna provides a class called `class_factory.py` in `common` package, in which only a few lines of changes are required to become a module of sedna.

Two classes are defined in `class_factory.py`, namely `ClassType` and `ClassFactory`.

`ClassFactory` can register the modules you want to reuse through decorators. For example, in the following code example, you have customized an **hard_example_mining algorithm**, you only need to add a line of `ClassFactory.register(ClassType.HEM)` to complete the registration.

```
@ClassFactory.register(ClassType.HEM, alias="Threshold")
class ThresholdFilter(BaseFilter, abc.ABC):
    def __init__(self, threshold=0.5, **kwargs):
        self.threshold = float(threshold)

    def __call__(self, infer_result=None):
        # if invalid input, return False
        if not (infer_result
                and all(map(lambda x: len(x) > 4, infer_result))):
            return False
```

(continues on next page)

(continued from previous page)

```
image_score = 0

for bbox in infer_result:
    image_score += bbox[4]

average_score = image_score / (len(infer_result) or 1)
return average_score < self.threshold
```

After registration, you only need to change the name of the hem and parameters in the yaml file, and then the corresponding class will be automatically called according to the name.

```
deploySpec:
  hardExampleMining:
    name: "Threshold"
    parameters:
      - key: "threshold"
        value: "0.9"
```


19.1 Subpackages

19.1.1 lib.sedna.algorithms

Subpackages

lib.sedna.algorithms.aggregation

Submodules

lib.sedna.algorithms.aggregation.aggregation

Aggregation algorithms

Module Contents

Classes

<i>AggClient</i>	Client that interacts with cloud aggregator
<i>FedAvg</i>	Federated averaging algorithm

class lib.sedna.algorithms.aggregation.aggregation.**AggClient**

Client that interacts with cloud aggregator

Parameters

- **num_samples** (*int*) – number of samples for the current weights
- **weights** (*List*) – weights of the layer as a list of number-like array, such as `[[0, 0, 0, 0], [0, 0, 0, 0] ...]`

num_samples: `int`

weights: `List`

class lib.sedna.algorithms.aggregation.aggregation.**FedAvg**

Bases: `BaseAggregation, abc.ABC`

Federated averaging algorithm

aggregate(*clients*: List[[AggClient](#)])

Calculate the average weight according to the number of samples

Parameters

clients (*List*) – All clients in federated learning job

Returns

update_weights – final weights use to update model layer

Return type

Array-like

Package Contents

Classes

FedAvg	Federated averaging algorithm
MistNet	Abstract class of aggregator
AggClient	Client that interacts with cloud aggregator
FedAvgV2	Abstract class of aggregator

class lib.sedna.algorithms.aggregation.**FedAvg**

Bases: BaseAggregation, abc.ABC

Federated averaging algorithm

aggregate(*clients*: List[[AggClient](#)])

Calculate the average weight according to the number of samples

Parameters

clients (*List*) – All clients in federated learning job

Returns

update_weights – final weights use to update model layer

Return type

Array-like

class lib.sedna.algorithms.aggregation.**MistNet**(*cut_layer*, *epsilon*=100)

Bases: BaseAggregation, abc.ABC

Abstract class of aggregator

aggregate(*clients*: List[[AggClient](#)])

Some algorithms can be aggregated in sequence, but some can be calculated only after all aggregated data is uploaded. therefore, this abstractmethod should consider that all weights are uploaded.

Parameters

clients (*List*) – All clients in federated learning job

Returns

final weights use to update model layer

Return type

Array-like

class lib.sedna.algorithms.aggregation.**AggClient**

Client that interacts with cloud aggregator

Parameters

- **num_samples** (*int*) – number of samples for the current weights
- **weights** (*List*) – weights of the layer as a list of number-like array, such as [[0, 0, 0, 0], [0, 0, 0, 0] ...]

num_samples: int

weights: List

class lib.sedna.algorithms.aggregation.**FedAvgV2**

Bases: BaseAggregation, abc.ABC

Abstract class of aggregator

aggregate(*clients: List[AggClient]*)

Some algorithms can be aggregated in sequence, but some can be calculated only after all aggregated data is uploaded. therefore, this abstractmethod should consider that all weights are uploaded.

Parameters

clients (*List*) – All clients in federated learning job

Returns

final weights use to update model layer

Return type

Array-like

lib.sedna.algorithms.client_choose

Submodules

lib.sedna.algorithms.client_choose.client_choose

Module Contents

Classes

<i>AbstractClientChoose</i>	Abstract class of ClientChoose, which provides base client choose
<i>SimpleClientChoose</i>	A Simple Implementation of Client Choose.

class lib.sedna.algorithms.client_choose.client_choose.**AbstractClientChoose**

Abstract class of ClientChoose, which provides base client choose algorithm interfaces in federated learning.

class lib.sedna.algorithms.client_choose.client_choose.**SimpleClientChoose**(*per_round=1*)

Bases: *AbstractClientChoose*

A Simple Implementation of Client Choose.

Package Contents

Classes

<i>SimpleClientChoose</i>	A Simple Implementation of Client Choose.
---------------------------	---

class lib.sedna.algorithms.client_choose.**SimpleClientChoose**(*per_round=1*)

Bases: AbstractClientChoose

A Simple Implementation of Client Choose.

lib.sedna.algorithms.hard_example_mining

Submodules

lib.sedna.algorithms.hard_example_mining.hard_example_mining

Hard Example Mining Algorithms

Module Contents

Classes

<i>ThresholdFilter</i>	Object detection Hard samples discovery methods named <i>Threshold</i>
<i>CrossEntropyFilter</i>	Object detection Hard samples discovery methods named <i>CrossEntropy</i>
<i>IBTFilter</i>	Object detection Hard samples discovery methods named <i>IBT</i>

class lib.sedna.algorithms.hard_example_mining.hard_example_mining.**ThresholdFilter**(*threshold: float = 0.5, **kwargs*)

Bases: BaseFilter, abc.ABC

Object detection Hard samples discovery methods named *Threshold*

Parameters

threshold (*float*) – hard coefficient threshold score to filter img, default to 0.5.

__call__ (*infer_result=None*) → bool

predict function, judge the sample is hard or not.

Parameters

infer_result (*array_like*) – prediction result

Returns

is_hard_sample – *True* means hard sample, *False* means not.

Return type

bool

```
class lib.sedna.algorithms.hard_example_mining.hard_example_mining.CrossEntropyFilter(threshold_cross_entropy=0.5,
                                                                                       **kwargs)
```

Bases: BaseFilter, abc.ABC

Object detection Hard samples discovery methods named *CrossEntropy***Parameters**

threshold_cross_entropy (*float*) – hard coefficient threshold score to filter img, default to 0.5.

__call__ (*infer_result=None*) → bool
judge the img is hard sample or not.

Parameters

infer_result (*array_like*) – prediction classes list, such as [class1-score, class2-score, class2-score,...], where class-score is the score corresponding to the class, class-score value is in [0,1], who will be ignored if its value not in [0,1].

Returns

is hard sample – *True* means hard sample, *False* means not.

Return type

bool

```
class lib.sedna.algorithms.hard_example_mining.hard_example_mining.IBTFilter(threshold_img=0.5,
                                                                              thresh-
                                                                              old_box=0.5,
                                                                              **kwargs)
```

Bases: BaseFilter, abc.ABC

Object detection Hard samples discovery methods named *IBT***Parameters**

- **threshold_img** (*float*) – hard coefficient threshold score to filter img, default to 0.5.
- **threshold_box** (*float*) – threshold_box to calculate hard coefficient, formula is hard coefficient = number(prediction_boxes less than threshold_box) / number(prediction_boxes)

__call__ (*infer_result=None*) → bool
Judge the img is hard sample or not.

Parameters

infer_result (*array_like*) – prediction boxes list, such as [bbox1, bbox2, bbox3,...], where bbox = [xmin, ymin, xmax, ymax, score, label] score should be in [0,1], who will be ignored if its value not in [0,1].

Returns

is hard sample – *True* means hard sample, *False* means not.

Return type

bool

`lib.sedna.algorithms.multi_task_learning`

Subpackages

`lib.sedna.algorithms.multi_task_learning.task_jobs`

Submodules

`lib.sedna.algorithms.multi_task_learning.task_jobs.artifact`

Module Contents

Classes

Task

TaskGroup

Model

class `lib.sedna.algorithms.multi_task_learning.task_jobs.artifact.Task`(*entry, samples,*
meta_attr=None)

class `lib.sedna.algorithms.multi_task_learning.task_jobs.artifact.TaskGroup`(*entry, tasks:*
List[Task])

class `lib.sedna.algorithms.multi_task_learning.task_jobs.artifact.Model`(*index: int, entry,*
model, result)

`lib.sedna.algorithms.multi_task_learning.task_jobs.inference_integrate`

Integrate the inference results of all related tasks

Module Contents

Classes

DefaultInferenceIntegrate

Default calculation algorithm for inference integration

class `lib.sedna.algorithms.multi_task_learning.task_jobs.inference_integrate.DefaultInferenceIntegrate`(

Default calculation algorithm for inference integration

Parameters

models (*All models used for sample inference*) –

`__call__` (*tasks*: List[`lib.sedna.algorithms.multi_task_learning.task_jobs.artifact.Task`])

Parameters

tasks (All tasks with sample result) –

Returns

result

Return type

minimum result

`lib.sedna.algorithms.multi_task_learning.task_jobs.task_definition`

Divide multiple tasks based on data

param samples Train data

param see `sedna.datasources.BaseDataSource` for more detail.

returns

- **tasks** (All tasks based on training data.)
- **task_extractor** (Model with a method to predicting target tasks)

Module Contents

Classes

<code>TaskDefinitionBySVC</code>	Dividing datasets with <i>AgglomerativeClustering</i> based on kernel distance,
<code>TaskDefinitionByDataAttr</code>	Dividing datasets based on the common attributes,

class `lib.sedna.algorithms.multi_task_learning.task_jobs.task_definition.TaskDefinitionBySVC` (**kwargs)

Dividing datasets with *AgglomerativeClustering* based on kernel distance, Using SVC to fit the clustering result.

Parameters

None (*n_class* *int* or) – The number of clusters to find, default=2.

`__call__` (*samples*: `sedna.datasources.BaseDataSource`) →

Tuple[List[`lib.sedna.algorithms.multi_task_learning.task_jobs.artifact.Task`], Any, `sedna.datasources.BaseDataSource`]

class `lib.sedna.algorithms.multi_task_learning.task_jobs.task_definition.TaskDefinitionByDataAttr` (**kwargs)

Dividing datasets based on the common attributes, generally used for structured data.

Parameters

List[Metadata] (*attribute*) – metadata is usually a class feature label with a finite values.

`__call__` (*samples*: `sedna.datasources.BaseDataSource`) →

Tuple[List[`lib.sedna.algorithms.multi_task_learning.task_jobs.artifact.Task`], Any, `sedna.datasources.BaseDataSource`]

`lib.sedna.algorithms.multi_task_learning.task_jobs.task_mining`

Mining tasks of inference sample base on task attribute extractor

param `samples` *infer sample*
param see *sedna.datasources.BaseDataSource* for more detail.
returns
 allocations

rtype
 tasks that assigned to each sample

Module Contents

Classes

<i>TaskMiningBySVC</i>	Corresponding to <i>TaskDefinitionBySVC</i>
<i>TaskMiningByDataAttr</i>	Corresponding to <i>TaskDefinitionByDataAttr</i>

class `lib.sedna.algorithms.multi_task_learning.task_jobs.task_mining.TaskMiningBySVC`(*task_extractor*,
 ***kwargs*)

Corresponding to *TaskDefinitionBySVC*

Parameters

task_extractor (*Model*) – SVC Model used to predicting target tasks

__call__ (*samples*: *sedna.datasources.BaseDataSource*)

class `lib.sedna.algorithms.multi_task_learning.task_jobs.task_mining.TaskMiningByDataAttr`(*task_extractor*,
 ***kwargs*)

Corresponding to *TaskDefinitionByDataAttr*

Parameters

- **task_extractor** (*Dict*) – used to match target tasks
- **attr_filed** (*List[Metadata]*) – metadata is usually a class feature label with a finite values.

__call__ (*samples*: *sedna.datasources.BaseDataSource*)

`lib.sedna.algorithms.multi_task_learning.task_jobs.task_relation_discover`

Discover relationships between all tasks

param `tasks` all tasks form *task_definition*
returns
 task_groups

rtype
 List of groups which including at least 1 task.

Module Contents

Classes

<i>DefaultTaskRelationDiscover</i>	Assume that each task is independent of each other
------------------------------------	--

class `lib.sedna.algorithms.multi_task_learning.task_jobs.task_relation_discover.DefaultTaskRelationDiscover`

Assume that each task is independent of each other

__call__ (*tasks*: *List*[`lib.sedna.algorithms.multi_task_learning.task_jobs.artifact.Task`]) →
List[`lib.sedna.algorithms.multi_task_learning.task_jobs.artifact.TaskGroup`]

lib.sedna.algorithms.multi_task_learning.task_jobs.task_remodeling

Remodeling tasks based on their relationships

param mappings all assigned tasks get from the *task_mining*

param samples

type samples
input samples

returns

models

rtype

List of groups which including at least 1 task.

Module Contents

Classes

<i>DefaultTaskRemodeling</i>	Assume that each task is independent of each other
------------------------------	--

class `lib.sedna.algorithms.multi_task_learning.task_jobs.task_remodeling.DefaultTaskRemodeling`(*models*:
list,
***kwargs*)

Assume that each task is independent of each other

__call__ (*samples*: `sedna.datasources.BaseDataSource`, *mappings*: *List*)
Grouping based on assigned tasks

Submodules

`lib.sedna.algorithms.multi_task_learning.multi_task_learning`

Multiple task transfer learning algorithms

Module Contents

Classes

<i>MulTaskLearning</i>	An auto machine learning framework for edge-cloud multitask learning
------------------------	--

```
class lib.sedna.algorithms.multi_task_learning.multi_task_learning.MulTaskLearning(estimator=None,
                                                                                      task_definition=None,
                                                                                      task_relationship_discovery=None,
                                                                                      task_mining=None,
                                                                                      task_remodeling=None,
                                                                                      inference_integrate=None)
```

An auto machine learning framework for edge-cloud multitask learning

See also:

Train

Data + Estimator -> Task Definition -> Task Relationship Discovery -> Feature Engineering -> Training

Inference

Data -> Task Allocation -> Task Mining -> Feature Engineering -> Task Remodeling -> Inference

Parameters

- **estimator** (*Instance*) – An instance with the high-level API that greatly simplifies machine learning programming. Estimators encapsulate training, evaluation, prediction, and exporting for your model.
- **task_definition** (*Dict*) – Divide multiple tasks based on data, see *task_jobs.task_definition* for more detail.
- **task_relationship_discovery** (*Dict*) – Discover relationships between all tasks, see *task_jobs.task_relationship_discovery* for more detail.
- **task_mining** (*Dict*) – Mining tasks of inference sample, see *task_jobs.task_mining* for more detail.
- **task_remodeling** (*Dict*) – Remodeling tasks based on their relationships, see *task_jobs.task_remodeling* for more detail.
- **inference_integrate** (*Dict*) – Integrate the inference results of all related tasks, see *task_jobs.inference_integrate* for more detail.

Examples

```
>>> from xgboost import XGBClassifier
>>> from sedna.algorithms.multi_task_learning import MulTaskLearning
>>> estimator = XGBClassifier(objective="binary:logistic")
>>> task_definition = {
    "method": "TaskDefinitionByDataAttr",
    "param": {"attribute": ["season", "city"]}
}
>>> task_relationship_discovery = {
    "method": "DefaultTaskRelationDiscover", "param": {}
}
>>> task_mining = {
    "method": "TaskMiningByDataAttr",
    "param": {"attribute": ["season", "city"]}
}
>>> task_remodeling = None
>>> inference_integrate = {
    "method": "DefaultInferenceIntegrate", "param": {}
}
>>> mul_task_instance = MulTaskLearning(
    estimator=estimator,
    task_definition=task_definition,
    task_relationship_discovery=task_relationship_discovery,
    task_mining=task_mining,
    task_remodeling=task_remodeling,
    inference_integrate=inference_integrate
)
```

Notes

All method defined under *task_jobs* and registered in *ClassFactory*.

train(*train_data*: [sedna.datasources.BaseDataSource](#), *valid_data*: [sedna.datasources.BaseDataSource](#) = *None*, *post_process*=*None*, ***kwargs*)

fit for update the knowledge based on training data.

Parameters

- **train_data** ([BaseDataSource](#)) – Train data, see *sedna.datasources.BaseDataSource* for more detail.
- **valid_data** ([BaseDataSource](#)) – Valid data, [BaseDataSource](#) or *None*.
- **post_process** (*function*) – function or a registered method, callback after *estimator* train.
- **kwargs** (*Dict*) – parameters for *estimator* training, Like: *early_stopping_rounds* in *Xgboost.XGBClassifier*

Returns

- **feedback** (*Dict*) – contain all training result in each tasks.
- **task_index_url** (*str*) – task extractor model path, used for task mining.

load(*task_index_url=None*)

load task_detail (tasks/models etc ...) from task index file. It'll automatically loaded during *inference* and *evaluation* phases.

Parameters

task_index_url (*str*) – task index file path, default self.task_index_url.

predict(*data: sedna.datasources.BaseDataSource, post_process=None, **kwargs*)

predict the result for input data based on training knowledge.

Parameters

- **data** (*BaseDataSource*) – inference sample, see *sedna.datasources.BaseDataSource* for more detail.
- **post_process** (*function*) – function or a registered method, effected after *estimator* prediction, like: label transform.
- **kwargs** (*Dict*) – parameters for *estimator* predict, Like: *ntree_limit* in Xgboost.XGBClassifier

Returns

- **result** (*array_like*) – results array, contain all inference results in each sample.
- **tasks** (*List*) – tasks assigned to each sample.

evaluate(*data: sedna.datasources.BaseDataSource, metrics=None, metrics_param=None, **kwargs*)

evaluated the performance of each task from training, filter tasks based on the defined rules.

Parameters

- **data** (*BaseDataSource*) – valid data, see *sedna.datasources.BaseDataSource* for more detail.
- **metrics** (*function / str*) – Metrics to assess performance on the task by given prediction.
- **metrics_param** (*Dict*) – parameter for metrics function.
- **kwargs** (*Dict*) – parameters for *estimator* evaluate, Like: *ntree_limit* in Xgboost.XGBClassifier

Returns

- **task_eval_res** (*Dict*) – all metric results.
- **tasks_detail** (*List[Object]*) – all metric results in each task.

Package Contents

Classes

MultiTaskLearning

An auto machine learning framework for edge-cloud multitask learning

```
class lib.sedna.algorithms.multi_task_learning.MulTaskLearning(estimator=None,
                                                                task_definition=None,
                                                                task_relationship_discovery=None,
                                                                task_mining=None,
                                                                task_remodeling=None,
                                                                inference_integrate=None)
```

An auto machine learning framework for edge-cloud multitask learning

See also:

Train

Data + Estimator -> Task Definition -> Task Relationship Discovery -> Feature Engineering -> Training

Inference

Data -> Task Allocation -> Task Mining -> Feature Engineering -> Task Remodeling -> Inference

Parameters

- **estimator** (*Instance*) – An instance with the high-level API that greatly simplifies machine learning programming. Estimators encapsulate training, evaluation, prediction, and exporting for your model.
- **task_definition** (*Dict*) – Divide multiple tasks based on data, see *task_jobs.task_definition* for more detail.
- **task_relationship_discovery** (*Dict*) – Discover relationships between all tasks, see *task_jobs.task_relationship_discovery* for more detail.
- **task_mining** (*Dict*) – Mining tasks of inference sample, see *task_jobs.task_mining* for more detail.
- **task_remodeling** (*Dict*) – Remodeling tasks based on their relationships, see *task_jobs.task_remodeling* for more detail.
- **inference_integrate** (*Dict*) – Integrate the inference results of all related tasks, see *task_jobs.inference_integrate* for more detail.

Examples

```
>>> from xgboost import XGBClassifier
>>> from sedna.algorithms.multi_task_learning import MulTaskLearning
>>> estimator = XGBClassifier(objective="binary:logistic")
>>> task_definition = {
    "method": "TaskDefinitionByDataAttr",
    "param": {"attribute": ["season", "city"]}
}
>>> task_relationship_discovery = {
    "method": "DefaultTaskRelationDiscover", "param": {}
}
>>> task_mining = {
    "method": "TaskMiningByDataAttr",
    "param": {"attribute": ["season", "city"]}
}
>>> task_remodeling = None
>>> inference_integrate = {
    "method": "DefaultInferenceIntegrate", "param": {}
}
```

(continues on next page)

(continued from previous page)

```

    }
    >>> mul_task_instance = MulTaskLearning(
        estimator=estimator,
        task_definition=task_definition,
        task_relationship_discovery=task_relationship_discovery,
        task_mining=task_mining,
        task_remodeling=task_remodeling,
        inference_integrate=inference_integrate
    )

```

Notes

All method defined under *task_jobs* and registered in *ClassFactory*.

train(*train_data*: [sedna.datasources.BaseDataSource](#), *valid_data*: [sedna.datasources.BaseDataSource](#) = *None*, *post_process*=*None*, ***kwargs*)

fit for update the knowledge based on training data.

Parameters

- **train_data** ([BaseDataSource](#)) – Train data, see *sedna.datasources.BaseDataSource* for more detail.
- **valid_data** ([BaseDataSource](#)) – Valid data, [BaseDataSource](#) or *None*.
- **post_process** (*function*) – function or a registered method, callback after *estimator* train.
- **kwargs** (*Dict*) – parameters for *estimator* training, Like: *early_stopping_rounds* in *Xgboost.XGBClassifier*

Returns

- **feedback** (*Dict*) – contain all training result in each tasks.
- **task_index_url** (*str*) – task extractor model path, used for task mining.

load(*task_index_url*=*None*)

load *task_detail* (tasks/models etc ...) from task index file. It'll automatically loaded during *inference* and *evaluation* phases.

Parameters

task_index_url (*str*) – task index file path, default *self.task_index_url*.

predict(*data*: [sedna.datasources.BaseDataSource](#), *post_process*=*None*, ***kwargs*)

predict the result for input data based on training knowledge.

Parameters

- **data** ([BaseDataSource](#)) – inference sample, see *sedna.datasources.BaseDataSource* for more detail.
- **post_process** (*function*) – function or a registered method, effected after *estimator* prediction, like: label transform.
- **kwargs** (*Dict*) – parameters for *estimator* predict, Like: *ntree_limit* in *Xgboost.XGBClassifier*

Returns

- **result** (*array_like*) – results array, contain all inference results in each sample.
- **tasks** (*List*) – tasks assigned to each sample.

evaluate(*data*: [sedna.datasources.BaseDataSource](#), *metrics*=None, *metrics_param*=None, ***kwargs*)

evaluated the performance of each task from training, filter tasks based on the defined rules.

Parameters

- **data** ([BaseDataSource](#)) – valid data, see [sedna.datasources.BaseDataSource](#) for more detail.
- **metrics** (*function* / *str*) – Metrics to assess performance on the task by given prediction.
- **metrics_param** (*Dict*) – parameter for metrics function.
- **kwargs** (*Dict*) – parameters for *estimator* evaluate, Like: *ntree_limit* in `Xgboost.XGBClassifier`

Returns

- **task_eval_res** (*Dict*) – all metric results.
- **tasks_detail** (*List[Object]*) – all metric results in each task.

[lib.sedna.algorithms.optical_flow](#)

Optical Flow Algorithms

[lib.sedna.algorithms.reid](#)

Submodules

[lib.sedna.algorithms.reid.close_contact_estimation](#)

Module Contents

Classes

ContactTracker	The ContactTracker object is invoked in frames where the
--------------------------------	--

class `lib.sedna.algorithms.reid.close_contact_estimation.ContactTracker`(*draw_top_view*=False)

Bases: `object`

The ContactTracker object is invoked in frames where the target person was identified.

prep_homography(*img_shape*, *bbox_target*, *h_ratio*: *float* = 0.5, *v_ratio*: *float* = 0.5)

compute_homography(*img_shape*: *List[int]*) → None

Calculate homography @param *img_shape*: List [h,w]

in_risk_zone(*img*: *numpy.ndarray*, *bbox_candidate*: *List[int]* = None) → bool

create_ellipse(*bbox_list*: List[List[int]] = None) → Tuple[List, List]

Create ellipses for each of the generated bounding boxes. @param *bbox_list*:

check_bbox_overlap(*ellipse1*: Tuple, *ellipse2*: Tuple) → bool

Check if ellipse bounding rectangles overlap or not. :param *ellipse1*: ellipse one :type *ellipse1*: tuple :param *ellipse2*: ellipse two :type *ellipse2*: tuple

Return type

boolean

to_rectangle(*ellipse*: Tuple) → Tuple

Convert ellipse to rectangle (top, left, bottom, right) :param *ellipse*: bounding rectangle descriptor :type *ellipse*: tuple

get_homography_matrix()

`lib.sedna.algorithms.reid.multi_img_matching`

Module Contents

Functions

<code>cosine_similarity_score</code> ([<i>query</i> , <i>candidates</i>])	Computes the cosine similarity score between the
<code>tensor_reshape</code> (→ torch.Tensor)	

<code>match_query_to_targets</code> (→ Tuple[int, float])	Query features refer to the features of
---	---

`lib.sedna.algorithms.reid.multi_img_matching.cosine_similarity_score`(*query*: numpy.ndarray = None, *candidates*: numpy.ndarray = None)

Computes the cosine similarity score between the
query feature and the candidate features.

@param query: Feature map of dimension
[1, n_feat_dim] representing the query.

@param candidates: Feature map of dimension
[n_candidates, n_feat_dim] representing the candidate for match.

`lib.sedna.algorithms.reid.multi_img_matching.tensor_reshape`(*data*: Any) → torch.Tensor

`lib.sedna.algorithms.reid.multi_img_matching.match_query_to_targets`(*query_feats*: List, *candidate_feats*: List, *avg_mode*: bool = False) → Tuple[int, float]

Query features refer to the features of the person we are looking for in the video. Candidate features refers to features of the persons found by the detector in the current scene. :param *query_feats*: [M x d] M being the number of target images in the query :param *candidate_feats*: [N x d] N is the number of persons detected in the scene :param *avg_mode*: If set, use an average representation of the query.

Query feats becomes [1 x d]

Returns

Id of the candidate which best matches the query

`lib.sedna.algorithms.seen_task_learning`

Subpackages

`lib.sedna.algorithms.seen_task_learning.inference_integration`

Submodules

`lib.sedna.algorithms.seen_task_learning.inference_integration.base_inference_integrate`

Integrate the inference results of all related tasks

Module Contents

Classes

<i>BaseInferenceIntegrate</i>	Base class for default calculation algorithm for inference integration
-------------------------------	--

class `lib.sedna.algorithms.seen_task_learning.inference_integration.base_inference_integrate.BaseInferenceIntegrate`

Base class for default calculation algorithm for inference integration

Parameters

models (*All models used for sample inference*) –

abstract `__call__` (*tasks: List[lib.sedna.algorithms.seen_task_learning.artifact.Task]*)

Parameters

tasks (*All tasks with sample result*) –

Returns

result

Return type

inference results for all the inference samples

`lib.sedna.algorithms.seen_task_learning.inference_integration.inference_integrate`

Integrate the inference results of all related tasks

Module Contents

Classes

*DefaultInferenceIntegrate*Default calculation algorithm for inference integration

```
class lib.sedna.algorithms.seen_task_learning.inference_integration.inference_integrate.DefaultInferenceIntegrate
```

Bases: *lib.sedna.algorithms.seen_task_learning.inference_integration.base_inference_integrate.BaseInferenceIntegrate*

Default calculation algorithm for inference integration

Parameters

models (*All models used for sample inference*) –

__call__ (*tasks: List[lib.sedna.algorithms.seen_task_learning.artifact.Task]*)

Parameters

tasks (*All tasks with sample result*) –

Returns

result

Return type

minimum result

```
lib.sedna.algorithms.seen_task_learning.task_allocation
```

Submodules

```
lib.sedna.algorithms.seen_task_learning.task_allocation.base_task_allocation
```

Mining tasks of inference sample based on task attribute extractor

param samples *infer sample*

param see *sedna.datasources.BaseDataSource* for more detail.

returns

allocations

rtype

tasks that assigned to each sample

Module Contents

Classes

<i>BaseTaskAllocation</i>	Base class of task allocation algorithm
---------------------------	---

```
class lib.sedna.algorithms.seen_task_learning.task_allocation.base_task_allocation.BaseTaskAllocation(task_extractor, task_definition, **kwargs):
```

Base class of task allocation algorithm

Parameters

task_extractor (*Model* or *dict*) – task extractor is used to predict target tasks

abstract __call__ (*samples*: *sedna.datasources.BaseDataSource*)

```
lib.sedna.algorithms.seen_task_learning.task_allocation.task_allocation
```

Mining tasks of inference sample based on task attribute extractor

param samples *infer sample*

param see *sedna.datasources.BaseDataSource* for more detail.

returns

allocations

rtype

tasks that assigned to each sample

Module Contents

Classes

<i>TaskAllocationBySVC</i>	Corresponding to <i>TaskDefinitionBySVC</i>
<i>TaskAllocationByDataAttr</i>	Corresponding to <i>TaskDefinitionByDataAttr</i>
<i>TaskAllocationDefault</i>	Task allocation specifically for unstructured data

```
class lib.sedna.algorithms.seen_task_learning.task_allocation.task_allocation.TaskAllocationBySVC(task_extractor, task_definition, **kwargs):
```

Corresponding to *TaskDefinitionBySVC*

Parameters

task_extractor (*Model*) – SVC Model used to predicting target tasks

__call__ (*samples*: *sedna.datasources.BaseDataSource*)

```
class lib.sedna.algorithms.seen_task_learning.task_allocation.task_allocation.TaskAllocationByDataAttr(task_extractor, task_definition, **kwargs):
```

Corresponding to *TaskDefinitionByDataAttr*

Parameters

- **task_extractor** (*Dict*) – used to match target tasks

- **attr_filed** (*List[Metadata]*) – metadata is usually a class feature label with a finite values.

`__call__(samples: sedna.datasources.BaseDataSource)`

`class lib.sedna.algorithms.seen_task_learning.task_allocation.task_allocation.TaskAllocationDefault(task`
`**kw`

Task allocation specifically for unstructured data

Parameters

task_extractor (*Dict*) – used to match target tasks

`__call__(samples: sedna.datasources.BaseDataSource)`

`lib.sedna.algorithms.seen_task_learning.task_allocation.task_allocation_by_origin`

Module Contents

Classes

TaskAllocationByOrigin

Corresponding to *TaskDefinitionByOrigin*

`class lib.sedna.algorithms.seen_task_learning.task_allocation.task_allocation_by_origin.TaskAllocationByOrigin`

Bases: `lib.sedna.algorithms.seen_task_learning.task_allocation.base_task_allocation.BaseTaskAllocation`

Corresponding to *TaskDefinitionByOrigin*

Parameters

- **task_extractor** (*Dict*) – used to predict target tasks for each inference samples
- **origins** (*List[Metadata]*) – metadata is usually a class feature label with a finite values.

`__call__(samples: sedna.datasources.BaseDataSource)`

`lib.sedna.algorithms.seen_task_learning.task_allocation.task_allocation_stream`

Module Contents

Classes

TaskAllocationStream

Corresponding to *TaskDefinitionByOrigin*

`class lib.sedna.algorithms.seen_task_learning.task_allocation.task_allocation_stream.TaskAllocationStream`

Bases: `lib.sedna.algorithms.seen_task_learning.task_allocation.base_task_allocation.BaseTaskAllocation`

Corresponding to *TaskDefinitionByOrigin*

Parameters

- **task_extractor** (*Dict*) – used to predict target tasks for each inference sample
 - **origins** (*List[Metadata]*) – metadata is usually a class feature label with a finite values.
- __call__** (*samples: sedna.datasources.BaseDataSource*)

`lib.sedna.algorithms.seen_task_learning.task_definition`

Submodules

`lib.sedna.algorithms.seen_task_learning.task_definition.base_task_definition`

Divide multiple tasks based on data

- param samples** Train data
param see *sedna.datasources.BaseDataSource* for more detail.
returns
- **tasks** (*All tasks based on training data.*)
 - **task_extractor** (*Model or dict with a method to predict target tasks*)

Module Contents

Classes

BaseTaskDefinition

Dividing datasets with all sorts of methods

class `lib.sedna.algorithms.seen_task_learning.task_definition.base_task_definition.BaseTaskDefinition`(*)
 Dividing datasets with all sorts of methods

abstract **__call__** (*samples: sedna.datasources.BaseDataSource*) →
Tuple[List[lib.sedna.algorithms.seen_task_learning.artifact.Task], Any, sedna.datasources.BaseDataSource]

`lib.sedna.algorithms.seen_task_learning.task_definition.task_definition`

Divide multiple tasks based on data

- param samples** Train data
param see *sedna.datasources.BaseDataSource* for more detail.
returns
- **tasks** (*All tasks based on training data.*)
 - **task_extractor** (*Model with a method to predicting target tasks*)

Module Contents

Classes

<i>TaskDefinitionBySVC</i>	Dividing datasets with <i>AgglomerativeClustering</i> based on kernel distance,
<i>TaskDefinitionByDataAttr</i>	Dividing datasets based on the common attributes,

class `lib.sedna.algorithms.seen_task_learning.task_definition.task_definition.TaskDefinitionBySVC(**kwargs)`
Dividing datasets with *AgglomerativeClustering* based on kernel distance, Using SVC to fit the clustering result.

Parameters

None (*n_class* *int* *or*) – The number of clusters to find, default=2.

__call__ (*samples*: `sedna.datasources.BaseDataSource`) →
Tuple[List[`lib.sedna.algorithms.seen_task_learning.artifact.Task`], Any,
`sedna.datasources.BaseDataSource`]

class `lib.sedna.algorithms.seen_task_learning.task_definition.task_definition.TaskDefinitionByDataAttr()`
Dividing datasets based on the common attributes, generally used for structured data.

Parameters

List[Metadata] (*attribute*) – metadata is usually a class feature label with a finite values.

__call__ (*samples*: `sedna.datasources.BaseDataSource`, ****kwargs**) →
Tuple[List[`lib.sedna.algorithms.seen_task_learning.artifact.Task`], Any,
`sedna.datasources.BaseDataSource`]

`lib.sedna.algorithms.seen_task_learning.task_definition.task_definition_by_origin`

Module Contents

Classes

<i>TaskDefinitionByOrigin</i>	Dividing datasets based on the their origins.
-------------------------------	---

class `lib.sedna.algorithms.seen_task_learning.task_definition.task_definition_by_origin.TaskDefinitionByOrigin`
Bases: `lib.sedna.algorithms.seen_task_learning.task_definition.base_task_definition.BaseTaskDefinition`

Dividing datasets based on the their origins.

Parameters

Tuple[Metadata] (*attr_filed*) – metadata is usually a class feature label with a finite values.

__call__ (*samples*: `sedna.datasources.BaseDataSource`, ****kwargs**) →
Tuple[List[`lib.sedna.algorithms.seen_task_learning.artifact.Task`], Any,
`sedna.datasources.BaseDataSource`]

`lib.sedna.algorithms.seen_task_learning.task_relation_discovery`

Submodules

`lib.sedna.algorithms.seen_task_learning.task_relation_discovery.base_task_relation_discovery`

Discover relationships between all tasks

param tasks all tasks form *task_definition*

returns

task_groups

rtype

List of groups which including at least 1 task.

Module Contents

Classes

<i>BaseTaskRelationDiscover</i>	Assume that each task is independent of each other
---	--

class `lib.sedna.algorithms.seen_task_learning.task_relation_discovery.base_task_relation_discovery.BaseTaskRelationDiscover`

Assume that each task is independent of each other

abstract `__call__`(tasks: List[lib.sedna.algorithms.seen_task_learning.artifact.Task]) →
List[lib.sedna.algorithms.seen_task_learning.artifact.TaskGroup]

`lib.sedna.algorithms.seen_task_learning.task_relation_discovery.task_relation_discovery`

Discover relationships between all tasks

param tasks all tasks form *task_definition*

returns

task_groups

rtype

List of groups which including at least 1 task.

Module Contents

Classes

<i>DefaultTaskRelationDiscover</i>	Assume that each task is independent of each other
--	--

class `lib.sedna.algorithms.seen_task_learning.task_relation_discovery.task_relation_discovery.DefaultTaskRelationDiscover`

Bases: [*lib.sedna.algorithms.seen_task_learning.task_relation_discovery.base_task_relation_discovery.BaseTaskRelationDiscover*](#)

Assume that each task is independent of each other

```
__call__(tasks: List[lib.sedna.algorithms.seen_task_learning.artifact.Task]) →  
List[lib.sedna.algorithms.seen_task_learning.artifact.TaskGroup]
```

`lib.sedna.algorithms.seen_task_learning.task_remodeling`

Submodules

`lib.sedna.algorithms.seen_task_learning.task_remodeling.base_task_remodeling`

Remodeling tasks based on their relationships

param mappings all assigned tasks get from the *task_allocation*

param samples

type samples

input samples

returns

models

rtype

List of groups which including at least 1 task.

Module Contents

Classes

BaseTaskRemodeling

Assume that each task is independent of each other

```
class lib.sedna.algorithms.seen_task_learning.task_remodeling.base_task_remodeling.BaseTaskRemodeling(m  
li.  
*:
```

Assume that each task is independent of each other

abstract __call__(*samples*: sedna.datasources.BaseDataSource, *mappings*: List)

Grouping based on assigned tasks

`lib.sedna.algorithms.seen_task_learning.task_remodeling.task_remodeling`

Remodeling tasks based on their relationships

param mappings all assigned tasks get from the *task_mining*

param samples

type samples

input samples

returns

models

rtype

List of groups which including at least 1 task.

Module Contents

Classes

<i>DefaultTaskRemodeling</i>	Assume that each task is independent of each other
------------------------------	--

```
class lib.sedna.algorithms.seen_task_learning.task_remodeling.task_remodeling.DefaultTaskRemodeling(mod
list,
**k)
```

Bases: *lib.sedna.algorithms.seen_task_learning.task_remodeling.base_task_remodeling.BaseTaskRemodeling*

Assume that each task is independent of each other

__call__ (*samples: sedna.datasources.BaseDataSource, mappings: List*)

Grouping based on assigned tasks

lib.sedna.algorithms.seen_task_learning.task_update_decision

Submodules

lib.sedna.algorithms.seen_task_learning.task_update_decision.base_task_update_decision

Divide multiple tasks based on data

param samples Train data

param see *sedna.datasources.BaseDataSource* for more detail.

returns

- **tasks** (*All tasks based on training data.*)
- **task_extractor** (*Model or dict with a method to predict target tasks*)

Module Contents

Classes

<i>BaseTaskUpdateDecision</i>	Decide processing strategies for different tasks
-------------------------------	--

```
class lib.sedna.algorithms.seen_task_learning.task_update_decision.base_task_update_decision.BaseTaskUp
```

Decide processing strategies for different tasks with labeled unseen samples. Turn unseen samples to be seen.

Parameters

task_index (*str or Dict*) –

abstract __call__ (*samples: sedna.datasources.BaseDataSource*) →

Tuple[List[*lib.sedna.algorithms.seen_task_learning.artifact.Task*], Dict]

```
lib.sedna.algorithms.seen_task_learning.task_update_decision.task_update_decision_finetune
```

Submodules

```
lib.sedna.algorithms.seen_task_learning.artifact
```

Module Contents

Classes

Task

TaskGroup

Model

```
class lib.sedna.algorithms.seen_task_learning.artifact.Task(entry, samples, meta_attr=None)
```

```
class lib.sedna.algorithms.seen_task_learning.artifact.TaskGroup(entry, tasks: List[Task])
```

```
class lib.sedna.algorithms.seen_task_learning.artifact.Model(index: int, entry, model, result)
```

```
lib.sedna.algorithms.seen_task_learning.seen_task_learning
```

Multiple task transfer learning algorithms

Module Contents

Classes

SeenTaskLearning

An auto machine learning framework for edge-cloud multitask learning

```
class lib.sedna.algorithms.seen_task_learning.seen_task_learning.SeenTaskLearning(estimator=None,
                                                                                   task_definition=None,
                                                                                   task_relationship_discovery=None,
                                                                                   seen_task_allocation=None,
                                                                                   task_remodeling=None,
                                                                                   inference_integrate=None)
```

An auto machine learning framework for edge-cloud multitask learning

See also:

Train

Data + Estimator -> Task Definition -> Task Relationship Discovery -> Feature Engineering -> Training

Inference

Data -> Task Allocation -> Feature Engineering -> Task Remodeling -> Inference

Parameters

- **estimator** (*Instance*) – An instance with the high-level API that greatly simplifies machine learning programming. Estimators encapsulate training, evaluation, prediction, and exporting for your model.
- **task_definition** (*Dict*) – Divide multiple tasks based on data, see *task_jobs.task_definition* for more detail.
- **task_relationship_discovery** (*Dict*) – Discover relationships between all tasks, see *task_jobs.task_relationship_discovery* for more detail.
- **seen_task_allocation** (*Dict*) – Mining tasks of inference sample, see *task_jobs.task_mining* for more detail.
- **task_remodeling** (*Dict*) – Remodeling tasks based on their relationships, see *task_jobs.task_remodeling* for more detail.
- **inference_integrate** (*Dict*) – Integrate the inference results of all related tasks, see *task_jobs.inference_integrate* for more detail.

Examples

```
>>> from xgboost import XGBClassifier
>>> from sedna.algorithms.multi_task_learning import MultiTaskLearning
>>> estimator = XGBClassifier(objective="binary:logistic")
>>> task_definition = {
    "method": "TaskDefinitionByDataAttr",
    "param": {"attribute": ["season", "city"]}
}
>>> task_relationship_discovery = {
    "method": "DefaultTaskRelationDiscover", "param": {}
}
>>> seen_task_allocation = {
    "method": "TaskAllocationByDataAttr",
    "param": {"attribute": ["season", "city"]}
}
>>> task_remodeling = None
>>> inference_integrate = {
    "method": "DefaultInferenceIntegrate", "param": {}
}
>>> mul_task_instance = MultiTaskLearning(
    estimator=estimator,
    task_definition=task_definition,
    task_relationship_discovery=task_relationship_discovery,
    seen_task_allocation=seen_task_allocation,
    task_remodeling=task_remodeling,
    inference_integrate=inference_integrate
)
```

Notes

All method defined under *task_jobs* and registered in *ClassFactory*.

train(*train_data*: [sedna.datasources.BaseDataSource](#), *valid_data*: [sedna.datasources.BaseDataSource](#) = *None*, *post_process*=*None*, ***kwargs*)

fit for update the knowledge based on training data.

Parameters

- **train_data** ([BaseDataSource](#)) – Train data, see *sedna.datasources.BaseDataSource* for more detail.
- **valid_data** ([BaseDataSource](#)) – Valid data, [BaseDataSource](#) or *None*.
- **post_process** (*function*) – function or a registered method, callback after *estimator* train.
- **kwargs** (*Dict*) – parameters for *estimator* training, Like: *early_stopping_rounds* in *Xgboost.XGBClassifier*

Returns

- **feedback** (*Dict*) – contain all training result in each tasks.
- **task_index_url** (*str*) – task extractor model path, used for task allocation.

update(*tasks*, *task_update_strategies*, ***kwargs*)

Parameters:

tasks: *List[Task]*

from the output of module *task_update_decision*

task_update_strategies: *object*

from the output of module *task_update_decision*

returns

task_index – updated seen task index of knowledge base

rtype

Dict

load(*task_index*)

load *task_detail* (tasks/models etc ...) from task index file. It'll automatically loaded during *inference* and *evaluation* phases.

Parameters

task_index (*str* or *Dict*) – task index file path, default *self.task_index_url*.

predict(*data*: [sedna.datasources.BaseDataSource](#), *post_process*=*None*, ***kwargs*)

predict the result for input data based on training knowledge.

Parameters

- **data** ([BaseDataSource](#)) – inference sample, see *sedna.datasources.BaseDataSource* for more detail.
- **post_process** (*function*) – function or a registered method, effected after *estimator* prediction, like: label transform.

- **kwargs** (*Dict*) – parameters for *estimator* predict, Like: *ntree_limit* in *Xgboost.XGBClassifier*

Returns

- **result** (*array_like*) – results array, contain all inference results in each sample.
- **tasks** (*List*) – tasks assigned to each sample.

evaluate(*data*: [sedna.datasources.BaseDataSource](#), *metrics*=None, *metrics_param*=None, ***kwargs*)
evaluated the performance of each task from training, filter tasks based on the defined rules.

Parameters

- **data** ([BaseDataSource](#)) – valid data, see *sedna.datasources.BaseDataSource* for more detail.
- **metrics** (*function* / *str*) – Metrics to assess performance on the task by given prediction.
- **metrics_param** (*Dict*) – parameter for metrics function.
- **kwargs** (*Dict*) – parameters for *estimator* evaluate, Like: *ntree_limit* in *Xgboost.XGBClassifier*

Returns

- **task_eval_res** (*Dict*) – all metric results.
- **tasks_detail** (*List[Object]*) – all metric results in each task.

Package Contents

Classes

SeenTaskLearning	An auto machine learning framework for edge-cloud multitask learning
----------------------------------	--

```
class lib.sedna.algorithms.seen_task_learning.SeenTaskLearning(estimator=None,
                                                                task_definition=None,
                                                                task_relationship_discovery=None,
                                                                seen_task_allocation=None,
                                                                task_remolding=None,
                                                                inference_integrate=None)
```

An auto machine learning framework for edge-cloud multitask learning

See also:

Train

Data + Estimator -> Task Definition -> Task Relationship Discovery -> Feature Engineering -> Training

Inference

Data -> Task Allocation -> Feature Engineering -> Task Remodeling -> Inference

Parameters

- **estimator** (*Instance*) – An instance with the high-level API that greatly simplifies machine learning programming. Estimators encapsulate training, evaluation, prediction, and exporting for your model.

- **task_definition** (*Dict*) – Divide multiple tasks based on data, see *task_jobs.task_definition* for more detail.
- **task_relationship_discovery** (*Dict*) – Discover relationships between all tasks, see *task_jobs.task_relationship_discovery* for more detail.
- **seen_task_allocation** (*Dict*) – Mining tasks of inference sample, see *task_jobs.task_mining* for more detail.
- **task_remodeling** (*Dict*) – Remodeling tasks based on their relationships, see *task_jobs.task_remodeling* for more detail.
- **inference_integrate** (*Dict*) – Integrate the inference results of all related tasks, see *task_jobs.inference_integrate* for more detail.

Examples

```
>>> from xgboost import XGBClassifier
>>> from sedna.algorithms.multi_task_learning import MulTaskLearning
>>> estimator = XGBClassifier(objective="binary:logistic")
>>> task_definition = {
    "method": "TaskDefinitionByDataAttr",
    "param": {"attribute": ["season", "city"]}
}
>>> task_relationship_discovery = {
    "method": "DefaultTaskRelationDiscover", "param": {}
}
>>> seen_task_allocation = {
    "method": "TaskAllocationByDataAttr",
    "param": {"attribute": ["season", "city"]}
}
>>> task_remodeling = None
>>> inference_integrate = {
    "method": "DefaultInferenceIntegrate", "param": {}
}
>>> mul_task_instance = MulTaskLearning(
    estimator=estimator,
    task_definition=task_definition,
    task_relationship_discovery=task_relationship_discovery,
    seen_task_allocation=seen_task_allocation,
    task_remodeling=task_remodeling,
    inference_integrate=inference_integrate
)
```

Notes

All method defined under *task_jobs* and registered in *ClassFactory*.

train(*train_data*: *sedna.datasources.BaseDataSource*, *valid_data*: *sedna.datasources.BaseDataSource* = *None*, *post_process*=*None*, ***kwargs*)

fit for update the knowledge based on training data.

Parameters

- **train_data** ([BaseDataSource](#)) – Train data, see *sedna.datasources.BaseDataSource* for more detail.
- **valid_data** ([BaseDataSource](#)) – Valid data, [BaseDataSource](#) or `None`.
- **post_process** (*function*) – function or a registered method, callback after *estimator* train.
- **kwargs** (*Dict*) – parameters for *estimator* training, Like: *early_stopping_rounds* in `Xgboost.XGBClassifier`

Returns

- **feedback** (*Dict*) – contain all training result in each tasks.
- **task_index_url** (*str*) – task extractor model path, used for task allocation.

update(*tasks*, *task_update_strategies*, ***kwargs*)

Parameters:

tasks: `List[Task]`

from the output of module `task_update_decision`

task_update_strategies: `object`

from the output of module `task_update_decision`

returns

task_index – updated seen task index of knowledge base

rtype

`Dict`

load(*task_index*)

load `task_detail` (tasks/models etc ...) from task index file. It'll automatically loaded during *inference* and *evaluation* phases.

Parameters

task_index (*str* or *Dict*) – task index file path, default `self.task_index_url`.

predict(*data*: [sedna.datasources.BaseDataSource](#), *post_process*=`None`, ***kwargs*)

predict the result for input data based on training knowledge.

Parameters

- **data** ([BaseDataSource](#)) – inference sample, see *sedna.datasources.BaseDataSource* for more detail.
- **post_process** (*function*) – function or a registered method, effected after *estimator* prediction, like: label transform.
- **kwargs** (*Dict*) – parameters for *estimator* predict, Like: *ntree_limit* in `Xgboost.XGBClassifier`

Returns

- **result** (*array_like*) – results array, contain all inference results in each sample.
- **tasks** (*List*) – tasks assigned to each sample.

evaluate(*data*: [sedna.datasources.BaseDataSource](#), *metrics*=None, *metrics_param*=None, ***kwargs*)
evaluated the performance of each task from training, filter tasks based on the defined rules.

Parameters

- **data** ([BaseDataSource](#)) – valid data, see *sedna.datasources.BaseDataSource* for more detail.
- **metrics** (*function* / *str*) – Metrics to assess performance on the task by given prediction.
- **metrics_param** (*Dict*) – parameter for metrics function.
- **kwargs** (*Dict*) – parameters for *estimator* evaluate, Like: *ntree_limit* in *Xgboost.XGBClassifier*

Returns

- **task_eval_res** (*Dict*) – all metric results.
- **tasks_detail** (*List[Object]*) – all metric results in each task.

`lib.sedna.algorithms.transmitter`

Submodules

`lib.sedna.algorithms.transmitter.transmitter`

Module Contents

Classes

<i>AbstractTransmitter</i>	Abstract class of Transmitter, which provides base transmission
<i>WSTransmitter</i>	An implementation of Transmitter based on WebSocket.
<i>S3Transmitter</i>	An implementation of Transmitter based on S3 protocol.

class `lib.sedna.algorithms.transmitter.transmitter.AbstractTransmitter`

Bases: `abc.ABC`

Abstract class of Transmitter, which provides base transmission interfaces between edge and cloud.

abstract `recv()`

abstract `send(data)`

class `lib.sedna.algorithms.transmitter.transmitter.WSTransmitter`

Bases: [*AbstractTransmitter*](#), `abc.ABC`

An implementation of Transmitter based on WebSocket.

recv()

send(data)

```
class lib.sedna.algorithms.transmitter.transmitter.S3Transmitter(s3_endpoint_url, access_key,  
                                                                secret_key, transmitter_url)
```

Bases: *AbstractTransmitter*, abc.ABC

An implementation of Transmitter based on S3 protocol.

recv()

send(*data*)

Package Contents

Classes

<i>S3Transmitter</i>	An implementation of Transmitter based on S3 protocol.
<i>WSTransmitter</i>	An implementation of Transmitter based on WebSocket.

```
class lib.sedna.algorithms.transmitter.S3Transmitter(s3_endpoint_url, access_key, secret_key,  
                                                    transmitter_url)
```

Bases: AbstractTransmitter, abc.ABC

An implementation of Transmitter based on S3 protocol.

recv()

send(*data*)

```
class lib.sedna.algorithms.transmitter.WSTransmitter
```

Bases: AbstractTransmitter, abc.ABC

An implementation of Transmitter based on WebSocket.

recv()

send(*data*)

```
lib.sedna.algorithms.unseen_task_detect
```

Submodules

```
lib.sedna.algorithms.unseen_task_detect.unseen_task_detect
```

Unseen task detection algorithms for Lifelong Learning

Module Contents

Classes

<i>ModelProbeFilter</i>	Judgment based on the confidence of the prediction result,
<i>TaskAttrFilter</i>	Judgment based on whether the metadata of the sample has been found in KB

class lib.sedna.algorithms.unseen_task_detect.unseen_task_detect.**ModelProbeFilter**

Bases: BaseFilter, abc.ABC

Judgment based on the confidence of the prediction result, typically used for classification problems

__call__ (*tasks*: List[[sedna.algorithms.multi_task_learning.task_jobs.artifact.Task](#)] = None, *threshold*=0.5, ***kwargs*)

Parameters

- **tasks** (*inference task*) –
- **threshold** (*float*) – threshold considered credible

Returns

is unseen task – *True* means unseen task, *False* means not.

Return type

bool

class lib.sedna.algorithms.unseen_task_detect.unseen_task_detect.**TaskAttrFilter**

Bases: BaseFilter, abc.ABC

Judgment based on whether the metadata of the sample has been found in KB

__call__ (*tasks*: List[[sedna.algorithms.multi_task_learning.task_jobs.artifact.Task](#)] = None, ***kwargs*)

Parameters

- **tasks** (*inference task*) –

Returns

is unseen task – *True* means unseen task, *False* means not.

Return type

bool

Package Contents

Classes

<i>ModelProbeFilter</i>	Judgment based on the confidence of the prediction result,
<i>TaskAttrFilter</i>	Judgment based on whether the metadata of the sample has been found in KB

class lib.sedna.algorithms.unseen_task_detect.**ModelProbeFilter**

Bases: BaseFilter, abc.ABC

Judgment based on the confidence of the prediction result, typically used for classification problems

__call__(tasks: List[[sedna.algorithms.multi_task_learning.task_jobs.artifact.Task](#)] = None, threshold=0.5, **kwargs)

Parameters

- **tasks** (*inference task*) –
- **threshold** (*float*) – threshold considered credible

Returns

is unseen task – *True* means unseen task, *False* means not.

Return type

bool

class lib.sedna.algorithms.unseen_task_detect.**TaskAttrFilter**

Bases: BaseFilter, abc.ABC

Judgment based on whether the metadata of the sample has been found in KB

__call__(tasks: List[[sedna.algorithms.multi_task_learning.task_jobs.artifact.Task](#)] = None, **kwargs)

Parameters

tasks (*inference task*) –

Returns

is unseen task – *True* means unseen task, *False* means not.

Return type

bool

lib.sedna.algorithms.unseen_task_detection

Subpackages

lib.sedna.algorithms.unseen_task_detection.unseen_sample_re_recognition

Submodules

lib.sedna.algorithms.unseen_task_detection.unseen_sample_re_recognition.
base_unseen_sample_re_recognition

Divide labeled unseen samples into seen tasks and unseen tasks.

param task_index

knowledge base index which includes indexes of tasks, samples, models, etc.

type task_index

str or Dict

returns

- **seen_task_samples** (seen samples, see [sedna.datasources.BaseDataSource](#))
- *for more detail*

- **unseen_task_samples** (unseen samples, see *sedna.datasources.BaseDataSource*)
- *for more detail*

Module Contents

Classes

<i>BaseSampleReRegonition</i>	Divide labeled unseen samples into seen tasks and unseen tasks.
-------------------------------	---

```
class lib.sedna.algorithms.unseen_task_detection.unseen_sample_re_recognition.base_unseen_sample_re_recognition
```

Divide labeled unseen samples into seen tasks and unseen tasks.

Parameters

task_index (*str* or *Dict*) – knowledge base index which includes indexes of tasks, samples, models, etc.

abstract `__call__` (*samples*: *sedna.datasources.BaseDataSource*)

Parameters

samples (*training samples*) –

Returns

- **seen_task_samples** (*BaseDataSource*)
- **unseen_task_samples** (*BaseDataSource*)

```
lib.sedna.algorithms.unseen_task_detection.unseen_sample_re_recognition.  
unseen_sample_re_recognition
```

Module Contents

Classes

<i>SampleReRegonitionDefault</i>	Divide labeled unseen samples into seen tasks and unseen tasks.
----------------------------------	---

```
class lib.sedna.algorithms.unseen_task_detection.unseen_sample_re_recognition.unseen_sample_re_recognition
```

Bases: *lib.sedna.algorithms.unseen_task_detection.unseen_sample_re_recognition.
base_unseen_sample_re_recognition.BaseSampleReRegonition*

Divide labeled unseen samples into seen tasks and unseen tasks.

Parameters

task_index (*str* or *Dict*) – knowledge base index which includes indexes of tasks, samples, models, etc.

__call__ (*samples*: [sedna.datasources.BaseDataSource](#))

Parameters

samples (*training samples*) –

Returns

- **seen_task_samples** (*BaseDataSource*)
- **unseen_task_samples** (*BaseDataSource*)

[lib.sedna.algorithms.unseen_task_detection.unseen_sample_recognition](#)

Submodules

[lib.sedna.algorithms.unseen_task_detection.unseen_sample_recognition.
base_unseen_sample_recognition](#)

Divide inference samples into seen samples and unseen samples

param task_index

knowledge base index which includes indexes of tasks, samples and etc.

type task_index

str or dict

returns

- **seen_task_samples** (seen samples, see [sedna.datasources.BaseDataSource](#))
- *for more detail*
- **unseen_task_samples** (unseen samples, see [sedna.datasources.BaseDataSource](#))
- *for more detail*

Module Contents

Classes

BaseSampleRegonition	Divide inference samples into seen samples and unseen samples
--------------------------------------	---

class [lib.sedna.algorithms.unseen_task_detection.unseen_sample_recognition.base_unseen_sample_recognition](#)

Divide inference samples into seen samples and unseen samples

Parameters

task_index (*str or dict*) – knowledge base index which includes indexes of tasks, samples and etc.

abstract __call__ (*samples*: [sedna.datasources.BaseDataSource](#)) →

[Tuple](#)[[sedna.datasources.BaseDataSource](#), [sedna.datasources.BaseDataSource](#)]

Parameters

samples ([BaseDataSource](#)) – inference samples

Returns

- `seen_task_samples` (*BaseDataSource*)
- `unseen_task_samples` (*BaseDataSource*)

```
lib.sedna.algorithms.unseen_task_detection.unseen_sample_recognition.  
unseen_sample_detection
```

Module Contents**Classes**

<i>UnseenSampleDetection</i>	Divide inference samples into seen samples and unseen samples
------------------------------	---

```
class lib.sedna.algorithms.unseen_task_detection.unseen_sample_recognition.unseen_sample_detection.UnseenSampleDetection:
```

Bases: `threading.Thread`

Divide inference samples into seen samples and unseen samples

Parameters

task_index (*str or dict*) – knowledge base index which includes indexes of tasks, samples and etc.

run()

Method representing the thread's activity.

You may override this method in a subclass. The standard `run()` method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the `args` and `kwargs` arguments, respectively.

get_index(*samples, timestamp*)

get_environ_varia()

```
lib.sedna.algorithms.unseen_task_detection.unseen_sample_recognition.  
unseen_sample_recognition
```

Module Contents**Classes**

<i>SampleRegonitionDefault</i>	Divide inference samples into seen samples and unseen samples
--------------------------------	---

```
class lib.sedna.algorithms.unseen_task_detection.unseen_sample_recognition.unseen_sample_recognition.SampleRegonitionDefault:
```

Bases: `lib.sedna.algorithms.unseen_task_detection.unseen_sample_recognition.base_unseen_sample_recognition.BaseSampleRegonition`

Divide inference samples into seen samples and unseen samples

Parameters

task_index (*str or dict*) – knowledge base index which includes indexes of tasks, samples and etc.

__call__ (*samples: sedna.datasources.BaseDataSource*) → Tuple[*sedna.datasources.BaseDataSource*, *sedna.datasources.BaseDataSource*]

Parameters

samples (*BaseDataSource*) – inference samples

Returns

- **seen_task_samples** (*BaseDataSource*)
- **unseen_task_samples** (*BaseDataSource*)

`lib.sedna.algorithms.unseen_task_processing`

Subpackages

`lib.sedna.algorithms.unseen_task_processing.unseen_task_allocation`

Submodules

`lib.sedna.algorithms.unseen_task_processing.unseen_task_allocation.
base_unseen_task_allocation`

Mining tasks of inference unseen sample base on unseen task attribute extractor

param samples infer unseen sample

:param : :param see *sedna.datasources.BaseDataSource* for more detail.:

returns

allocations

rtype

tasks that assigned to each sample

Module Contents

Classes

BaseUnseenTaskAllocation

Task allocation for unseen data

class `lib.sedna.algorithms.unseen_task_processing.unseen_task_allocation.base_unseen_task_allocation.Ba`

Task allocation for unseen data

Parameters

task_extractor (*Dict*) – used to match target tasks

```
abstract __call__(samples: sedna.datasources.BaseDataSource)
```

Parameters

samples (*samples to be allocated*) –

Returns

- **samples** (*BaseDataSource*) – grouped samples based on allocations
- **allocations** (*List*) – allocation decision for actual inference

```
lib.sedna.algorithms.unseen_task_processing.unseen_task_allocation.unseen_task_allocation
```

Module Contents**Classes**

<i>UnseenTaskAllocationDefault</i>

Task allocation for unseen data

```
class lib.sedna.algorithms.unseen_task_processing.unseen_task_allocation.unseen_task_allocation.UnseenTaskAllocation
```

Bases: *lib.sedna.algorithms.unseen_task_processing.unseen_task_allocation.base_unseen_task_allocation.BaseUnseenTaskAllocation*

Task allocation for unseen data

Parameters

task_extractor (*Dict*) – used to match target tasks

```
__call__(samples: sedna.datasources.BaseDataSource)
```

Parameters

samples (*samples to be allocated*) –

Returns

- **samples** (*BaseDataSource*)
- **allocations** (*List*) – allocation decision for actual inference

Submodules

```
lib.sedna.algorithms.unseen_task_processing.unseen_task_processing
```

Module Contents**Classes**

<i>UnseenTaskProcessing</i>

Process unseen tasks given task update strategies

```
class lib.sedna.algorithms.unseen_task_processing.unseen_task_processing.UnseenTaskProcessing(estimator,
                                                                                          un-
                                                                                          seen_task_a
                                                                                          **kwargs)
```

Process unseen tasks given task update strategies

Parameters:

estimator: Instance

An instance with the high-level API that greatly simplifies machine learning programming. Estimators encapsulate training, evaluation, prediction, and exporting for your model.

cloud_knowledge_management: Instance of class CloudKnowledgeManagement unseen_task_allocation: Dict

Mining tasks of unseen inference sample.

train()

Intialize unseen task groups

Returns: res: Dict

evaluation result.

task_index: Dict or str

unseen task index which includes models, samples, extractor and etc.

update(*tasks*, *task_update_strategies*, ***kwargs*)

Parameters:

tasks: List[Task]

from the output of module task_update_decision

task_update_strategies: Dict

from the output of module task_update_decision

returns

task_index – updated unseen task index of knowledge base

rtype

Dict

predict(*data*, *post_process=None*, ***kwargs*)

Predict the result for unseen data.

Parameters

- **data** ([BaseDataSource](#)) – inference sample, see *sedna.datasources.BaseDataSource* for more detail.
- **post_process** (*function*) – function or a registered method, effected after *estimator* prediction, like: label transform.

Returns

- **result** (*array_like*) – results array, contain all inference results in each sample.

- **tasks** (*List*) – tasks assigned to each sample.

load(*task_index*)

load task_detail (tasks/models etc ...) from task index file. It'll automatically loaded during *inference* phases.

Parameters

task_index_url (*str*) – task index file path.

Package Contents

Classes

<i>UnseenTaskProcessing</i>	Process unseen tasks given task update strategies
-----------------------------	---

class lib.sedna.algorithms.unseen_task_processing.**UnseenTaskProcessing**(*estimator, unseen_task_allocation=None, **kwargs*)

Process unseen tasks given task update strategies

Parameters:

estimator: Instance

An instance with the high-level API that greatly simplifies machine learning programming. Estimators encapsulate training, evaluation, prediction, and exporting for your model.

cloud_knowledge_management: Instance of class CloudKnowledgeManagement unseen_task_allocation: Dict

Mining tasks of unseen inference sample.

train()

Intialize unseen task groups

Returns: res: Dict

evaluation result.

task_index: Dict or str

unseen task index which includes models, samples, extractor and etc.

update(*tasks, task_update_strategies, **kwargs*)

Parameters:

tasks: List[Task]

from the output of module task_update_decision

task_update_strategies: Dict

from the output of module task_update_decision

returns

task_index – updated unseen task index of knowledge base

rtype
Dict

predict(*data*, *post_process=None*, ***kwargs*)

Predict the result for unseen data.

Parameters

- **data** ([BaseDataSource](#)) – inference sample, see *sedna.datasources.BaseDataSource* for more detail.
- **post_process** (*function*) – function or a registered method, effected after *estimator* prediction, like: label transform.

Returns

- **result** (*array_like*) – results array, contain all inference results in each sample.
- **tasks** (*List*) – tasks assigned to each sample.

load(*task_index*)

load task_detail (tasks/models etc ...) from task index file. It'll automatically loaded during *inference* phases.

Parameters

task_index_url (*str*) – task index file path.

19.1.2 lib.sedna.backend

Framework Backend class.

Subpackages

`lib.sedna.backend.mindspore`

Package Contents

Classes

MSBackend

ML Framework Backend base Class

class `lib.sedna.backend.mindspore.MSBackend`(*estimator*, *fine_tune=True*, ***kwargs*)

Bases: `sedna.backend.base.BackendBase`

ML Framework Backend base Class

train(*train_data*, *valid_data=None*, ***kwargs*)

Train model.

predict(*data*, ***kwargs*)

Inference model.

evaluate(*data*, ***kwargs*)

evaluate model.

finetune()
 todo: no support yet

load_weights()

get_weights()
 todo: no support yet

set_weights(weights)
 todo: no support yet

lib.sedna.backend.tensorflow

Package Contents

Classes

<i>TFBackend</i>	Tensorflow Framework Backend base Class
<i>KerasBackend</i>	Keras Framework Backend base Class

Attributes

<i>ConfigProto</i>

lib.sedna.backend.tensorflow.ConfigProto

class lib.sedna.backend.tensorflow.**TFBackend**(*estimator, fine_tune=True, **kwargs*)

 Bases: sedna.backend.base.BackendBase

 Tensorflow Framework Backend base Class

train(*train_data, valid_data=None, **kwargs*)
 Train model.

predict(*data, **kwargs*)
 Inference model.

evaluate(*data, **kwargs*)
 evaluate model.

finetune()
 todo: no support yet

load_weights()

get_weights()
 todo: no support yet

set_weights(weights)
 todo: no support yet

```
model_info(model, relpath=None, result=None)
```

```
class lib.sedna.backend.tensorflow.KerasBackend(estimator, fine_tune=True, **kwargs)
```

Bases: [*TFBackend*](#)

Keras Framework Backend base Class

```
set_session()
```

```
finetune()
```

todo: no support yet

```
get_weights()
```

todo: no support yet

```
set_weights(weights)
```

todo: no support yet

lib.sedna.backend.torch

Package Contents

Classes

[*TorchBackend*](#)

ML Framework Backend base Class

```
class lib.sedna.backend.torch.TorchBackend(estimator, fine_tune=True, **kwargs)
```

Bases: `sedna.backend.base.BackendBase`

ML Framework Backend base Class

```
evaluate(**kwargs)
```

evaluate model.

```
train(**kwargs)
```

Not implemented!

```
predict(data, **kwargs)
```

Inference model.

```
load(model_url="", model_name=None, **kwargs)
```

Submodules

lib.sedna.backend.base

Module Contents

Classes

[*BackendBase*](#)

ML Framework Backend base Class

```
class lib.sedna.backend.base.BackendBase(estimator, fine_tune=True, **kwargs)
    ML Framework Backend base Class

    property model_name

    static parse_kwargs(func, **kwargs)

    train(*args, **kwargs)
        Train model.

    update(*args, **kwargs)
        Update model.

    predict(*args, **kwargs)
        Inference model.

    predict_proba(*args, **kwargs)
        Compute probabilities of possible outcomes for samples in X.

    evaluate(*args, **kwargs)
        evaluate model.

    save(model_url="", model_name=None)

    model_info(model, relpath=None, result=None)

    load(model_url="", model_name=None, **kwargs)

    abstract set_weights(weights)
        Set weight with memory tensor.

    abstract get_weights()
        Get the weights.
```

Package Contents

Functions

<code>set_backend([estimator, config])</code>	Create Trainer class
---	----------------------

```
lib.sedna.backend.set_backend(estimator=None, config=None)
    Create Trainer class
```

19.1.3 lib.sedna.common

Submodules

`lib.sedna.common.class_factory`

Management class registration and bind configuration properties, provides the type of class supported.

Module Contents

Classes

<i>ClassType</i>	Const class saved defined class type.
<i>ClassFactory</i>	A Factory Class to manage all class need to register with config.

class lib.sedna.common.class_factory.**ClassType**

Const class saved defined class type.

GENERAL = 'general'

HEM = 'hard_example_mining'

FL_AGG = 'aggregation'

MTL = 'multi_task_learning'

UTD = 'unseen_task_detect'

OF = 'optical_flow'

ALGORITHM = 'algorithm'

DATASET = 'data_process'

CALLBACK = 'post_process_callback'

UTP = 'unseen_task_processing'

KM = 'knowledge_management'

STP = 'seen_task_processing'

class lib.sedna.common.class_factory.**ClassFactory**

Bases: object

A Factory Class to manage all class need to register with config.

__registry__

classmethod register(*type_name=ClassType.GENERAL, alias=None*)

Register class into registry.

Parameters

- **type_name** – type_name: type name of class registry
- **alias** – alias of class name

Returns

wrapper

classmethod register_cls(*t_cls, type_name=ClassType.GENERAL, alias=None*)

Register class with type name.

Parameters

- **t_cls** – class need to register.

- **type_name** – type name.
- **alias** – class name.

Returns**classmethod** **register_from_package**(*package*, *type_name=ClassType.GENERAL*)

Register all public class from package.

Parameters

- **package** – package need to register.
- **type_name** – type name.

Returns**classmethod** **is_exists**(*type_name*, *cls_name=None*)

Determine whether class name is in the current type group.

Parameters

- **type_name** – type name of class registry
- **cls_name** – class name

Returns

True/False

classmethod **get_cls**(*type_name*, *t_cls_name=None*)

Get class and bind config to class.

Parameters

- **type_name** – type name of class registry
- **t_cls_name** – class name

Returns

t_cls

lib.sedna.common.config**Module Contents****Classes**

<i>BaseConfig</i>	The base config
<i>Context</i>	The Context provides the capability of obtaining the context

class lib.sedna.common.config.**BaseConfig**

Bases: ConfigSerializable

The base config

device_category**backend_type**

```

lc_server
original_dataset_url
train_dataset_url
test_dataset_url
data_path_prefix
namespace
worker_name
service_name
job_name
pretrained_model_url
model_url
model_name
log_level
transmitter
agg_data_path
s3_endpoint_url
access_key_id
secret_access_key
parameters

```

```
class lib.sedna.common.config.Context
```

The Context provides the capability of obtaining the context

```
parameters
```

```
classmethod get_parameters(param, default=None)
```

get the value of the key *param* in *PARAMETERS*, if not exist, the default value is returned

```
classmethod get_algorithm_from_api(algorithm, **param) → dict
```

get the algorithm and parameter from api

```
lib.sedna.common.constant
```

Module Contents

Classes

<i>K8sResourceKind</i>	Sedna job/service kind
<i>K8sResourceKindStatus</i>	Job/Service status
<i>KBResourceConstant</i>	Knowledge used constant

```
class lib.sedna.common.constant.K8sResourceKind
    Bases: enum.Enum
    Sedna job/service kind
    DEFAULT = 'default'
    REID_JOB = 'reidjob'
    VIDEO_ANALYTICS_JOB = 'videoanalyticsjob'
    FEATURE_EXTRACTION_SERVICE = 'featureextractionservice'
    JOINT_INFERENCE_SERVICE = 'jointinferenceservice'
    FEDERATED_LEARNING_JOB = 'federatedlearningjob'
    INCREMENTAL_JOB = 'incrementallearningjob'
    LIFELONG_JOB = 'lifelonglearningjob'

class lib.sedna.common.constant.K8sResourceKindStatus
    Bases: enum.Enum
    Job/Service status
    COMPLETED = 'completed'
    FAILED = 'failed'
    RUNNING = 'running'

class lib.sedna.common.constant.KBResourceConstant
    Bases: enum.Enum
    Knowledge used constant
    MIN_TRAIN_SAMPLE = 10
    KB_INDEX_NAME = 'index.pkl'
    TASK_EXTRACTOR_NAME = 'task_attr_extractor.pkl'
    SEEN_TASK = 'seen_task'
    UNSEEN_TASK = 'unseen_task'
    TASK_GROUPS = 'task_groups'
    EXTRACTOR = 'extractor'
    EDGE_KB_DIR = '/var/lib/sedna/kb'
```


lib.sedna.common.file_ops

FileOps class.

Module Contents**Classes***FileOps*

This is a class with some class methods

class lib.sedna.common.file_ops.FileOps

This is a class with some class methods to handle some files or folder.

classmethod make_dir(*args)

Make new a local directory.

Parameters

args (*) – list of str path to joined as a new directory to make.

classmethod get_file_hash(filepath)**classmethod clean_folder(target, clean=True)**

clean the target directories. create path if *target* not exists, initial path if *clean* be True

Parameters

- **target** (*list*) – list of str path need to clean.
- **clean** (*bool*) – clear target if exists.

classmethod delete(path)**classmethod make_base_dir(*args)**

Make new a base directory.

Parameters

args (*) – list of str path to joined as a

new base directory to make.

classmethod join_path(*args)

Join list of path and return.

Parameters

args (*) – list of str path to be joined.

Returns

joined path str.

Return type

str

classmethod remove_path_prefix(org_str: str, prefix: str)

remove the prefix, for converting path in container to path in host.

classmethod **dump_pickle**(*obj, filename*)

Dump a object to a file using pickle.

Parameters

- **obj** (*object*) – target object.
- **filename** (*str*) – target pickle file path.

classmethod **load_pickle**(*filename*)

Load a pickle file and return the object.

Parameters

filename (*str*) – target pickle file path.

Returns

return the loaded original object.

Return type

object or None.

classmethod **copy_folder**(*src, dst*)

Copy a folder from source to destination.

Parameters

- **src** (*str*) – source path.
- **dst** (*str*) – destination path.

classmethod **copy_file**(*src, dst*)

Copy a file from source to destination.

Parameters

- **src** (*str*) – source path.
- **dst** (*str*) – destination path.

classmethod **dump**(*obj, dst=None*) → str

classmethod **load**(*src: str*)

classmethod **is_remote**(*src*)

classmethod **download**(*src, dst=None, unzip=False*) → str

classmethod **upload**(*src, dst, tar=False, clean=True*) → str

classmethod **is_local**(*src*)

classmethod **gcs_download**(*src, dst*)

todo: not support now

classmethod **gcs_upload**(*src, dst*)

todo: not support now

classmethod **s3_download**(*src, dst*)

classmethod **s3_upload**(*src, dst*)

classmethod `http_download(src, dst)`

Download data from http or https web site.

Parameters

- **src** (*str*) – the data path
- **dst** (*str*) – the data path

Raises

FileNotFoundError – if the file path is not exist, an error will raise

classmethod `exists(folder)`

Is folder existed or not.

Parameters

folder (*str*) – folder

Returns

folder existed or not.

Return type

bool

classmethod `obj_to_pickle_string(x)`

classmethod `pickle_string_to_obj(s)`

`lib.sedna.common.log`

Base logger

Module Contents

Classes

<i>Logger</i>	Deafult logger in sedna
---------------	-------------------------

Attributes

<i>LOG_LEVEL</i>
<i>LOGGER</i>

`lib.sedna.common.log.LOG_LEVEL`

class `lib.sedna.common.log.Logger(name: str = BaseConfig.job_name)`

Deafult logger in sedna :param name: Logger name, default is 'sedna' :type name: str

`lib.sedna.common.log.LOGGER`

`lib.sedna.common.utils`

This script contains some common tools.

Module Contents

Functions

<code>get_host_ip()</code>	Get local ip address.
<code>singleton(cls)</code>	Set class to singleton class.

`lib.sedna.common.utils.get_host_ip()`

Get local ip address.

`lib.sedna.common.utils.singleton(cls)`

Set class to singleton class.

Parameters

cls – class

Returns

instance

19.1.4 `lib.sedna.core`

Subpackages

`lib.sedna.core.federated_learning`

Submodules

`lib.sedna.core.federated_learning.federated_learning`

Module Contents

Classes

<code>FederatedLearning</code>	Federated learning enables multiple actors to build a common, robust
<code>FederatedLearningV2</code>	

class `lib.sedna.core.federated_learning.federated_learning.FederatedLearning`(*estimator*,
aggregation='FedAvg')

Bases: `sedna.core.base.JobBase`

Federated learning enables multiple actors to build a common, robust machine learning model without sharing data, thus allowing to address critical issues such as data privacy, data security, data access rights and access to heterogeneous data.

Sedna provide the related interfaces for application development.

Parameters

- **estimator** (*Instance*) – An instance with the high-level API that greatly simplifies machine learning programming. Estimators encapsulate training, evaluation, prediction, and exporting for your model.
- **aggregation** (*str*) – aggregation algo which has registered to ClassFactory, see *sedna.algorithms.aggregation* for more detail.

Examples

```
>>> Estimator = keras.models.Sequential()
>>> fl_model = FederatedLearning(
    estimator=Estimator,
    aggregation="FedAvg"
)
```

register(*timeout=300*)

Deprecated, Client proactively subscribes to the aggregation service.

Parameters

timeout (*int*, *connect timeout. Default: 300*) –

train(*train_data*, *valid_data=None*, *post_process=None*, ***kwargs*)

Training task for FederatedLearning

Parameters

- **train_data** (*BaseDataSource*) – datasource use for train, see *sedna.datasources.BaseDataSource* for more detail.
- **valid_data** (*BaseDataSource*) – datasource use for evaluation, see *sedna.datasources.BaseDataSource* for more detail.
- **post_process** (*function or a registered method*) – effected after *estimator* training.
- **kwargs** (*Dict*) – parameters for *estimator* training, Like: *early_stopping_rounds* in Xgboost.XGBClassifier

```
class lib.sedna.core.federated_learning.federated_learning.FederatedLearningV2(data=None,
                                                                              estima-
                                                                              tor=None,
                                                                              aggrega-
                                                                              tion=None,
                                                                              transmit-
                                                                              ter=None)
```

```
classmethod get_transmitter_from_config()
```

```
train()
```

Package Contents

Classes

<i>FederatedLearning</i>	Federated learning enables multiple actors to build a common, robust
<i>FederatedLearningV2</i>	

class lib.sedna.core.federated_learning.**FederatedLearning**(*estimator*, *aggregation*='FedAvg')

Bases: sedna.core.base.JobBase

Federated learning enables multiple actors to build a common, robust machine learning model without sharing data, thus allowing to address critical issues such as data privacy, data security, data access rights and access to heterogeneous data.

Sedna provide the related interfaces for application development.

Parameters

- **estimator** (*Instance*) – An instance with the high-level API that greatly simplifies machine learning programming. Estimators encapsulate training, evaluation, prediction, and exporting for your model.
- **aggregation** (*str*) – aggregation algo which has registered to ClassFactory, see *sedna.algorithms.aggregation* for more detail.

Examples

```
>>> Estimator = keras.models.Sequential()
>>> fl_model = FederatedLearning(
    estimator=Estimator,
    aggregation="FedAvg"
)
```

register(*timeout*=300)

Deprecated, Client proactively subscribes to the aggregation service.

Parameters

timeout (*int*, *connect timeout. Default: 300*) –

train(*train_data*, *valid_data*=None, *post_process*=None, ***kwargs*)

Training task for FederatedLearning

Parameters

- **train_data** (*BaseDataSource*) – datasource use for train, see *sedna.datasources.BaseDataSource* for more detail.
- **valid_data** (*BaseDataSource*) – datasource use for evaluation, see *sedna.datasources.BaseDataSource* for more detail.
- **post_process** (*function or a registered method*) – effected after *estimator* training.
- **kwargs** (*Dict*) – parameters for *estimator* training, Like: *early_stopping_rounds* in Xgboost.XGBClassifier

```
class lib.sedna.core.federated_learning.FederatedLearningV2(data=None, estimator=None,
                                                            aggregation=None,
                                                            transmitter=None)
```

```
    classmethod get_transmitter_from_config()
```

```
    train()
```

```
lib.sedna.core.incremental_learning
```

Submodules

```
lib.sedna.core.incremental_learning.incremental_learning
```

Module Contents

Classes

IncrementalLearning

Incremental learning is a method of machine learning in which input data

```
class lib.sedna.core.incremental_learning.incremental_learning.IncrementalLearning(estimator,
                                                                                    hard_example_mining:
                                                                                    dict =
                                                                                    None)
```

Bases: `sedna.core.base.JobBase`

Incremental learning is a method of machine learning in which input data is continuously used to extend the existing model's knowledge i.e. to further train the model. It represents a dynamic technique of supervised learning and unsupervised learning that can be applied when training data becomes available gradually over time.

Sedna provide the related interfaces for application development.

Parameters

- **estimator** (*Instance*) – An instance with the high-level API that greatly simplifies machine learning programming. Estimators encapsulate training, evaluation, prediction, and exporting for your model.
- **hard_example_mining** (*Dict*) – HEM algorithms with parameters which has registered to ClassFactory, see `sedna.algorithms.hard_example_mining` for more detail.

Examples

```
>>> Estimator = keras.models.Sequential()
>>> il_model = IncrementalLearning(
    estimator=Estimator,
    hard_example_mining={
        "method": "IBT",
        "param": {
            "threshold_img": 0.9
        }
    }
)
```

Notes

Sedna provide an interface call `get_hem_algorithm_from_config` to build the `hard_example_mining` parameter from CRD definition.

classmethod `get_hem_algorithm_from_config(**param)`

get the *algorithm* name and *param* of `hard_example_mining` from `crd`

Parameters

param (*Dict*) – update value in parameters of `hard_example_mining`

Returns

e.g.: {"method": "IBT", "param": {"threshold_img": 0.5}}

Return type

dict

Examples

```
>>> IncrementalLearning.get_hem_algorithm_from_config(
    threshold_img=0.9
)
{"method": "IBT", "param": {"threshold_img": 0.9}}
```

train(*train_data*, *valid_data*=None, *post_process*=None, ****kwargs**)

Training task for `IncrementalLearning`

Parameters

- **train_data** (*BaseDataSource*) – datasource use for train, see *sedna.datasources.BaseDataSource* for more detail.
- **valid_data** (*BaseDataSource*) – datasource use for evaluation, see *sedna.datasources.BaseDataSource* for more detail.
- **post_process** (*function or a registered method*) – effected after *estimator* training.
- **kwargs** (*Dict*) – parameters for *estimator* training, Like: *early_stopping_rounds* in `Xgboost.XGBClassifier`

Return type

estimator

inference(*data=None, post_process=None, **kwargs*)

Inference task for IncrementalLearning

Parameters

- **data** ([BaseDataSource](#)) – datasource use for inference, see *sedna.datasources.BaseDataSource* for more detail.
- **post_process** (*function or a registered method*) – effected after *estimator* inference.
- **kwargs** (*Dict*) – parameters for *estimator* inference, Like: *ntree_limit* in *Xgboost.XGBClassifier*

Returns

- **inference result** (*object*)
- **result after post_process** (*object*)
- **if is hard sample** (*bool*)

evaluate(*data, post_process=None, **kwargs*)

Evaluate task for IncrementalLearning

Parameters

- **data** ([BaseDataSource](#)) – datasource use for evaluation, see *sedna.datasources.BaseDataSource* for more detail.
- **post_process** (*function or a registered method*) – effected after *estimator* evaluation.
- **kwargs** (*Dict*) – parameters for *estimator* evaluate, Like: *metric_name* in *Xgboost.XGBClassifier*

Returns

evaluate metrics

Return type

List

Package Contents

Classes

IncrementalLearning

Incremental learning is a method of machine learning in which input data

class `lib.sedna.core.incremental_learning.IncrementalLearning`(*estimator, hard_example_mining: dict = None*)

Bases: `sedna.core.base.JobBase`

Incremental learning is a method of machine learning in which input data is continuously used to extend the existing model's knowledge i.e. to further train the model. It represents a dynamic technique of supervised learning and unsupervised learning that can be applied when training data becomes available gradually over time.

Sedna provide the related interfaces for application development.

Parameters

- **estimator** (*Instance*) – An instance with the high-level API that greatly simplifies machine learning programming. Estimators encapsulate training, evaluation, prediction, and exporting for your model.
- **hard_example_mining** (*Dict*) – HEM algorithms with parameters which has registered to ClassFactory, see *sedna.algorithms.hard_example_mining* for more detail.

Examples

```
>>> Estimator = keras.models.Sequential()
>>> il_model = IncrementalLearning(
    estimator=Estimator,
    hard_example_mining={
        "method": "IBT",
        "param": {
            "threshold_img": 0.9
        }
    }
)
```

Notes

Sedna provide an interface call *get_hem_algorithm_from_config* to build the *hard_example_mining* parameter from CRD definition.

classmethod *get_hem_algorithm_from_config*(***param*)

get the *algorithm* name and *param* of *hard_example_mining* from crd

Parameters

param (*Dict*) – update value in parameters of *hard_example_mining*

Returns

e.g.: {"method": "IBT", "param": {"threshold_img": 0.5}}

Return type

dict

Examples

```
>>> IncrementalLearning.get_hem_algorithm_from_config(
    threshold_img=0.9
)
{"method": "IBT", "param": {"threshold_img": 0.9}}
```

train(*train_data*, *valid_data*=None, *post_process*=None, ***kwargs*)

Training task for IncrementalLearning

Parameters

- **train_data** (*BaseDataSource*) – datasource use for train, see *sedna.datasources.BaseDataSource* for more detail.

- **valid_data** ([BaseDataSource](#)) – datasource use for evaluation, see *sedna.datasources.BaseDataSource* for more detail.
- **post_process** (*function or a registered method*) – effected after *estimator* training.
- **kwargs** (*Dict*) – parameters for *estimator* training, Like: *early_stopping_rounds* in Xgboost.XGBClassifier

Return type

estimator

inference(*data=None, post_process=None, **kwargs*)

Inference task for IncrementalLearning

Parameters

- **data** ([BaseDataSource](#)) – datasource use for inference, see *sedna.datasources.BaseDataSource* for more detail.
- **post_process** (*function or a registered method*) – effected after *estimator* inference.
- **kwargs** (*Dict*) – parameters for *estimator* inference, Like: *ntree_limit* in Xgboost.XGBClassifier

Returns

- **inference result** (*object*)
- **result after post_process** (*object*)
- **if is hard sample** (*bool*)

evaluate(*data, post_process=None, **kwargs*)

Evaluate task for IncrementalLearning

Parameters

- **data** ([BaseDataSource](#)) – datasource use for evaluation, see *sedna.datasources.BaseDataSource* for more detail.
- **post_process** (*function or a registered method*) – effected after *estimator* evaluation.
- **kwargs** (*Dict*) – parameters for *estimator* evaluate, Like: *metric_name* in Xgboost.XGBClassifier

Returns

evaluate metrics

Return type

List

`lib.sedna.core.joint_inference`

Submodules

`lib.sedna.core.joint_inference.joint_inference`

Module Contents

Classes

<i>BigModelService</i>	Large model services implemented
<i>JointInference</i>	Sedna provide a framework make sure under the condition of limited

class `lib.sedna.core.joint_inference.joint_inference.BigModelService`(*estimator=None*)

Bases: `sedna.core.base.JobBase`

Large model services implemented Provides RESTful interfaces for large-model inference.

Parameters

estimator (*Instance, big model*) – An instance with the high-level API that greatly simplifies machine learning programming. Estimators encapsulate training, evaluation, prediction, and exporting for your model.

Examples

```
>>> Estimator = xgboost.XGBClassifier()
>>> BigModelService(estimator=Estimator).start()
```

start()

Start inference rest server

train(*train_data, valid_data=None, post_process=None, **kwargs*)

todo: no support yet

inference(*data=None, post_process=None, **kwargs*)

Inference task for JointInference

Parameters

- **data** (*BaseDataSource*) – datasource use for inference, see `sedna.datasources.BaseDataSource` for more detail.
- **post_process** (*function or a registered method*) – effected after *estimator* inference.
- **kwargs** (*Dict*) – parameters for *estimator* inference, Like: *ntree_limit* in `Xgboost.XGBClassifier`

Return type

inference result

```
class lib.sedna.core.joint_inference.joint_inference.JointInference(estimator=None,
                                                                    hard_example_mining: dict
                                                                    = None)
```

Bases: `sedna.core.base.JobBase`

Sedna provide a framework make sure under the condition of limited resources on the edge, difficult inference tasks are offloaded to the cloud to improve the overall performance, keeping the throughput.

Parameters

- **estimator** (*Instance*) – An instance with the high-level API that greatly simplifies machine learning programming. Estimators encapsulate training, evaluation, prediction, and exporting for your model.
- **hard_example_mining** (*Dict*) – HEM algorithms with parameters which has registered to ClassFactory, see `sedna.algorithms.hard_example_mining` for more detail.

Examples

```
>>> Estimator = keras.models.Sequential()
>>> ji_service = JointInference(
    estimator=Estimator,
    hard_example_mining={
        "method": "IBT",
        "param": {
            "threshold_img": 0.9
        }
    }
)
```

Notes

Sedna provide an interface call `get_hem_algorithm_from_config` to build the `hard_example_mining` parameter from CRD definition.

```
classmethod get_hem_algorithm_from_config(**param)
```

get the *algorithm* name and *param* of `hard_example_mining` from `crd`

Parameters

param (*Dict*) – update value in parameters of `hard_example_mining`

Returns

e.g.: {"method": "IBT", "param": {"threshold_img": 0.5}}

Return type

dict

Examples

```
>>> JointInference.get_hem_algorithm_from_config(  
    threshold_img=0.9  
)  
{"method": "IBT", "param": {"threshold_img": 0.9}}
```

inference(*data=None, post_process=None, **kwargs*)

Inference task with JointInference

Parameters

- **data** (*BaseDataSource*) – datasource use for inference, see *sedna.datasources.BaseDataSource* for more detail.
- **post_process** (*function or a registered method*) – effected after *estimator* inference.
- **kwargs** (*Dict*) – parameters for *estimator* inference, Like: *ntree_limit* in *Xgboost.XGBClassifier*

Returns

- **if is hard sample** (*bool*)
- **inference result** (*object*)
- **result from little-model** (*object*)
- **result from big-model** (*object*)

Package Contents

Classes

<i>JointInference</i>	Sedna provide a framework make sure under the condition of limited
<i>BigModelService</i>	Large model services implemented

class `lib.sedna.core.joint_inference.JointInference`(*estimator=None, hard_example_mining: dict = None*)

Bases: `sedna.core.base.JobBase`

Sedna provide a framework make sure under the condition of limited resources on the edge, difficult inference tasks are offloaded to the cloud to improve the overall performance, keeping the throughput.

Parameters

- **estimator** (*Instance*) – An instance with the high-level API that greatly simplifies machine learning programming. Estimators encapsulate training, evaluation, prediction, and exporting for your model.
- **hard_example_mining** (*Dict*) – HEM algorithms with parameters which has registered to *ClassFactory*, see *sedna.algorithms.hard_example_mining* for more detail.

Examples

```
>>> Estimator = keras.models.Sequential()
>>> ji_service = JointInference(
    estimator=Estimator,
    hard_example_mining={
        "method": "IBT",
        "param": {
            "threshold_img": 0.9
        }
    }
)
```

Notes

Sedna provide an interface call `get_hem_algorithm_from_config` to build the `hard_example_mining` parameter from CRD definition.

classmethod `get_hem_algorithm_from_config(**param)`

get the *algorithm* name and *param* of `hard_example_mining` from `crd`

Parameters

param (*Dict*) – update value in parameters of `hard_example_mining`

Returns

e.g.: {"method": "IBT", "param": {"threshold_img": 0.5}}

Return type

dict

Examples

```
>>> JointInference.get_hem_algorithm_from_config(
    threshold_img=0.9
)
{"method": "IBT", "param": {"threshold_img": 0.9}}
```

inference(*data=None, post_process=None, **kwargs*)

Inference task with `JointInference`

Parameters

- **data** (`BaseDataSource`) – datasource use for inference, see `sedna.datasources.BaseDataSource` for more detail.
- **post_process** (*function or a registered method*) – effected after *estimator* inference.
- **kwargs** (*Dict*) – parameters for *estimator* inference, Like: *ntree_limit* in `Xgboost.XGBClassifier`

Returns

- **if is hard sample** (*bool*)
- **inference result** (*object*)

- **result from little-model** (*object*)
- **result from big-model** (*object*)

class `lib.sedna.core.joint_inference.BigModelService`(*estimator=None*)

Bases: `sedna.core.base.JobBase`

Large model services implemented Provides RESTful interfaces for large-model inference.

Parameters

estimator (*Instance, big model*) – An instance with the high-level API that greatly simplifies machine learning programming. Estimators encapsulate training, evaluation, prediction, and exporting for your model.

Examples

```
>>> Estimator = xgboost.XGBClassifier()
>>> BigModelService(estimator=Estimator).start()
```

start()

Start inference rest server

train(*train_data, valid_data=None, post_process=None, **kwargs*)

todo: no support yet

inference(*data=None, post_process=None, **kwargs*)

Inference task for JointInference

Parameters

- **data** (*BaseDataSource*) – datasource use for inference, see `sedna.datasources.BaseDataSource` for more detail.
- **post_process** (*function or a registered method*) – effected after *estimator* inference.
- **kwargs** (*Dict*) – parameters for *estimator* inference, Like: *ntree_limit* in `Xgboost.XGBClassifier`

Return type

inference result

`lib.sedna.core.lifelong_learning`

Subpackages

`lib.sedna.core.lifelong_learning.knowledge_management`

Submodules

`lib.sedna.core.lifelong_learning.knowledge_management.base_knowledge_management`

Module Contents

Classes

<i>BaseKnowledgeManagement</i>	Base class of knowledge management.
--------------------------------	-------------------------------------

class `lib.sedna.core.lifelong_learning.knowledge_management.base_knowledge_management.BaseKnowledgeManagement`

Base class of knowledge management. It includes model and sample update to knowledge base server.

Parameters: `config`: `BaseConfig`, see ‘`sedna.common.config.BaseConfig`’ for more details.

It sets basic configs for knowledge management.

seen_estimator: Instance

An instance with the high-level API that greatly simplifies machine learning programming. Estimators encapsulate training, evaluation, prediction, and exporting for a model.

unseen_estimator: Instance

An instance with the high-level API that greatly simplifies mechanism model learning programming. Estimators encapsulate training, evaluation, prediction, and exporting for a mechanism model.

abstract `update_kb(task_index)`

abstract `save_task_index(task_index, task_type=None, **kwargs)`

`lib.sedna.core.lifelong_learning.knowledge_management.cloud_knowledge_management`

Module Contents

Classes

<i>CloudKnowledgeManagement</i>	Manage task processing, kb update and task deployment, etc., at cloud.
---------------------------------	--

class `lib.sedna.core.lifelong_learning.knowledge_management.cloud_knowledge_management.CloudKnowledgeManagement`

Bases: `lib.sedna.core.lifelong_learning.knowledge_management.base_knowledge_management.BaseKnowledgeManagement`

Manage task processing, kb update and task deployment, etc., at cloud.

update_kb(`task_index`)

save_task_index(`task_index, task_type='seen_task'`)

evaluate_tasks(*tasks_detail*, ***kwargs*)

Parameters

tasks_detail (*List[Task]*) – output of module `task_update_decision`, consisting of results of evaluation.

Returns

drop_task – names of the tasks which will not to be deployed to the edge.

Return type

`List[str]`

`lib.sedna.core.lifelong_learning.knowledge_management.edge_knowledge_management`

Module Contents

Classes

<i>EdgeKnowledgeManagement</i>	Manage inference, knowledge base update, etc., at the edge.
--------------------------------	---

class `lib.sedna.core.lifelong_learning.knowledge_management.edge_knowledge_management.EdgeKnowledgeManag`

Bases: `lib.sedna.core.lifelong_learning.knowledge_management.base_knowledge_management.BaseKnowledgeManagement`

Manage inference, knowledge base update, etc., at the edge.

update_kb(*task_index*)

save_task_index(*task_index*, *task_type='seen_task'*)

save_unseen_samples(*samples*, *post_process*)

start_services()

Package Contents

Classes

<i>BaseKnowledgeManagement</i>	Base class of knowledge management.
<i>CloudKnowledgeManagement</i>	Manage task processing, kb update and task deployment, etc., at cloud.
<i>EdgeKnowledgeManagement</i>	Manage inference, knowledge base update, etc., at the edge.

```
class lib.sedna.core.lifelong_learning.knowledge_management.BaseKnowledgeManagement(config,
                                                                                   seen_estimator,
                                                                                   un-
                                                                                   seen_estimator)
```

Base class of knowledge management. It includes model and sample update to knowledge base server.

Parameters: *config*: BaseConfig, see 'sedna.common.config.BaseConfig' for more details.

It sets basic configs for knowledge management.

seen_estimator: Instance

An instance with the high-level API that greatly simplifies machine learning programming. Estimators encapsulate training, evaluation, prediction, and exporting for a model.

unseen_estimator: Instance

An instance with the high-level API that greatly simplifies mechanism model learning programming. Estimators encapsulate training, evaluation, prediction, and exporting for a mechanism model.

abstract update_kb(*task_index*)

abstract save_task_index(*task_index*, *task_type*=None, ***kwargs*)

```
class lib.sedna.core.lifelong_learning.knowledge_management.CloudKnowledgeManagement(config,
                                                                                   seen_estimator,
                                                                                   un-
                                                                                   seen_estimator,
                                                                                   **kwargs)
```

Bases: [lib.sedna.core.lifelong_learning.knowledge_management.base_knowledge_management.BaseKnowledgeManagement](#)

Manage task processing, kb update and task deployment, etc., at cloud.

update_kb(*task_index*)

save_task_index(*task_index*, *task_type*='seen_task')

evaluate_tasks(*tasks_detail*, ***kwargs*)

Parameters

tasks_detail (*List[Task]*) – output of module task_update_decision, consisting of results of evaluation.

Returns

drop_task – names of the tasks which will not to be deployed to the edge.

Return type

List[str]

```
class lib.sedna.core.lifelong_learning.knowledge_management.EdgeKnowledgeManagement(config,
                                                                                   seen_estimator,
                                                                                   un-
                                                                                   seen_estimator,
                                                                                   **kwargs)
```

Bases: [lib.sedna.core.lifelong_learning.knowledge_management.base_knowledge_management.BaseKnowledgeManagement](#)

Manage inference, knowledge base update, etc., at the edge.

```
update_kb(task_index)

save_task_index(task_index, task_type='seen_task')

save_unseen_samples(samples, post_process)

start_services()
```

Submodules

`lib.sedna.core.lifelong_learning.lifelong_learning`

Module Contents

Classes

<i>LifelongLearning</i>	Lifelong Learning (LL) is an advanced machine learning (ML) paradigm that
-------------------------	---

```
class lib.sedna.core.lifelong_learning.lifelong_learning.LifelongLearning(seen_estimator, un-
    seen_estimator=None,
    task_definition=None,
    task_relationship_discovery=None,
    task_allocation=None,
    task_remodeling=None,
    inference_integrate=None,
    task_update_decision=None,
    unseen_task_allocation=None,
    unseen_sample_recognition=None,
    unseen_sample_re_recognition=None)
```

Bases: `sedna.core.base.JobBase`

Lifelong Learning (LL) is an advanced machine learning (ML) paradigm that learns continuously, accumulates the knowledge learned in the past, and uses/adapts it to help future learning and problem solving.

Sedna provide the related interfaces for application development.

Parameters

- **estimator** (*Instance*) – An instance with the high-level API that greatly simplifies machine learning programming. Estimators encapsulate training, evaluation, prediction, and exporting for your model.
- **unseen_estimator** (*Instance*) – An instance with the high-level API that greatly simplifies mechanism model learning programming. Estimators encapsulate training, evaluation, prediction, and exporting for your mechanism model.
- **task_definition** (*Dict*) – Divide multiple tasks based on data, see `task_definition.task_definition` for more detail.

- **task_relationship_discovery** (*Dict*) – Discover relationships between all tasks, see *task_relationship_discovery.task_relationship_discovery* for more detail.
- **task_allocation** (*Dict*) – Mining seen tasks of inference sample, see *task_allocation.task_allocation* for more detail.
- **task_remoting** (*Dict*) – Remoting tasks based on their relationships, see *task_remoting.task_remoting* for more detail.
- **inference_integrate** (*Dict*) – Integrate the inference results of all related tasks, see *inference_integrate.inference_integrate* for more detail.
- **task_update_decision** (*Dict*) – Task update strategy making algorithms, see ‘knowledge_management.task_update_decision.task_update_decision’ for more detail.
- **unseen_task_allocation** (*Dict*) – Mining unseen tasks of inference sample, see *unseen_task_processing.unseen_task_allocation.unseen_task_allocation* for more detail.
- **unseen_sample_recognition** (*Dict*) – Dividing inference samples into seen tasks and unseen tasks, see ‘unseen_task_processing.unseen_sample_recognition.unseen_sample_recognition’ for more detail.
- **unseen_sample_re_recognition** (*Dict*) – Dividing unseen training samples into seen tasks and unseen tasks, see ‘unseen_task_processing.unseen_sample_re_recognition.unseen_sample_re_recognition’ for more detail.

Examples

```
>>> estimator = XGBClassifier(objective="binary:logistic")
>>> unseen_estimator = None
>>> task_definition = {
    "method": "TaskDefinitionByDataAttr",
    "param": {"attribute": ["season", "city"]}
}
>>> task_relationship_discovery = {
    "method": "DefaultTaskRelationDiscover", "param": {}
}
>>> task_mining = {
    "method": "TaskMiningByDataAttr",
    "param": {"attribute": ["season", "city"]}
}
>>> task_remoting = None
>>> inference_integrate = {
    "method": "DefaultInferenceIntegrate", "param": {}
}
>>> task_update_decision = {
    "method": "UpdateStrategyDefault", "param": {}
}
>>> unseen_task_allocation = {
    "method": "UnseenTaskAllocationDefault", "param": {}
}
>>> unseen_sample_recognition = {
    "method": "SampleRegonitionDefault", "param": {}
}
>>> unseen_sample_re_recognition = {
    "method": "SampleReRegonitionDefault", "param": {}
}
```

(continues on next page)

(continued from previous page)

```

    }
    >>> ll_jobs = LifelongLearning(
        estimator,
        unseen_estimator=None,
        task_definition=None,
        task_relationship_discovery=None,
        task_allocation=None,
        task_remodeling=None,
        inference_integrate=None,
        task_update_decision=None,
        unseen_task_allocation=None,
        unseen_sample_recognition=None,
        unseen_sample_re_recognition=None,
    )

```

train(*train_data*, *valid_data*=None, *post_process*=None, ***kwargs*)

fit for update the knowledge based on training data.

Parameters

- **train_data** ([BaseDataSource](#)) – Train data, see *sedna.datasources.BaseDataSource* for more detail.
- **valid_data** ([BaseDataSource](#)) – Valid data, [BaseDataSource](#) or None.
- **post_process** (*function*) – function or a registered method, callback after *estimator* train.
- **kwargs** (*Dict*) – parameters for *estimator* training, Like: *early_stopping_rounds* in Xgboost.XGBClassifier

Returns

train_history

Return type

object

update(*train_data*, *valid_data*=None, *post_process*=None, ***kwargs*)

fit for update the knowledge based on incremental data.

Parameters

- **train_data** ([BaseDataSource](#)) – Train data, see *sedna.datasources.BaseDataSource* for more detail.
- **valid_data** ([BaseDataSource](#)) – Valid data, [BaseDataSource](#) or None.
- **post_process** (*function*) – function or a registered method, callback after *estimator* train.
- **kwargs** (*Dict*) – parameters for *estimator* training, Like: *early_stopping_rounds* in Xgboost.XGBClassifier

Returns

train_history

Return type

object

evaluate(*data*, *post_process*=None, ***kwargs*)

evaluated the performance of each task from training, filter tasks based on the defined rules.

Parameters

- **data** ([BaseDataSource](#)) – valid data, see *sedna.datasources.BaseDataSource* for more detail.
- **kwargs** (*Dict*) – parameters for *estimator* evaluate, Like: *ntree_limit* in *Xgboost.XGBClassifier*

inference(*data*=None, *post_process*=None, *unseen_sample_postprocess*=None, ***kwargs*)

predict the result for input data based on training knowledge.

Parameters

- **data** ([BaseDataSource](#)) – inference sample, see *sedna.datasources.BaseDataSource* for more detail.
- **post_process** (*function*) – function or a registered method, effected after *estimator* prediction, like: label transform.
- **unseen_sample_postprocess** (*function*) – function or a registered method, effected when unseen samples need to be saved
- **kwargs** (*Dict*) – parameters for *estimator* predict, Like: *ntree_limit* in *Xgboost.XGBClassifier*

Package Contents

Classes

<i>LifelongLearning</i>	Lifelong Learning (LL) is an advanced machine learning (ML) paradigm that
---	---

```
class lib.sedna.core.lifelong_learning.LifelongLearning(seen_estimator, unseen_estimator=None,
                                                    task_definition=None,
                                                    task_relationship_discovery=None,
                                                    task_allocation=None,
                                                    task_remodeling=None,
                                                    inference_integrate=None,
                                                    task_update_decision=None,
                                                    unseen_task_allocation=None,
                                                    unseen_sample_recognition=None,
                                                    unseen_sample_re_recognition=None)
```

Bases: *sedna.core.base.JobBase*

Lifelong Learning (LL) is an advanced machine learning (ML) paradigm that learns continuously, accumulates the knowledge learned in the past, and uses/adapts it to help future learning and problem solving.

Sedna provide the related interfaces for application development.

Parameters

- **estimator** (*Instance*) – An instance with the high-level API that greatly simplifies machine learning programming. Estimators encapsulate training, evaluation, prediction, and exporting for your model.
- **unseen_estimator** (*Instance*) – An instance with the high-level API that greatly simplifies mechanism model learning programming. Estimators encapsulate training, evaluation, prediction, and exporting for your mechanism model.
- **task_definition** (*Dict*) – Divide multiple tasks based on data, see *task_definition.task_definition* for more detail.
- **task_relationship_discovery** (*Dict*) – Discover relationships between all tasks, see *task_relationship_discovery.task_relationship_discovery* for more detail.
- **task_allocation** (*Dict*) – Mining seen tasks of inference sample, see *task_allocation.task_allocation* for more detail.
- **task_remodeling** (*Dict*) – Remodeling tasks based on their relationships, see *task_remodeling.task_remodeling* for more detail.
- **inference_integrate** (*Dict*) – Integrate the inference results of all related tasks, see *inference_integrate.inference_integrate* for more detail.
- **task_update_decision** (*Dict*) – Task update strategy making algorithms, see ‘knowledge_management.task_update_decision.task_update_decision’ for more detail.
- **unseen_task_allocation** (*Dict*) – Mining unseen tasks of inference sample, see *unseen_task_processing.unseen_task_allocation.unseen_task_allocation* for more detail.
- **unseen_sample_recognition** (*Dict*) – Dividing inference samples into seen tasks and unseen tasks, see ‘unseen_task_processing.unseen_sample_recognition.unseen_sample_recognition’ for more detail.
- **unseen_sample_re_recognition** (*Dict*) – Dividing unseen training samples into seen tasks and unseen tasks, see ‘unseen_task_processing.unseen_sample_re_recognition.unseen_sample_re_recognition’ for more detail.

Examples

```
>>> estimator = XGBClassifier(objective="binary:logistic")
>>> unseen_estimator = None
>>> task_definition = {
    "method": "TaskDefinitionByDataAttr",
    "param": {"attribute": ["season", "city"]}
}
>>> task_relationship_discovery = {
    "method": "DefaultTaskRelationDiscover", "param": {}
}
>>> task_mining = {
    "method": "TaskMiningByDataAttr",
    "param": {"attribute": ["season", "city"]}
}
>>> task_remodeling = None
>>> inference_integrate = {
    "method": "DefaultInferenceIntegrate", "param": {}
}
>>> task_update_decision = {
```

(continues on next page)

(continued from previous page)

```

        "method": "UpdateStrategyDefault", "param": {}
    }
    >>> unseen_task_allocation = {
        "method": "UnseenTaskAllocationDefault", "param": {}
    }
    >>> unseen_sample_recognition = {
        "method": "SampleRegonitionDefault", "param": {}
    }
    >>> unseen_sample_re_recognition = {
        "method": "SampleReRegonitionDefault", "param": {}
    }
    >>> ll_jobs = LifelongLearning(
        estimator,
        unseen_estimator=None,
        task_definition=None,
        task_relationship_discovery=None,
        task_allocation=None,
        task_remodeling=None,
        inference_integrate=None,
        task_update_decision=None,
        unseen_task_allocation=None,
        unseen_sample_recognition=None,
        unseen_sample_re_recognition=None,
    )

```

train(*train_data*, *valid_data*=None, *post_process*=None, ***kwargs*)

fit for update the knowledge based on training data.

Parameters

- **train_data** ([BaseDataSource](#)) – Train data, see *sedna.datasources.BaseDataSource* for more detail.
- **valid_data** ([BaseDataSource](#)) – Valid data, [BaseDataSource](#) or None.
- **post_process** (*function*) – function or a registered method, callback after *estimator* train.
- **kwargs** (*Dict*) – parameters for *estimator* training, Like: *early_stopping_rounds* in Xgboost.XGBClassifier

Returns

train_history

Return type

object

update(*train_data*, *valid_data*=None, *post_process*=None, ***kwargs*)

fit for update the knowledge based on incremental data.

Parameters

- **train_data** ([BaseDataSource](#)) – Train data, see *sedna.datasources.BaseDataSource* for more detail.
- **valid_data** ([BaseDataSource](#)) – Valid data, [BaseDataSource](#) or None.

- **post_process** (*function*) – function or a registered method, callback after *estimator* train.
- **kwargs** (*Dict*) – parameters for *estimator* training, Like: *early_stopping_rounds* in Xgboost.XGBClassifier

Returns**train_history****Return type**

object

evaluate(*data*, *post_process*=None, ***kwargs*)

evaluated the performance of each task from training, filter tasks based on the defined rules.

Parameters

- **data** ([BaseDataSource](#)) – valid data, see *sedna.datasources.BaseDataSource* for more detail.
- **kwargs** (*Dict*) – parameters for *estimator* evaluate, Like: *ntree_limit* in Xgboost.XGBClassifier

inference(*data*=None, *post_process*=None, *unseen_sample_postprocess*=None, ***kwargs*)

predict the result for input data based on training knowledge.

Parameters

- **data** ([BaseDataSource](#)) – inference sample, see *sedna.datasources.BaseDataSource* for more detail.
- **post_process** (*function*) – function or a registered method, effected after *estimator* prediction, like: label transform.
- **unseen_sample_postprocess** (*function*) – function or a registered method, effected when unseen samples need to be saved
- **kwargs** (*Dict*) – parameters for *estimator* predict, Like: *ntree_limit* in Xgboost.XGBClassifier

lib.sedna.core.multi_edge_inference**Subpackages****lib.sedna.core.multi_edge_inference.components****Submodules****lib.sedna.core.multi_edge_inference.components.detector****Module Contents****Classes**

*ObjectDetector*In MultiEdgeInference, the Object Detection/Tracking component

```

class lib.sedna.core.multi_edge_inference.components.detector.ObjectDetector(
    consumer_topics=['enriched_object_detection'],
    producer_topics=['object_detection'],
    plugins:
        List[lib.sedna.core.multi_edge_inference.components.pluggable_network_service.PluggableNetworkService] = [],
    models:
        List[lib.sedna.core.multi_edge_inference.components.pluggable_model.PluggableModel] = [],
    timeout=10,
    asynchronous=False)

```

Bases: `sedna.core.multi_edge_inference.components.BaseService`, `sedna.core.multi_edge_inference.components.FileOperations`

In MultiEdgeInference, the Object Detection/Tracking component is deployed as a service at the edge and it is used to detect or track objects (for example, pedestrians) and send the result to the cloud for further processing using Kafka or REST API.

Parameters

- **consumer_topics** (*List*) – A list of Kafka topics used to communicate with the Feature Extraction service (to receive data from it). This is accessed only if the Kafka backend is in use.
- **producer_topics** (*List*) – A list of Kafka topics used to communicate with the Feature Extraction service (to send data to it). This is accessed only if the Kafka backend is in use.
- **plugins** (*List*) – A list of `PluggableNetworkService`. It can be left empty as the Object-Detector service is already preconfigured to connect to the correct network services.
- **models** (*List*) – A list of `PluggableModel`. By passing a specific instance of the model, it is possible to customize the `ObjectDetector` to, for example, track different objects as long as the `PluggableModel` interface is respected.
- **timeout** (*int*) – It sets a timeout condition to terminate the main fetch loop after the specified amount of seconds has passed since we received the last frame.
- **asynchronous** (*bool*) – If `True`, the AI processing will be decoupled from the data acquisition step. If `False`, the processing will be sequential. In general, set it to `True` when ingesting a stream (e.g., RTSP) and to `False` when reading from disk (e.g., a video file).

Examples

```
model = ByteTracker() # A class implementing the PluggableModel abstract class (example in pedestrian_tracking/detector/model/bytetracker.py)
objecttracking_service = ObjectDetector(models=[model], asynchronous=True)
```

Notes

For the parameters described above, only ‘models’ has to be defined, while for others the default value will work in most cases.

process_data(*ai*, *data*, ***kwargs*)

The user needs to implement this function to call the main processing function of the AI model and decide what to do with the result.

preprocess(*data*)

The user can override this function to inject some preprocessing operation to be executed before the data is added to the data structure by the ‘put()’ function.

close()

update_operational_mode(*status*)

The user needs to trigger updates to the AI model, if necessary.

`lib.sedna.core.multi_edge_inference.components.feature_extraction`

Module Contents

Classes

<i>FEService</i>	In MultiEdgeInference, the Feature Extraction component
------------------	---

```
class lib.sedna.core.multi_edge_inference.components.feature_extraction.FEService(consumer_topics=['object_tracking'], producer_topics=['enriched_object_tracking'], plugins: List[sedna.core.multi_edge_inference.plugins.PluggableModel] = [], models: List[sedna.core.multi_edge_inference.models.PluggableModel] = [], time_out=10, asynchronous=False)
```

Bases: `sedna.core.multi_edge_inference.components.BaseService`

In MultiEdgeInference, the Feature Extraction component is deployed in the edge or the cloud and it used to extract ReID features from frames received by the ObjectDetector component and send back to it the enriched data using Kafka or REST API.

Parameters

- **consumer_topics** (*List*) – A list of Kafka topics used to communicate with the Object Detector service (to receive data from it). This is accessed only if the Kafka backend is in use.
- **producer_topics** (*List*) – A list of Kafka topics used to communicate with the Object Detector service (to send data to it). This is accessed only if the Kafka backend is in use.
- **plugins** (*List*) – A list of PluggableNetworkService. It can be left empty as the Feature-Extraction service is already preconfigured to connect to the correct network services.
- **models** (*List*) – A list of PluggableModel. By passing a specific instance of the model, it is possible to customize the FeatureExtraction component to, for example, extract differently the objects features.
- **timeout** (*int*) – It sets a timeout condition to terminate the main fetch loop after the specified amount of seconds has passed since we received the last frame.
- **asynchronous** (*bool*) – If True, the AI processing will be decoupled from the data acquisition step. If False, the processing will be sequential. In general, set it to True when ingesting a stream (e.g., RTSP) and to False when reading from disk (e.g., a video file).

Examples

```
model = FeatureExtractionAI() # A class implementing the PluggableModel abstract class (example pedestrian_tracking/feature_extraction/worker.py)
fe_service = FEService(models=[model], asynchronous=False)
```

Notes

For the parameters described above, only ‘models’ has to be defined, while for others the default value will work in most cases.

process_data(*ai, data, **kwargs*)

The user needs to implement this function to call the main processing function of the AI model and decide what to do with the result.

update_operational_mode(*status*)

The user needs to trigger updates to the AI model, if necessary.

get_target_features(*ldata*)

`lib.sedna.core.multi_edge_inference.components.reid`

Module Contents

Classes

ReID

In MultiEdgeInference, the ReID component is deployed in the cloud

```
class lib.sedna.core.multi_edge_inference.components.reid.ReID(consumer_topics=[],
                                                                producer_topics=[], plugins:
                                                                List[lib.sedna.core.multi_edge_inference.plugins.PluggableNetworkService]
                                                                = [], models:
                                                                List[lib.sedna.core.multi_edge_inference.plugins.PluggableModel]
                                                                = [], timeout=10,
                                                                asynchronous=True)
```

Bases: `sedna.core.multi_edge_inference.components.BaseService`, `sedna.core.multi_edge_inference.components.FileOperations`

In MultiEdgeInference, the ReID component is deployed in the cloud and it used to identify a target by comparing its features with the ones generated from the Feature Extraction component.

Parameters

- **consumer_topics** (*List*) – Leave empty.
- **producer_topics** (*List*) – Leave empty.
- **plugins** (*List*) – A list of `PluggableNetworkService`. It can be left empty as the ReID component is already preconfigured to connect to the correct network services.
- **models** (*List*) – A list of `PluggableModel`. In this case we abuse of the term model as the ReID doesn't really use an AI model but rather a wrapper for the ReID functions.
- **timeout** (*int*) – It sets a timeout condition to terminate the main fetch loop after the specified amount of seconds has passed since we received the last frame.
- **asynchronous** (*bool*) – If True, the AI processing will be decoupled from the data acquisition step. If False, the processing will be sequential. In general, set it to True when ingesting a stream (e.g., RTSP) and to False when reading from disk (e.g., a video file).

Examples

```
model = ReIDWorker() # A class implementing the PluggableModel abstract class (example in pedestrian_tracking/reid/worker.py)
```

```
self.job = ReID(models=[model], asynchronous=False)
```

Notes

For the parameters described above, only 'models' has to be defined, while for others the default value will work in most cases.

process_data(*ai, data, **kwargs*)

The user needs to implement this function to call the main processing function of the AI model and decide what to do with the result.

update_operational_mode(*status*)

The user needs to trigger updates to the AI model, if necessary.

get_target_features(*ldata*)

Package Contents

Classes

<i>BaseService</i>	Base MultiEdgeInference wrapper for video analytics, feature extraction,
<i>FileOperations</i>	Class containing file operations to read/write from disk.

Attributes

<i>POLL_INTERVAL</i>

```
lib.sedna.core.multi_edge_inference.components.POLL_INTERVAL = 0.01
```

```
class lib.sedna.core.multi_edge_inference.components.BaseService(consumer_topics=[],
                                                                    producer_topics=[], plugins:
                                                                    List[lib.sedna.core.multi_edge_inference.plugins.Plugin]
                                                                    = [], models:
                                                                    List[lib.sedna.core.multi_edge_inference.plugins.Plugin]
                                                                    = [], timeout=10,
                                                                    asynchronous=False)
```

Bases: `abc.ABC`

Base MultiEdgeInference wrapper for video analytics, feature extraction, and reid components.

put(*data*)

Call this function to push data into the component. For example, after you extract a frame from video stream, you can call `put(image)`. Depending on the value of the ‘asynchronous’ parameter, the data will be put into a different data structure.

fetch_data()

get_plugin(*plugin_key*: `lib.sedna.core.multi_edge_inference.plugins.PLUGIN`)

This function allows to select the network service to communicate to based on the name (given that is has been registered before). List of registered plugins can be found in `plugins/registered.py`.

flatten(*S*)

distribute_data(*data*=[], ***kwargs*)

This function sends the data to all the AI models passed to with this component during the initialization phase.

abstract process_data(*ai*, *data*, ***kwargs*)

The user needs to implement this function to call the main processing function of the AI model and decide what to do with the result.

abstract update_operational_mode(*status*)

The user needs to trigger updates to the AI model, if necessary.

preprocess(*data*, ***kwargs*)

The user can override this function to inject some preprocessing operation to be executed before the data is added to the data structure by the ‘put()’ function.

class `lib.sedna.core.multi_edge_inference.components.FileOperations`

Class containing file operations to read/write from disk.

read_from_disk(*path*)

delete_from_disk(*filename*)

write_to_disk(*data*, *folder*, *exts*='.dat')

get_files_list(*folder*)

`lib.sedna.core.multi_edge_inference.plugins`

Submodules

`lib.sedna.core.multi_edge_inference.plugins.registered`

Module Contents

Classes

<i>ReID_Server</i>	Abstract class to wrap a REST service.
<i>ReID_I</i>	Abstract class to wrap a REST service.
<i>Feature_Extraction</i>	Abstract class to wrap a REST service.
<i>Feature_Extraction_I</i>	Abstract class to wrap a REST service.
<i>VideoAnalytics</i>	Abstract class to wrap a REST service.
<i>VideoAnalytics_I</i>	Abstract class to wrap a REST service.

class `lib.sedna.core.multi_edge_inference.plugins.registered.ReID_Server`(*ip*=`get_parameters('REID_MODEL_BIND_IP')`,
get_host_ip()
port=`get_parameters('REID_MODEL_BIND_PORT')`,
'5000',
wrapper=`None`)

Bases: `sedna.core.multi_edge_inference.plugins.PluggableNetworkService`

Abstract class to wrap a REST service.

class `lib.sedna.core.multi_edge_inference.plugins.registered.ReID_I`(*ip*=`get_parameters('REID_MODEL_BIND_IP')`,
'reid-reid',
port=`get_parameters('REID_MODEL_BIND_PORT')`,
'5000')

Bases: `sedna.core.multi_edge_inference.plugins.PluggableNetworkService`

Abstract class to wrap a REST service.

class `lib.sedna.core.multi_edge_inference.plugins.registered.Feature_Extraction`(*ip*=`get_parameters('FE_MODEL_BIND_IP')`,
get_host_ip()
port=`get_parameters('FE_MODEL_BIND_PORT')`,
'6000',
wrapper=`None`)

Bases: `sedna.core.multi_edge_inference.plugins.PluggableNetworkService`

Abstract class to wrap a REST service.

```
class lib.sedna.core.multi_edge_inference.plugins.registered.Feature_Extraction_I(ip=get_parameters('FE_MODEL_IP'),
                                         port=get_parameters('FE_MODEL_PORT'),
                                         wrapper=None)
```

Bases: `sedna.core.multi_edge_inference.plugins.PluggableNetworkService`

Abstract class to wrap a REST service.

```
class lib.sedna.core.multi_edge_inference.plugins.registered.VideoAnalytics(ip=get_parameters('DET_MODEL_IP'),
                                         port=get_parameters('DET_MODEL_PORT'),
                                         wrapper=None)
```

Bases: `sedna.core.multi_edge_inference.plugins.PluggableNetworkService`

Abstract class to wrap a REST service.

```
class lib.sedna.core.multi_edge_inference.plugins.registered.VideoAnalytics_I(ip=get_parameters('DET_MODEL_IP'),
                                         port=get_parameters('DET_MODEL_PORT'),
                                         wrapper=None)
```

Bases: `sedna.core.multi_edge_inference.plugins.PluggableNetworkService`

Abstract class to wrap a REST service.

Package Contents

Classes

<i>PLUGIN</i>	Generic enumeration.
<i>PluggableNetworkService</i>	Abstract class to wrap a REST service.
<i>PluggableModel</i>	Abstract class to wrap and AI model.

Attributes

<i>MODEL_NOT_FOUND</i>
--

```
lib.sedna.core.multi_edge_inference.plugins.MODEL_NOT_FOUND = 'MODEL_UNKNOWN'
```

```
class lib.sedna.core.multi_edge_inference.plugins.PLUGIN
```

Bases: enum.Enum

Generic enumeration.

Derive from this class to define new enumerations.

```
REID_MANAGER = 'ReIDManager'
```

```
REID_MANAGER_I = 'ReIDManager_I'
```

```
REID = 'ReID_Server'
```

```
REID_I = 'ReID_I'
```

```
FEATURE_EXTRACTION = 'Feature_Extraction'
```

```
FEATURE_EXTRACTION_I = 'Feature_Extraction_I'
```

```
VIDEO_ANALYTICS = 'VideoAnalytics'
```

```
VIDEO_ANALYTICS_I = 'VideoAnalytics_I'
```

```
class lib.sedna.core.multi_edge_inference.plugins.PluggableNetworkService(ip, port, plugin_api:  
                                                                    object = None)
```

Bases: abc.ABC

Abstract class to wrap a REST service.

```
class lib.sedna.core.multi_edge_inference.plugins.PluggableModel
```

Bases: abc.ABC

Abstract class to wrap and AI model.

```
property model_path
```

```
property model_name
```

```
abstract load(**kwargs)
```

```
abstract update_plugin(update_object, **kwargs)
```

```
abstract evaluate(**kwargs)
```

```
abstract train(**kwargs)
```

```
inference(data=None, post_process=None, **kwargs)
```

Calls the model 'predict' function

```
evaluate(data, post_process=None, **kwargs)
```

Submodules

`lib.sedna.core.multi_edge_inference.data_classes`

Module Contents

Classes

<i>OP_MODE</i>	Generic enumeration.
<i>DetTrackResult</i>	Base data object exchanged by the MultiEdgeInference components.
<i>TargetImages</i>	
<i>Target</i>	

```
class lib.sedna.core.multi_edge_inference.data_classes.OP_MODE
```

Bases: `enum.Enum`

Generic enumeration.

Derive from this class to define new enumerations.

DETECTION = 'detection'

TRACKING = 'tracking'

COVID19 = 'covid19'

NOOP = 'noop'

```
class lib.sedna.core.multi_edge_inference.data_classes.DetTrackResult(frame_index: int = 0,
                                                                    bbox: List = None,
                                                                    scene=None, confidence:
                                                                    List = None,
                                                                    detection_time: List =
                                                                    None, camera: int = 0,
                                                                    bbox_coord: List = [],
                                                                    tracking_ids: List = [],
                                                                    features: List = [],
                                                                    is_target: bool = False,
                                                                    ID: List = [])
```

Base data object exchanged by the MultiEdgeInference components.

```
class lib.sedna.core.multi_edge_inference.data_classes.TargetImages(userid, targets=[])
```

```
class lib.sedna.core.multi_edge_inference.data_classes.Target(_userid, _features,
                                                                _targetid='0000',
                                                                _tracking_id=None,
                                                                _location=None, _frame_index=0)
```

`lib.sedna.core.multi_edge_inference.utils`

Module Contents

Functions

get_parameters(param[, default])

`lib.sedna.core.multi_edge_inference.utils.get_parameters(param, default=None)`

Package Contents

Classes

<i>BaseService</i>	Base MultiEdgeInference wrapper for video analytics, feature extraction,
<i>FileOperations</i>	Class containing file operations to read/write from disk.

Attributes

POLL_INTERVAL

`lib.sedna.core.multi_edge_inference.POLL_INTERVAL = 0.01`

`class lib.sedna.core.multi_edge_inference.BaseService(consumer_topics=[], producer_topics=[],
plugins:
List[sedna.core.multi_edge_inference.plugins.PluggableNetwork]
= [], models:
List[sedna.core.multi_edge_inference.plugins.PluggableModel]
= [], timeout=10, asynchronous=False)`

Bases: `abc.ABC`

Base MultiEdgeInference wrapper for video analytics, feature extraction, and reid components.

put(data)

Call this function to push data into the component. For example, after you extract a frame from video stream, you can call `put(image)`. Depending on the value of the ‘asynchronous’ parameter, the data will be put into a different data structure.

fetch_data()

get_plugin(plugin_key: `sedna.core.multi_edge_inference.plugins.PLUGIN`)

This function allows to select the network service to communicate to based on the name (given that is has been registered before). List of registered plugins can be found in `plugins/registered.py`.

flatten(S)

distribute_data(*data*=[], ***kwargs*)

This function sends the data to all the AI models passed to with this component during the initialization phase.

abstract process_data(*ai*, *data*, ***kwargs*)

The user needs to implement this function to call the main processing function of the AI model and decide what to do with the result.

abstract update_operational_mode(*status*)

The user needs to trigger updates to the AI model, if necessary.

preprocess(*data*, ***kwargs*)

The user can override this function to inject some preprocessing operation to be executed before the data is added to the data structure by the 'put()' function.

class lib.sedna.core.multi_edge_inference.**FileOperations**

Class containing file operations to read/write from disk.

read_from_disk(*path*)

delete_from_disk(*filename*)

write_to_disk(*data*, *folder*, *exts*='.dat')

get_files_list(*folder*)

Submodules

lib.sedna.core.base

Module Contents

Classes

JobBase

sedna feature base class

class lib.sedna.core.base.**JobBase**(*estimator*, *config*=None)

sedna feature base class

property *model_path*

parameters

abstract **train**(***kwargs*)

inference(*x*=None, *post_process*=None, ***kwargs*)

evaluate(*data*, *post_process*=None, ***kwargs*)

get_parameters(*param*, *default*=None)

report_task_info(*task_info*, *status*, *results*=None, *kind*='train')

19.1.5 lib.sedna.datasources

Subpackages

`lib.sedna.datasources.kafka`

Submodules

`lib.sedna.datasources.kafka.consumer`

Module Contents

Classes

<i>Consumer</i>	Helper class that provides a standard way to create an ABC using
-----------------	--

```
class lib.sedna.datasources.kafka.consumer.Consumer(address=['localhost'], port=[9092],
                                                    group_id='default',
                                                    consumer_timeout_ms=250)
```

Bases: `sedna.datasources.kafka.Client`

Helper class that provides a standard way to create an ABC using inheritance.

connect(*bootstrap_servers*)

get_topics()

subscribe(*topic*)

consume_messages()

consume_messages_poll()

pause()

resume()

close()

`lib.sedna.datasources.kafka.kafka_manager`

Module Contents

Classes

<i>KafkaProducer</i>	
----------------------	--

<i>KafkaConsumerThread</i>	A class that represents a thread of control.
----------------------------	--

```
class lib.sedna.datasources.kafka.kafka_manager.KafkaProducer(address, port, topic=[],
                                                             asynchronous=False)
```

```
    write_result(data)
```

```
class lib.sedna.datasources.kafka.kafka_manager.KafkaConsumerThread(address, port, topic=[],
                                                                    callback=None)
```

Bases: `threading.Thread`

A class that represents a thread of control.

This class can be safely subclassed in a limited fashion. There are two ways to specify the activity: by passing a callable object to the constructor, or by overriding the `run()` method in a subclass.

run()

Method representing the thread's activity.

You may override this method in a subclass. The standard `run()` method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the `args` and `kwargs` arguments, respectively.

lib.sedna.datasources.kafka.producer

Module Contents

Classes

<i>Producer</i>	Helper class that provides a standard way to create an ABC using
-----------------	--

```
class lib.sedna.datasources.kafka.producer.Producer(address=['localhost'], port=[9092])
```

Bases: `sedna.datasources.kafka.Client`

Helper class that provides a standard way to create an ABC using inheritance.

```
connect(bootstrap_servers)
```

```
publish_data_synchronous(data, topic='default')
```

```
publish_data_asynchronous(data, topic='default')
```

```
on_send_success(record_metadata)
```

```
on_send_error(excp)
```

```
close()
```

Package Contents

Classes

<i>Client</i>	Helper class that provides a standard way to create an ABC using
<i>AdminClient</i>	Helper class that provides a standard way to create an ABC using

```
class lib.sedna.datasources.kafka.Client(address=['localhost'], port=[9092])
```

Bases: abc.ABC

Helper class that provides a standard way to create an ABC using inheritance.

```
abstract connect(bootstrap_servers)
```

```
hardened_connect()
```

```
class lib.sedna.datasources.kafka.AdminClient(address=['localhost'], port=[9092])
```

Bases: *Client*

Helper class that provides a standard way to create an ABC using inheritance.

```
create_topics(topics, num_partitions=1, replication_factor=1)
```

```
delete_topics(topics, num_partitions=1, replication_factor=1)
```

```
lib.sedna.datasources.obs
```

Submodules

```
lib.sedna.datasources.obs.connector
```

Module Contents

Classes

<i>OBSClientWrapper</i>

```
class lib.sedna.datasources.obs.connector.OBSClientWrapper(file_server_url: string = ", vendor:  
string = ", region: string = ",  
bucket_name: string = ", app_token:  
string = ")
```

```
download_single_object(remote_path, local_path='.', failed_count=1, selected_index=0)
```

```
list_objects(remote_path, next_marker="")
```

```
upload_file(local_folder_absolute_path, filename, bucket_path, failed_count=1, selected_index=0)
```


Package Contents

Classes

<i>BaseDataSource</i>	An abstract class representing a <i>BaseDataSource</i> .
<i>TxtDataParse</i>	txt file which contain image list parser
<i>CSVDataParse</i>	csv file which contain Structured Data parser
<i>JSONDataParse</i>	json file which contain Structured Data parser

class lib.sedna.datasources.**BaseDataSource**(*data_type='train', func=None*)

An abstract class representing a *BaseDataSource*.

All datasets that represent a map from keys to data samples should subclass it. All subclasses should overwrite `parse`, supporting get train/eval/infer data by a function. Subclasses could also optionally overwrite `__len__`, which is expected to return the size of the dataset. overwrite `x` for the feature-embedding, `y` for the target label.

Parameters

- **data_type** (*str*) – define the datasource is train/eval/test
- **func** (*function*) – function use to parse an iter object batch by batch

property `is_test_data`

num_examples() → int

__len__()

abstract parse(*args, **kwargs)

save(*output=""*)

class lib.sedna.datasources.**TxtDataParse**(*data_type, func=None*)

Bases: *BaseDataSource*, abc.ABC

txt file which contain image list parser

parse(*args, **kwargs)

class lib.sedna.datasources.**CSVDataParse**(*data_type, func=None*)

Bases: *BaseDataSource*, abc.ABC

csv file which contain Structured Data parser

static parse_json(*lines: dict, **kwargs*) → pandas.DataFrame

parse(*args, **kwargs)

class lib.sedna.datasources.**JSONDataParse**(*data_type, func=None*)

Bases: *BaseDataSource*, abc.ABC

json file which contain Structured Data parser

parse(*args, **kwargs)

load_anno_from_ids(*id_*)

19.1.6 lib.sedna.service

Subpackages

`lib.sedna.service.multi_edge_inference`

Subpackages

`lib.sedna.service.multi_edge_inference.interface`

Submodules

`lib.sedna.service.multi_edge_inference.interface.detection_endpoint`

Module Contents

Classes

<i>Detection</i>	Endpoint to trigger the Object Tracking component
------------------	---

```
class lib.sedna.service.multi_edge_inference.interface.detection_endpoint.Detection(service_name,
                                                                                       ver-
                                                                                       sion="",
                                                                                       ip='127.0.0.1',
                                                                                       port='8080',
                                                                                       proto-
                                                                                       col='http')
```

Endpoint to trigger the Object Tracking component

check_server_status()

transmit(data, **kwargs)

Transfer enriched tracking object to video analytics job

update_service(data, **kwargs)

`lib.sedna.service.multi_edge_inference.interface.fe_endpoint`

Module Contents

Classes

<i>FE</i>	Endpoint to trigger the Feature Extraction
-----------	--

```
class lib.sedna.service.multi_edge_inference.interface.fe_endpoint.FE(service_name,
                                                                           version="",
                                                                           ip='127.0.0.1',
                                                                           port='8080',
                                                                           protocol='http')
```

Endpoint to trigger the Feature Extraction

check_server_status()

transmit(*data*, ***kwargs*)

Transfer feature vector to FE worker

get_target_features(*data*, ***kwargs*)

Send target images to FE service and receive back the ReID features

update_service(*data*, ***kwargs*)

`lib.sedna.service.multi_edge_inference.interface.reid_endpoint`

Module Contents

Classes

<i>ReID_Endpoint</i>	Endpoint to trigger the ReID
----------------------	------------------------------

```
class lib.sedna.service.multi_edge_inference.interface.reid_endpoint.ReID_Endpoint(service_name,
                                                                                     version="",
                                                                                     ip='127.0.0.1',
                                                                                     port='8080',
                                                                                     proto-
                                                                                     col='http')
```

Endpoint to trigger the ReID

check_server_status()

transmit(*data*: `sedna.core.multi_edge_inference.data_classes.DetTrackResult`, ***kwargs*)

Transfer feature vector to ReID worker

Package Contents

Classes

<i>Detection</i>	Endpoint to trigger the Object Tracking component
<i>FE</i>	Endpoint to trigger the Feature Extraction
<i>ReID_Endpoint</i>	Endpoint to trigger the ReID

```
class lib.sedna.service.multi_edge_inference.interface.Detection(service_name, version="",
                                                                    ip='127.0.0.1', port='8080',
                                                                    protocol='http')
```

Endpoint to trigger the Object Tracking component

check_server_status()

```
transmit(data, **kwargs)
```

Transfer enriched tracking object to video analytics job

```
update_service(data, **kwargs)
```

```
class lib.sedna.service.multi_edge_inference.interface.FE(service_name, version="", ip='127.0.0.1',  
                                                         port='8080', protocol='http')
```

Endpoint to trigger the Feature Extraction

```
check_server_status()
```

```
transmit(data, **kwargs)
```

Transfer feature vector to FE worker

```
get_target_features(data, **kwargs)
```

Send target images to FE service and receive back the ReID features

```
update_service(data, **kwargs)
```

```
class lib.sedna.service.multi_edge_inference.interface.ReID_Endpoint(service_name, version="",  
                                                                      ip='127.0.0.1',  
                                                                      port='8080',  
                                                                      protocol='http')
```

Endpoint to trigger the ReID

```
check_server_status()
```

```
transmit(data: sedna.core.multi_edge_inference.data_classes.DetTrackResult, **kwargs)
```

Transfer feature vector to ReID worker

```
lib.sedna.service.multi_edge_inference.server
```

Submodules

```
lib.sedna.service.multi_edge_inference.server.detection
```

Module Contents

Classes

DetectionServer

REST api server for object detection component

```
class lib.sedna.service.multi_edge_inference.server.detection.DetectionServer(model,  
                                                                              service_name,  
                                                                              ip: str =  
                                                                              '127.0.0.1',  
                                                                              port: int =  
                                                                              8080,  
                                                                              max_buffer_size:  
                                                                              int =  
                                                                              1004857600,  
                                                                              workers: int =  
                                                                              1)
```

Bases: `sedna.service.server.base.BaseServer`

REST api server for object detection component

start()

status(*request: fastapi.Request*)

async video_analytics(*request: fastapi.Request*)

async update_service(*request: fastapi.Request*)

`lib.sedna.service.multi_edge_inference.server.feature_extraction`

Module Contents

Classes

<i>FEServer</i>	rest api server for feature extraction
-----------------	--

```
class lib.sedna.service.multi_edge_inference.server.feature_extraction.FEServer(model, service_name,
ip: str = '127.0.0.1',
port: int = 8080,
max_buffer_size: int = 1004857600,
workers: int = 1)
```

Bases: `sedna.service.server.base.BaseServer`

rest api server for feature extraction

start()

status(*request: fastapi.Request*)

async feature_extraction(*request: fastapi.Request*)

async get_target_features(*request: fastapi.Request*)

async update_service(*request: fastapi.Request*)

`lib.sedna.service.multi_edge_inference.server.reid`

Module Contents

Classes

<i>ReIDServer</i>	REST api server for reid
-------------------	--------------------------

```
class lib.sedna.service.multi_edge_inference.server.reid.ReIDServer(model, service_name, ip:
                                                                    str = '127.0.0.1', port: int =
                                                                    8080, max_buffer_size: int
                                                                    = 104857600, workers: int
                                                                    = 1)
```

Bases: `sedna.service.server.base.BaseServer`

REST api server for reid

start()

status(request: *fastapi.Request*)

async reid(request: *fastapi.Request*)

Package Contents

Classes

<i>DetectionServer</i>	REST api server for object detection component
<i>FEServer</i>	rest api server for feature extraction
<i>ReIDServer</i>	REST api server for reid

```
class lib.sedna.service.multi_edge_inference.server.DetectionServer(model, service_name, ip:
                                                                    str = '127.0.0.1', port: int =
                                                                    8080, max_buffer_size: int
                                                                    = 1004857600, workers:
                                                                    int = 1)
```

Bases: `sedna.service.server.base.BaseServer`

REST api server for object detection component

start()

status(request: *fastapi.Request*)

async video_analytics(request: *fastapi.Request*)

async update_service(request: *fastapi.Request*)

```
class lib.sedna.service.multi_edge_inference.server.FEServer(model, service_name, ip: str =
                                                                    '127.0.0.1', port: int = 8080,
                                                                    max_buffer_size: int = 1004857600,
                                                                    workers: int = 1)
```

Bases: `sedna.service.server.base.BaseServer`

rest api server for feature extraction

start()

status(*request: fastapi.Request*)

async feature_extraction(*request: fastapi.Request*)

async get_target_features(*request: fastapi.Request*)

async update_service(*request: fastapi.Request*)

```
class lib.sedna.service.multi_edge_inference.server.ReIDServer(model, service_name, ip: str =
'127.0.0.1', port: int = 8080,
max_buffer_size: int =
104857600, workers: int = 1)
```

Bases: `sedna.service.server.base.BaseServer`

REST api server for reid

start()

status(*request: fastapi.Request*)

async reid(*request: fastapi.Request*)

Package Contents

Classes

<i>Detection</i>	Endpoint to trigger the Object Tracking component
<i>FE</i>	Endpoint to trigger the Feature Extraction
<i>ReID_Endpoint</i>	Endpoint to trigger the ReID
<i>DetectionServer</i>	REST api server for object detection component
<i>FEServer</i>	rest api server for feature extraction
<i>ReIDServer</i>	REST api server for reid

```
class lib.sedna.service.multi_edge_inference.Detection(service_name, version="", ip='127.0.0.1',
port='8080', protocol='http')
```

Endpoint to trigger the Object Tracking component

check_server_status()

transmit(*data, **kwargs*)

Transfer enriched tracking object to video analytics job

update_service(*data, **kwargs*)

```
class lib.sedna.service.multi_edge_inference.FE(service_name, version="", ip='127.0.0.1', port='8080',
protocol='http')
```

Endpoint to trigger the Feature Extraction

check_server_status()

transmit(data, ****kwargs**)

Transfer feature vector to FE worker

get_target_features(data, ****kwargs**)

Send target images to FE service and receive back the ReID features

update_service(data, ****kwargs**)

```
class lib.sedna.service.multi_edge_inference.ReID_Endpoint(service_name, version="",  
                                                         ip='127.0.0.1', port='8080',  
                                                         protocol='http')
```

Endpoint to trigger the ReID

check_server_status()

transmit(data: sedna.core.multi_edge_inference.data_classes.DetTrackResult, ****kwargs**)

Transfer feature vector to ReID worker

```
class lib.sedna.service.multi_edge_inference.DetectionServer(model, service_name, ip: str =  
                                                             '127.0.0.1', port: int = 8080,  
                                                             max_buffer_size: int = 1004857600,  
                                                             workers: int = 1)
```

Bases: sedna.service.server.base.BaseServer

REST api server for object detection component

start()

status(request: fastapi.Request)

async video_analytics(request: fastapi.Request)

async update_service(request: fastapi.Request)

```
class lib.sedna.service.multi_edge_inference.FEServer(model, service_name, ip: str = '127.0.0.1',  
                                                       port: int = 8080, max_buffer_size: int =  
                                                       1004857600, workers: int = 1)
```

Bases: sedna.service.server.base.BaseServer

rest api server for feature extraction

start()

status(request: fastapi.Request)

async feature_extraction(request: fastapi.Request)

async get_target_features(request: fastapi.Request)

async update_service(request: fastapi.Request)

```
class lib.sedna.service.multi_edge_inference.ReIDServer(model, service_name, ip: str = '127.0.0.1',  
                                                         port: int = 8080, max_buffer_size: int =  
                                                         104857600, workers: int = 1)
```

Bases: sedna.service.server.base.BaseServer

REST api server for reid


```

start()

status(request: fastapi.Request)

async reid(request: fastapi.Request)

```

```
lib.sedna.service.server
```

Subpackages

```
lib.sedna.service.server.knowledgeBase
```

Submodules

```
lib.sedna.service.server.knowledgeBase.database
```

Module Contents

```

lib.sedna.service.server.knowledgeBase.database.SQLALCHEMY_DATABASE_URL

lib.sedna.service.server.knowledgeBase.database.engine

lib.sedna.service.server.knowledgeBase.database.SessionLocal

lib.sedna.service.server.knowledgeBase.database.Base

```

```
lib.sedna.service.server.knowledgeBase.model
```

Module Contents

Classes

<i>TaskGrp</i>	Task groups
<i>Tasks</i>	Task table
<i>TaskModel</i>	model belong tasks
<i>TaskRelation</i>	relation between two tasks
<i>Samples</i>	Sample storage
<i>TaskSample</i>	Sample of tasks

Functions

<i>get_or_create</i> (session, model, **kwargs)
<i>init_db</i> ()

Attributes

engine

```
lib.sedna.service.server.knowledgeBase.model.engine

class lib.sedna.service.server.knowledgeBase.model.TaskGrp
    Bases: lib.sedna.service.server.knowledgeBase.database.Base
    Task groups
    __tablename__ = 'll_task_grp'
    id
    name
    deploy
    sample_num
    task_num

class lib.sedna.service.server.knowledgeBase.model.Tasks
    Bases: lib.sedna.service.server.knowledgeBase.database.Base
    Task table
    __tablename__ = 'll_tasks'
    id
    name
    task_attr
    created_at
    updated_at
    __repr__()

class lib.sedna.service.server.knowledgeBase.model.TaskModel
    Bases: lib.sedna.service.server.knowledgeBase.database.Base
    model belong tasks
    __tablename__ = 'll_task_models'
    id
    task_id
    task
    model_url
    is_current
```

```

    created_at

class lib.sedna.service.server.knowledgeBase.model.TaskRelation
    Bases: lib.sedna.service.server.knowledgeBase.database.Base
    relation between two tasks
    __tablename__ = 'll_task_relation'

    id
    grp_id
    grp
    task_id
    task
    transfer_radio

class lib.sedna.service.server.knowledgeBase.model.Samples
    Bases: lib.sedna.service.server.knowledgeBase.database.Base
    Sample storage
    __tablename__ = 'll_samples'

    id
    data_url
    descr
    data_type
    updated_at
    sample_num

class lib.sedna.service.server.knowledgeBase.model.TaskSample
    Bases: lib.sedna.service.server.knowledgeBase.database.Base
    Sample of tasks
    __tablename__ = 'll_task_sample'

    id
    sample_id
    sample
    task_id
    task

lib.sedna.service.server.knowledgeBase.model.get_or_create(session, model, **kwargs)
lib.sedna.service.server.knowledgeBase.model.init_db()

```

`lib.sedna.service.server.knowledgeBase.server`

Module Contents

Classes

<i>KBUpdateResult</i>	result
<i>TaskItem</i>	
<i>KBServer</i>	As knowledge base stored in sqlite, this class realizes creation,

```
class lib.sedna.service.server.knowledgeBase.server.KBUpdateResult
```

```
    Bases: pydantic.BaseModel
```

```
    result
```

```
    status: int
```

```
    tasks: Optional[str]
```

```
class lib.sedna.service.server.knowledgeBase.server.TaskItem
```

```
    Bases: pydantic.BaseModel
```

```
    tasks: List
```

```
class lib.sedna.service.server.knowledgeBase.server.KBServer(host: str, http_port: int = 8080,  
                                                             workers: int = 1, save_dir="")
```

```
    Bases: sedna.service.server.base.BaseServer
```

```
    As knowledge base stored in sqlite, this class realizes creation, update and query of the sqlite.
```

```
    start()
```

```
    query()
```

```
    async file_download(files: str, name: str = "")
```

```
    async file_upload(file: fastapi.UploadFile = File(...))
```

```
    update_status(data: KBUpdateResult = Body(...))
```

```
    update(task: fastapi.UploadFile = File(...))
```

Submodules

`lib.sedna.service.server.aggregation`

Module Contents

Classes

AggregationServer

AggregationServerV2

```
class lib.sedna.service.server.aggregation.AggregationServer(aggregation: str, host: str = None,
                                                             http_port: int = None, exit_round:
                                                             int = 1, participants_count: int = 1,
                                                             ws_size: int = 10 * 1024 * 1024)
```

Bases: *lib.sedna.service.server.base.BaseServer*

start()

Start the server

async client_info(request: *starlette.requests.Request*)

```
class lib.sedna.service.server.aggregation.AggregationServerV2(data=None, estimator=None,
                                                                aggregation=None,
                                                                transmitter=None,
                                                                chooser=None)
```

start()

lib.sedna.service.server.base

Module Contents

Classes

Server

BaseServer

```
class lib.sedna.service.server.base.Server
```

Bases: *uvicorn.Server*

install_signal_handlers()

run_in_thread()

```
class lib.sedna.service.server.base.BaseServer(servername: str, host: str = "", http_port: int = 8080,
                                                  grpc_port: int = 8081, workers: int = 1, ws_size: int =
                                                  16 * 1024 * 1024, ssl_key=None, ssl_cert=None,
                                                  timeout=300)
```

DEBUG = True

WAIT_TIME = 15

```
run(app, **kwargs)

wait_stop(current)
    wait the stop flag to shutdown the server

get_all_urls()
```

`lib.sedna.service.server.inference`

Module Contents

Classes

<i>InferenceServer</i>	rest api server for inference
------------------------	-------------------------------

```
class lib.sedna.service.server.inference.InferenceServer(model, servername, host: str = '127.0.0.1',
                                                         http_port: int = 8080, max_buffer_size:
                                                         int = 104857600, workers: int = 1)
```

Bases: `lib.sedna.service.server.base.BaseServer`

rest api server for inference

start()

model_info()

predict(data: *InferenceItem*)

Package Contents

Classes

<i>InferenceServer</i>	rest api server for inference
<i>AggregationServer</i>	

<i>AggregationServerV2</i>	
----------------------------	--

```
class lib.sedna.service.server.InferenceServer(model, servername, host: str = '127.0.0.1', http_port:
                                                         int = 8080, max_buffer_size: int = 104857600,
                                                         workers: int = 1)
```

Bases: `lib.sedna.service.server.base.BaseServer`

rest api server for inference

start()

model_info()

predict(data: *InferenceItem*)

```

class lib.sedna.service.server.AggregationServer(
    aggregation: str, host: str = None, http_port: int =
        None, exit_round: int = 1, participants_count: int =
        1, ws_size: int = 10 * 1024 * 1024)

    Bases: lib.sedna.service.server.base.BaseServer

    start()
        Start the server

    async client_info(request: starlette.requests.Request)

class lib.sedna.service.server.AggregationServerV2(
    data=None, estimator=None, aggregation=None,
    transmitter=None, chooser=None)

    start()

```

Submodules

`lib.sedna.service.client`

Module Contents

Classes

<code>LCReporter</code>	Inherited thread, which is an entity that periodically report to
<code>LCClient</code>	send info to LC by http
<code>AggregationClient</code>	Client that interacts with the cloud aggregator.
<code>ModelClient</code>	Remote model service
<code>KBCClient</code>	Communicate with Knowledge Base server

Functions

<code>http_request</code>	<code>(url[, method, timeout, binary, no_decode])</code>
---------------------------	--

```

lib.sedna.service.client.http_request(url, method=None, timeout=None, binary=True,
    no_decode=False, **kwargs)

```

```

class lib.sedna.service.client.LCReporter(lc_server, message, period_interval=30)

    Bases: threading.Thread

    Inherited thread, which is an entity that periodically report to the lc.

    update_for_edge_inference()

    update_for_collaboration_inference()

```

run()

Method representing the thread's activity.

You may override this method in a subclass. The standard `run()` method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the `args` and `kwargs` arguments, respectively.

class lib.sedna.service.client.LCClient

send info to LC by http

classmethod send(*lc_server, worker_name, message: dict*)**class lib.sedna.service.client.AggregationClient**(*url, client_id, **kwargs*)

Client that interacts with the cloud aggregator.

max_size**async connect()****send**(*data, msg_type='message', job_name=''*)**recv**(*wait_data_type=None*)**class lib.sedna.service.client.ModelClient**(*service_name, version='', host='127.0.0.1', port='8080', protocol='http'*)

Remote model service

check_server_status()**inference**(*x, **kwargs*)

Use the remote big model server to inference.

class lib.sedna.service.client.KBClient(*kbserver*)

Communicate with Knowledge Base server

upload_file(*files, name=''*)**update_db**(*task_info_file*)**update_task_status**(*tasks: str, new_status=1*)**lib.sedna.service.run_kb**

Module Contents

Functions

main()

lib.sedna.service.run_kb.main()

19.2 Submodules

19.2.1 `lib.sedna.__version__`

sedna version information.

Module Contents

`lib.sedna.__version__.tmp`

This document helps you prepare environment for developing code for Sedna. If you follow this guide and find some problem, please fill an issue to update this file.

1. INSTALL TOOLS

20.1 Install Git

Sedna is managed with [git](#), and to develop locally you will need to install [git](#).

You can check if [git](#) is already on your system and properly installed with the following command:

```
git --version
```

20.2 Install Go(optional)

All Sedna's control components(i.e. [GM/LC](#)) are written in the [Go](#). If you are planning to change them, you need to set up [Go](#).

Sedna currently builds with Go 1.16, install or upgrade [Go](#) using [the instructions for your operating system](#).

You can check if [Go](#) is in your system with the following command:

```
go version
```


2. CLONE THE CODE

Clone the Sedna repo:

```
git clone http://github.com/kubeedge/sedna.git
```

Note: If you want to add or change API in `pkg/apis`, you need to checkout the code to `$GOPATH/src/github.com/kubeedge/sedna`.

3. SET UP KUBERNETES/KUBEEDGE(OPTIONAL)

If you are planning to run or debug Sedna, you need to set up Kubernetes and KubeEdge.

Sedna requires Kubernetes version 1.16 or higher with CRD support.

Sedna requires KubeEdge version 1.5 or higher with edge support.

Note: You need to check [the Kubernetes compatibility of KubeEdge](#).

22.1 Install Kubernetes

Follow [Kubernetes setup guides](#) to set up and run Kubernetes, like:

If you're learning Kubernetes, use the [tools](#) to set up a Kubernetes cluster on a local machine, e.g.:

- [Installing Kubernetes with Kind](#)
- [Installing Kubernetes with Minikube](#)

22.2 Install KubeEdge

Please follow [the kubeedge instructions](#) to install KubeEdge.

4. WHAT'S NEXT?

Once you've set up the prerequisites, continue with:

- See [control plane development guide](#) for more details about how to build & test Sedna.
- See [lib development guide](#) for more details about how to develop AI algorithms and worker images based on [sedna lib code](#).

ROADMAP

This document defines a high level roadmap for sedna development.
The [milestones defined in GitHub](#) represent the most up-to-date plans.

24.1 2022 Roadmap

- Integrate some common multi-task migration algorithms to resolve the problem of low precision caused by small size samples.
- Integrate KubeFlow and ONNX into Sedna, to enable interoperability of edge models with diverse formats.
- Integrate typical AI frameworks into Sedna, include Tensorflow, Pytorch, PaddlePaddle and Mindspore etc.
- Support edge model and dataset management.

RELATED LINKS

25.1 Release

[Sedna0.4.0](#) [KubeEdgeSedna 0.3.0 AI](#) [KubeEdgeSedna 0.1 AI](#) [KubeEdgeSedna](#) [KubeEdge](#) [AI SIG](#)

25.2 Meetup and Conference

[HDC.Cloud 2021AI](#) [KubeEdge](#) [SednaAI50%](#)

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

|
 lib.sedna, 123
 lib.sedna.__version__, 229
 lib.sedna.algorithms, 123
 lib.sedna.algorithms.aggregation, 123
 lib.sedna.algorithms.aggregation.aggregation, 123
 lib.sedna.algorithms.client_choose, 125
 lib.sedna.algorithms.client_choose.client_choose, 125
 lib.sedna.algorithms.hard_example_mining, 126
 lib.sedna.algorithms.hard_example_mining.hard_example_mining, 126
 lib.sedna.algorithms.multi_task_learning, 128
 lib.sedna.algorithms.multi_task_learning.multi_task_learning, 132
 lib.sedna.algorithms.multi_task_learning.task_jobs, 128
 lib.sedna.algorithms.multi_task_learning.task_jobs.artifact, 128
 lib.sedna.algorithms.multi_task_learning.task_jobs.inference_integrate, 128
 lib.sedna.algorithms.multi_task_learning.task_jobs.task_definition, 129
 lib.sedna.algorithms.multi_task_learning.task_jobs.task_mining, 130
 lib.sedna.algorithms.multi_task_learning.task_jobs.task_relation_discover, 130
 lib.sedna.algorithms.multi_task_learning.task_jobs.task_remodeling, 131
 lib.sedna.algorithms.optical_flow, 137
 lib.sedna.algorithms.reid, 137
 lib.sedna.algorithms.reid.close_contact_estimation, 137
 lib.sedna.algorithms.reid.multi_img_matching, 138
 lib.sedna.algorithms.seen_task_learning, 139
 lib.sedna.algorithms.seen_task_learning.artifact, 148
 lib.sedna.algorithms.seen_task_learning.inference_integration, 139
 lib.sedna.algorithms.seen_task_learning.inference_integration.base_inference_integrate, 139
 lib.sedna.algorithms.seen_task_learning.inference_integrate, 139
 lib.sedna.algorithms.seen_task_learning.seen_task_learning, 148
 lib.sedna.algorithms.seen_task_learning.task_allocation, 140
 lib.sedna.algorithms.seen_task_learning.task_allocation.base_inference_integrate, 140
 lib.sedna.algorithms.seen_task_learning.task_allocation.task_allocation, 141
 lib.sedna.algorithms.seen_task_learning.task_allocation.task_allocation, 142
 lib.sedna.algorithms.seen_task_learning.task_allocation.task_allocation, 142
 lib.sedna.algorithms.seen_task_learning.task_definition, 143
 lib.sedna.algorithms.seen_task_learning.task_definition.base_inference_integrate, 143
 lib.sedna.algorithms.seen_task_learning.task_definition.task_definition, 143
 lib.sedna.algorithms.seen_task_learning.task_definition.task_definition, 144
 lib.sedna.algorithms.seen_task_learning.task_relation_discover, 145
 lib.sedna.algorithms.seen_task_learning.task_relation_discover, 145
 lib.sedna.algorithms.seen_task_learning.task_remodeling, 146
 lib.sedna.algorithms.seen_task_learning.task_remodeling.base_inference_integrate, 146
 lib.sedna.algorithms.seen_task_learning.task_remodeling.task_remodeling, 146
 lib.sedna.algorithms.seen_task_learning.task_update_decision, 147
 lib.sedna.algorithms.seen_task_learning.task_update_decision, 147
 lib.sedna.algorithms.seen_task_learning.task_update_decision, 147
 lib.sedna.algorithms.seen_task_learning.task_update_decision, 148
 lib.sedna.algorithms.transmitter, 154

lib.sedna.algorithms.transmitter.transmitter,	lib.sedna.core.lifelong_learning,	188
154	lib.sedna.core.lifelong_learning.knowledge_management,	
lib.sedna.algorithms.unseen_task_detect,	188	
lib.sedna.algorithms.unseen_task_detect.unseen_task_detector,	lib.sedna.core.lifelong_learning.knowledge_management.base	
155	188	
lib.sedna.algorithms.unseen_task_detection,	lib.sedna.core.lifelong_learning.knowledge_management.cloud	
157	189	
lib.sedna.algorithms.unseen_task_detection.unseen_sample_recognition,	lib.sedna.core.lifelong_learning.knowledge_management.edge	
157	190	
lib.sedna.algorithms.unseen_task_detection.unseen_sample_recognition.based_on_unseen_sample_learning,	lib.sedna.core.lifelong_learning.knowledge_management.unseen_sample_recognition,	
157	192	
lib.sedna.algorithms.unseen_task_detection.unseen_sample_recognition.inferencer,	lib.sedna.core.multi_edge_inference.sample_re_recognition,	
158	lib.sedna.core.multi_edge_inference.components,	
lib.sedna.algorithms.unseen_task_detection.unseen_sample_recognition,	198	
159	lib.sedna.core.multi_edge_inference.components.detector,	
lib.sedna.algorithms.unseen_task_detection.unseen_sample_recognition.base_unseen_sample_recognition,	lib.sedna.core.multi_edge_inference.components.feature_ext	
159	lib.sedna.core.multi_edge_inference.components.reid,	
lib.sedna.algorithms.unseen_task_detection.unseen_sample_recognition.unseen_sample_detection,	lib.sedna.core.multi_edge_inference.components.reid,	
160	lib.sedna.core.multi_edge_inference.data_classes,	
lib.sedna.algorithms.unseen_task_detection.unseen_sample_recognition.unseen_sample_recognition,	207	
160	lib.sedna.core.multi_edge_inference.plugins,	
lib.sedna.algorithms.unseen_task_processing,	lib.sedna.core.multi_edge_inference.plugins.registered,	
161	204	
lib.sedna.algorithms.unseen_task_processing.unseen_task_allocation,	lib.sedna.core.multi_edge_inference.plugins.registered,	
161	lib.sedna.core.multi_edge_inference.utils,	
lib.sedna.algorithms.unseen_task_processing.unseen_task_allocation.base_unseen_task_allocation,	lib.sedna.core.multi_edge_inference.utils,	
161	lib.sedna.core.multi_edge_inference.unseen_task_allocation,	
lib.sedna.algorithms.unseen_task_processing.unseen_task_allocation.unseen_task_allocation,	lib.sedna.datasources,	210
162	lib.sedna.datasources.kafka,	210
lib.sedna.algorithms.unseen_task_processing.unseen_task_allocation.unseen_task_allocation.kafka,	lib.sedna.datasources.kafka.consumer,	210
162	lib.sedna.datasources.kafka.kafka_manager,	
lib.sedna.backend,	210	
lib.sedna.backend.base,	lib.sedna.datasources.kafka.producer,	211
lib.sedna.backend.mindspore,	lib.sedna.datasources.obs,	212
lib.sedna.backend.tensorflow,	lib.sedna.datasources.obs.connector,	212
lib.sedna.backend.torch,	lib.sedna.service,	214
lib.sedna.common,	lib.sedna.service.client,	227
lib.sedna.common.class_factory,	lib.sedna.service.multi_edge_inference,	214
lib.sedna.common.config,	lib.sedna.service.multi_edge_inference.interface,	
lib.sedna.common.constant,	214	
lib.sedna.common.file_ops,	lib.sedna.service.multi_edge_inference.interface.detection	
lib.sedna.common.log,	214	
lib.sedna.common.utils,	lib.sedna.service.multi_edge_inference.interface.fe_endpoi	
lib.sedna.core,	214	
lib.sedna.core.base,	lib.sedna.service.multi_edge_inference.interface.reid_endp	
lib.sedna.core.federated_learning,	lib.sedna.service.multi_edge_inference.server,	
lib.sedna.core.federated_learning.federated_learning,	lib.sedna.service.multi_edge_inference.server.detection,	
176	216	
lib.sedna.core.incremental_learning,	lib.sedna.service.multi_edge_inference.server.feature_ext	
lib.sedna.core.incremental_learning.incremental_learning,	217	
179	lib.sedna.service.multi_edge_inference.server.reid,	
lib.sedna.core.joint_inference,	184	
lib.sedna.core.joint_inference.joint_inference,		
184		

218

`lib.sedna.service.run_kb`, [228](#)
`lib.sedna.service.server`, [221](#)
`lib.sedna.service.server.aggregation`, [224](#)
`lib.sedna.service.server.base`, [225](#)
`lib.sedna.service.server.inference`, [226](#)
`lib.sedna.service.server.knowledgeBase`, [221](#)
`lib.sedna.service.server.knowledgeBase.database`,
 [221](#)
`lib.sedna.service.server.knowledgeBase.model`,
 [221](#)
`lib.sedna.service.server.knowledgeBase.server`,
 [224](#)

INDEX

Symbols

Symbols

--call__()	(lib.sedna.algorithms.hard_example_mining.hard_example_minning.CrossEntropyFilter method), 127
--call__()	(lib.sedna.algorithms.hard_example_mining.hard_example_minning.IDF filter method), 127
--call__()	(lib.sedna.algorithms.hard_example_mining.hard_example_minning.ThresholdFilter method), 126
--call__()	(lib.sedna.algorithms.multi_task_learning.task_jobs.inference_integrate.DefaultInferenceIntegr rule method), 128
--call__()	(lib.sedna.algorithms.multi_task_learning.task_jobs.task_definition.TasksDefinitionByDataAttr method), 129
--call__()	(lib.sedna.algorithms.multi_task_learning.task_jobs.task_definition.TasksDefinitionBySVC method), 129
--call__()	(lib.sedna.algorithms.multi_task_learning.task_jobs.task_minling.TaskMinlingByDataAttr method), 130
--call__()	(lib.sedna.algorithms.multi_task_learning.task_jobs.task_minling.TaskMinlingBySVC method), 130
--call__()	(lib.sedna.algorithms.multi_task_learning.task_jobs.task_relation_discover.DefaultTaskRelationDiscover method), 131
--call__()	(lib.sedna.algorithms.multi_task_learning.task_jobs.task_remodeling.DefaultTaskRemodeling method), 131
--call__()	(lib.sedna.algorithms.seen_task_learning.inference_integration.BaseInferenceIntegr rule method), 139
--call__()	(lib.sedna.algorithms.seen_task_learning.inference_integration.infernce_integrate.DefaultInferenceIntegr rule method), 140
--call__()	(lib.sedna.algorithms.seen_task_learning.task_allocation.base_task_allocation.BaseTaskAllocation method), 141
--call__()	(lib.sedna.algorithms.seen_task_learning.task_allocation.task_allocator.TaskAllocatorByDataAttr method), 142
--call__()	(lib.sedna.algorithms.seen_task_learning.task_allocation.task_allocator.TaskAllocatorBySVC method), 141
--call__()	(lib.sedna.algorithms.seen_task_learning.task_allocation.task_allocator.TaskAllocationDefault method), 142
--call__()	(lib.sedna.algorithms.seen_task_learning.task_allocation_by_origin.TaskAllocationByOrigin method), 142
--call__()	(lib.sedna.common.class_factory.ClassFactory attribute), 169
--call__()	(lib.sedna.service.server_knowledgeBase.model.Tasks definition method), 222
--call__()	(lib.sedna.service.server_knowledgeBase.model.Samples definition attribute), 223
--call__()	(lib.sedna.service.server_knowledgeBase.model.TaskGrp definition attribute)

attribute), 222
 __tablename__ (lib.sedna.service.server.knowledgeBase.model.TaskDefinition attribute), 222
 __tablename__ (lib.sedna.service.server.knowledgeBase.model.TaskSimple attribute), 223
 __tablename__ (lib.sedna.service.server.knowledgeBase.model.TaskComplex attribute), 223
 __tablename__ (lib.sedna.service.server.knowledgeBase.model.TaskComplex attribute), 222
A
 AbstractClientChoose (class in lib.sedna.algorithms.client_choose.client_choose), 125
 AbstractTransmitter (class in lib.sedna.algorithms.transmitter.transmitter), 154
 access_key_id (lib.sedna.common.config.BaseConfig attribute), 171
 AdminClient (class in lib.sedna.datasources.kafka), 212
 agg_data_path (lib.sedna.common.config.BaseConfig attribute), 171
 AggClient (class in lib.sedna.algorithms.aggregation), 124
 AggClient (class in lib.sedna.algorithms.aggregation.aggregation), 123
 aggregate() (lib.sedna.algorithms.aggregation.aggregation.FedAvg method), 123
 aggregate() (lib.sedna.algorithms.aggregation.FedAvg method), 124
 aggregate() (lib.sedna.algorithms.aggregation.FedAvgV2 method), 125
 aggregate() (lib.sedna.algorithms.aggregation.MistNet method), 124
 AggregationClient (class in lib.sedna.service.client), 228
 AggregationServer (class in lib.sedna.service.server), 226
 AggregationServer (class in lib.sedna.service.server.aggregation), 225
 AggregationServerV2 (class in lib.sedna.service.server), 227
 AggregationServerV2 (class in lib.sedna.service.server.aggregation), 225
 ALGORITHM (lib.sedna.common.class_factory.ClassType attribute), 169
B
 backend_type (lib.sedna.common.config.BaseConfig attribute), 170
 BackendBase (class in lib.sedna.backend.base), 167
 Base (in module lib.sedna.service.server.knowledgeBase.database), 221
 BaseConfig (class in lib.sedna.common.config), 170
 BaseDataSource (class in lib.sedna.datasources), 213
 BaseInferenceIntegrate (class in lib.sedna.algorithms.seen_task_learning.inference_integration.base_inference_integrate), 213
 BaseKnowledgeManagement (class in lib.sedna.core.lifelong_learning.knowledge_management), 190
 BaseKnowledgeManagement (class in lib.sedna.core.lifelong_learning.knowledge_management.base_knowledge_management), 189
 BaseSampleRegonition (class in lib.sedna.algorithms.unseen_task_detection.unseen_sample_recognition), 159
 BaseSampleReRegonition (class in lib.sedna.algorithms.unseen_task_detection.unseen_sample_recognition), 158
 BaseServer (class in lib.sedna.service.server.base), 225
 BaseService (class in lib.sedna.core.multi_edge_inference), 208
 BaseService (class in lib.sedna.core.multi_edge_inference.components), 203
 BaseTaskAllocation (class in lib.sedna.algorithms.seen_task_learning.task_allocation.base_task_allocation), 141
 BaseTaskDefinition (class in lib.sedna.algorithms.seen_task_learning.task_definition.base_task_definition), 143
 BaseTaskRelationDiscover (class in lib.sedna.algorithms.seen_task_learning.task_relation_discovery), 145
 BaseTaskRemodeling (class in lib.sedna.algorithms.seen_task_learning.task_remodeling.base_task_remodeling), 146
 BaseTaskUpdateDecision (class in lib.sedna.algorithms.seen_task_learning.task_update_decision.base_task_update_decision), 147
 BaseUnseenTaskAllocation (class in lib.sedna.algorithms.unseen_task_processing.unseen_task_allocation), 161
 BigModelService (class in lib.sedna.core.joint_inference), 188
 BigModelService (class in lib.sedna.core.joint_inference.joint_inference), 184
C
 CALLBACK (lib.sedna.common.class_factory.ClassType attribute), 169
 check_bbox_overlap() (lib.sedna.algorithms.reid.close_contact_estimation.ContactTracker method), 138
 check_server_status() (lib.sedna.service.client.ModelClient method), 184

228
 check_server_status() (lib.sedna.service.multi_edge_inference.DetectionConfigProto (in module lib.sedna.backend.tensorflow), method), 219
 check_server_status() (lib.sedna.service.multi_edge_inference.FE method), 219
 check_server_status() (lib.sedna.service.multi_edge_inference.interface.Detection method), 215
 check_server_status() (lib.sedna.service.multi_edge_inference.interface.detection method), 214
 check_server_status() (lib.sedna.service.multi_edge_inference.interface.FE method), 216
 check_server_status() (lib.sedna.service.multi_edge_inference.interface.Consumer method), 215
 check_server_status() (lib.sedna.service.multi_edge_inference.interface.ReID_Endpoint method), 216
 check_server_status() (lib.sedna.service.multi_edge_inference.interface.ReID_Endpoint method), 215
 check_server_status() (lib.sedna.service.multi_edge_inference.ReID_Endpoint class method), 220
 ClassFactory (class in lib.sedna.common.class_factory), 169
 ClassType (class in lib.sedna.common.class_factory), 169
 clean_folder() (lib.sedna.common.file_ops.FileOps class method), 173
 Client (class in lib.sedna.datasources.kafka), 212
 client_info() (lib.sedna.service.server.aggregation.AggregationServer method), 225
 client_info() (lib.sedna.service.server.AggregationServer method), 227
 close() (lib.sedna.core.multi_edge_inference.components.detector.ObjectData method), 200
 close() (lib.sedna.datasources.kafka.consumer.Consumer method), 210
 close() (lib.sedna.datasources.kafka.producer.Producer method), 211
 CloudKnowledgeManagement (class in lib.sedna.core.lifelong_learning.knowledge_management), 191
 CloudKnowledgeManagement (class in lib.sedna.core.lifelong_learning.knowledge_management.cloud_knowledge_management), 189
 COMPLETED (lib.sedna.common.constant.K8sResourceKindStatus attribute), 172
 compute_homography() (lib.sedna.algorithms.reid.close_contact_estimation.ContactTracker method), 137
 connect() (lib.sedna.datasources.kafka.Client method), 212
 connect() (lib.sedna.datasources.kafka.consumer.Consumer method), 210
 connect() (lib.sedna.datasources.kafka.producer.Producer method), 211
 connect() (lib.sedna.service.client.AggregationClient method), 212
 consume_messages() (lib.sedna.datasources.kafka.consumer.Consumer method), 210
 consume_messages_poll() (lib.sedna.datasources.kafka.consumer.Consumer method), 210
 consume_messages_poll() (lib.sedna.datasources.kafka.consumer.Consumer method), 210
 consume_messages_poll() (lib.sedna.datasources.kafka.consumer.Consumer method), 210
 ContactTracker (class in lib.sedna.algorithms.reid.close_contact_estimation), 137
 Context (class in lib.sedna.common.config), 171
 copy_file() (lib.sedna.common.file_ops.FileOps class method), 174
 copy_folder() (lib.sedna.common.file_ops.FileOps class method), 174
 cosine_similarity_score() (in module lib.sedna.algorithms.reid.multi_img_matching), 138
 COVID19 (lib.sedna.core.multi_edge_inference.data_classes.OP_MODE attribute), 207
 create_ellipse() (lib.sedna.algorithms.reid.close_contact_estimation.ContactTracker method), 137
 create_topics() (lib.sedna.datasources.kafka.AdminClient method), 212
 created_at (lib.sedna.service.server.knowledgeBase.model.TaskModel attribute), 222
 created_at (lib.sedna.service.server.knowledgeBase.model.Tasks attribute), 222
 CrossEntropyFilter (class in lib.sedna.algorithms.hard_example_mining.hard_example_mining), 127
 CSVDataParse (class in lib.sedna.datasources), 213
 D
 data_path_prefix (lib.sedna.common.config.BaseConfig attribute), 171
 data_type (lib.sedna.service.server.knowledgeBase.model.Samples attribute), 223
 data_url (lib.sedna.service.server.knowledgeBase.model.Samples attribute), 223
 DATASET (lib.sedna.common.class_factory.ClassType attribute), 169

DEBUG (*lib.sedna.service.server.base.BaseServer* attribute), 225
DEFAULT (*lib.sedna.common.constant.K8sResourceKind* attribute), 172
DefaultInferenceIntegrate (class in *lib.sedna.algorithms.multi_task_learning.task_jobs.inference.integrate*), 128
DefaultInferenceIntegrate (class in *lib.sedna.algorithms.seen_task_learning.inference.integrate*), 140
DefaultTaskRelationDiscover (class in *lib.sedna.algorithms.multi_task_learning.task_jobs.discovery.task_relation_discovery*), 131
DefaultTaskRelationDiscover (class in *lib.sedna.algorithms.seen_task_learning.task_relation_discovery.task_relation_discovery*), 145
DefaultTaskRemodeling (class in *lib.sedna.algorithms.multi_task_learning.task_jobs.task_remodeling*), 131
DefaultTaskRemodeling (class in *lib.sedna.algorithms.seen_task_learning.task_remodeling.task_remodeling*), 147
delete() (*lib.sedna.common.file_ops.FileOps* class method), 173
delete_from_disk() (*lib.sedna.core.multi_edge_inference.components.FileOperations* method), 204
delete_from_disk() (*lib.sedna.core.multi_edge_inference.FileOperations* method), 209
delete_topics() (*lib.sedna.datasources.kafka.AdminClient* method), 212
deploy (*lib.sedna.service.server.knowledgeBase.model.TaskGrp* attribute), 222
descr (*lib.sedna.service.server.knowledgeBase.model.Samples* attribute), 223
Detection (class in *lib.sedna.service.multi_edge_inference*), 219
Detection (class in *lib.sedna.service.multi_edge_inference.interface*), 215
Detection (class in *lib.sedna.service.multi_edge_inference.interface.detection_endpoint*), 214
DETECTION (*lib.sedna.core.multi_edge_inference.data_classes.OP_MODE* attribute), 207
DetectionServer (class in *lib.sedna.service.multi_edge_inference*), 220
DetectionServer (class in *lib.sedna.service.multi_edge_inference.server*), 218
DetectionServer (class in *lib.sedna.service.multi_edge_inference.server.detection*), 216
DetTrackResult (class in *lib.sedna.core.multi_edge_inference.data_classes*), 207
device_category (*lib.sedna.common.config.BaseConfig* attribute), 170
distribute_data() (*lib.sedna.core.multi_edge_inference.BaseService* method), 208
distribute_data() (*lib.sedna.core.multi_edge_inference.components.BaseService* method), 203
download() (*lib.sedna.common.file_ops.FileOps* class method), 174
download_single_object() (*lib.sedna.datasources.obs.connector.OBSClientWrapper* method), 212
dump() (*lib.sedna.common.file_ops.FileOps* class method), 174
dump_pickle() (*lib.sedna.common.file_ops.FileOps* class method), 174
easy_installation (E), 172
EdgeKnowledgeManagement (class in *lib.sedna.core.lifelong_learning.knowledge_management*), 191
EdgeKnowledgeManagement (class in *lib.sedna.core.lifelong_learning.knowledge_management.edge_knowledge_management*), 191
engine (in module *lib.sedna.service.server.knowledgeBase.database*), 222
engine (in module *lib.sedna.service.server.knowledgeBase.model*), 222
evaluate() (*lib.sedna.algorithms.multi_task_learning.MulTaskLearning* method), 137
evaluate() (*lib.sedna.algorithms.multi_task_learning.multi_task_learning* method), 134
evaluate() (*lib.sedna.algorithms.seen_task_learning.seen_task_learning* method), 151
evaluate() (*lib.sedna.algorithms.seen_task_learning.SeenTaskLearning* method), 153
evaluate() (*lib.sedna.backend.base.BackendBase* method), 165
evaluate() (*lib.sedna.backend.mindspore.MSBackend* method), 166
evaluate() (*lib.sedna.backend.tensorflow.TFBackend* method), 167
evaluate() (*lib.sedna.backend.torch.TorchBackend* method), 167
evaluate() (*lib.sedna.core.base.JobBase* method), 209
evaluate() (*lib.sedna.core.incremental_learning.incremental_learning* method), 181
evaluate() (*lib.sedna.core.incremental_learning.IncrementalLearning* method), 183
evaluate() (*lib.sedna.core.lifelong_learning.lifelong_learning.LifelongLearning* method), 194
evaluate() (*lib.sedna.core.lifelong_learning.LifelongLearning* method), 198

[evaluate\(\)](#) (`lib.sedna.core.multi_edge_inference.plugins.PluggableModel`
`method`), 206
[evaluate_tasks\(\)](#) (`lib.sedna.core.lifelong_learning.knowledge_management.CloudKnowledgeManager`
`method`), 189
[evaluate_tasks\(\)](#) (`lib.sedna.core.lifelong_learning.knowledge_management.FederatedLearningV2`
`method`), 191
[exists\(\)](#) (`lib.sedna.common.file_ops.FileOps` `class` `FEServer` (`class` in `lib.sedna.service.multi_edge_inference`),
`method`), 175
[EXTRACTOR](#) (`lib.sedna.common.constant.KBResourceConstant` `attribute`), 172
F
[FAILED](#) (`lib.sedna.common.constant.K8sResourceKindStatus` `attribute`), 172
[FE](#) (`class` in `lib.sedna.service.multi_edge_inference`), 219
[FE](#) (`class` in `lib.sedna.service.multi_edge_inference.interface`), 216
[FE](#) (`class` in `lib.sedna.service.multi_edge_inference.interface.fe_endpoint`), 214
[FeatureExtraction](#) (`class` in `lib.sedna.core.multi_edge_inference.plugins.registered`), 204
[FEATURE_EXTRACTION](#) (`lib.sedna.core.multi_edge_inference.plugins.registered` `attribute`), 206
[feature_extraction\(\)](#) (`lib.sedna.service.multi_edge_inference.FEServer` `method`), 220
[feature_extraction\(\)](#) (`lib.sedna.service.multi_edge_inference.server.feature_extraction.FEServer` `method`), 217
[feature_extraction\(\)](#) (`lib.sedna.service.multi_edge_inference.server.FEServer` `method`), 219
[FeatureExtraction_I](#) (`class` in `lib.sedna.core.multi_edge_inference.plugins.registered`), 205
[FEATURE_EXTRACTION_I](#) (`lib.sedna.core.multi_edge_inference.plugins.PLUGIN` `attribute`), 206
[FEATURE_EXTRACTION_SERVICE](#) (`lib.sedna.common.constant.K8sResourceKind` `attribute`), 172
[FedAvg](#) (`class` in `lib.sedna.algorithms.aggregation`), 124
[FedAvg](#) (`class` in `lib.sedna.algorithms.aggregation.aggregation`), 123
[FedAvgV2](#) (`class` in `lib.sedna.algorithms.aggregation`), 125
[FEDERATED_LEARNING_JOB](#) (`lib.sedna.common.constant.K8sResourceKind` `attribute`), 172
[FederatedLearning](#) (`class` in `lib.sedna.core.federated_learning`), 178
[FederatedLearning](#) (`class` in `lib.sedna.core.federated_learning.federated_learning`), 178
[FederatedLearningV2](#) (`class` in `lib.sedna.core.federated_learning.federated_learning`), 178
[FEServer](#) (`class` in `lib.sedna.service.multi_edge_inference.server`), 218
[FEServer](#) (`class` in `lib.sedna.service.multi_edge_inference.server.feature_extraction`), 217
[FEService](#) (`class` in `lib.sedna.core.multi_edge_inference.components.feature_extraction`), 200
[fetch_data\(\)](#) (`lib.sedna.core.multi_edge_inference.BaseService` `method`), 208
[fetch_data\(\)](#) (`lib.sedna.core.multi_edge_inference.components.BaseService` `method`), 203
[file_download\(\)](#) (`lib.sedna.service.server.knowledgeBase.server.KBServer` `method`), 224
[file_upload\(\)](#) (`lib.sedna.service.server.knowledgeBase.server.KBServer` `method`), 224
[FileOperations](#) (`class` in `lib.sedna.core.multi_edge_inference.components`), 209
[FileOperations](#) (`class` in `lib.sedna.core.multi_edge_inference.components`), 203
[FileOps](#) (`class` in `lib.sedna.common.file_ops`), 173
[finetune\(\)](#) (`lib.sedna.backend.mindspore.MSBackend` `method`), 165
[finetune\(\)](#) (`lib.sedna.backend.tensorflow.KerasBackend` `method`), 167
[finetune\(\)](#) (`lib.sedna.backend.tensorflow.TFBackend` `method`), 166
[FL_AGG](#) (`lib.sedna.common.class_factory.ClassType` `attribute`), 169
[flatten\(\)](#) (`lib.sedna.core.multi_edge_inference.BaseService` `method`), 208
[flatten\(\)](#) (`lib.sedna.core.multi_edge_inference.components.BaseService` `method`), 203
G
[gcs_download\(\)](#) (`lib.sedna.common.file_ops.FileOps` `class` `method`), 174
[gcs_upload\(\)](#) (`lib.sedna.common.file_ops.FileOps` `class` `method`), 174
[GENERAL](#) (`lib.sedna.common.class_factory.ClassType` `attribute`), 169
[get_algorithm_from_api\(\)](#) (`lib.sedna.common.config.Context` `class` `method`), 171
[get_all_urls\(\)](#) (`lib.sedna.service.server.base.BaseServer` `method`), 226

`get_cls()` (`lib.sedna.common.class_factory.ClassFactory` class method), 170
`get_envron_varia()` (`lib.sedna.algorithms.unseen_task_detection.unseen_sample_detection.UnseenSampleDetection` method), 160
`get_file_hash()` (`lib.sedna.common.file_ops.FileOps` class method), 173
`get_files_list()` (`lib.sedna.core.multi_edge_inference.components.feature_extraction.FEService` method), 204
`get_files_list()` (`lib.sedna.core.multi_edge_inference.FileOperations` method), 209
`get_hem_algorithm_from_config()` (`lib.sedna.core.incremental_learning.incremental_learning.IncrementalLearning` class method), 180
`get_hem_algorithm_from_config()` (`lib.sedna.core.incremental_learning.IncrementalLearning` class method), 182
`get_hem_algorithm_from_config()` (`lib.sedna.core.joint_inference.joint_inference.JointInference` class method), 185
`get_hem_algorithm_from_config()` (`lib.sedna.core.joint_inference.JointInference` class method), 187
`get_homography_matrix()` (`lib.sedna.algorithms.reid.close_contact_estimation.ContactEstimation` method), 138
`get_host_ip()` (in module `lib.sedna.common.utils`), 176
`get_index()` (`lib.sedna.algorithms.unseen_task_detection.unseen_sample_detection.UnseenSampleDetection` method), 160
`get_or_create()` (in module `lib.sedna.service.server.knowledgeBase.model`), 223
`get_parameters()` (in module `lib.sedna.core.multi_edge_inference.utils`), 208
`get_parameters()` (`lib.sedna.common.config.Context` class method), 171
`get_parameters()` (`lib.sedna.core.base.JobBase` method), 209
`get_plugin()` (`lib.sedna.core.multi_edge_inference.BaseService` method), 208
`get_plugin()` (`lib.sedna.core.multi_edge_inference.components.BaseService` method), 203
`get_target_features()` (`lib.sedna.core.multi_edge_inference.components.feature_extraction.FEService` method), 201
`get_target_features()` (`lib.sedna.core.multi_edge_inference.components.reid.ReID` method), 202
`get_target_features()` (`lib.sedna.service.multi_edge_inference.FE` method), 220
`get_target_features()` (`lib.sedna.service.multi_edge_inference.FEServer` method), 220
`get_target_features()` (`lib.sedna.service.multi_edge_inference.interface.FE` method), 215
`get_target_features()` (`lib.sedna.service.multi_edge_inference.interface.fe_endpoint.FEEndpoint` method), 215
`get_target_features()` (`lib.sedna.service.multi_edge_inference.server.feature_extraction.FEServer` method), 217
`get_target_features()` (`lib.sedna.service.multi_edge_inference.server.FEServer` method), 219
`get_topics()` (`lib.sedna.datasources.kafka.consumer.Consumer` method), 210
`get_transmitter_from_config()` (`lib.sedna.core.federated_learning.federated_learning.FederatedLearning` class method), 177
`get_transmitter_from_config()` (`lib.sedna.core.federated_learning.FederatedLearningV2` class method), 179
`get_weights()` (`lib.sedna.backend.base.BackendBase` method), 168
`get_weights()` (`lib.sedna.backend.mindspore.MSBackend` method), 166
`get_weights()` (`lib.sedna.backend.tensorflow.KerasBackend` method), 167
`get_weights()` (`lib.sedna.backend.tensorflow.KerasBackend` method), 167
`get_weights()` (`lib.sedna.backend.tensorflow.KerasBackend` method), 166
`grp` (`lib.sedna.service.server.knowledgeBase.model.TaskRelation` attribute), 223
`grp_id` (`lib.sedna.service.server.knowledgeBase.model.TaskRelation` attribute), 223

H

`hardened_connect()` (`lib.sedna.datasources.kafka.Client` method), 212
`HEM` (`lib.sedna.common.class_factory.ClassType` attribute), 169
`http_download()` (`lib.sedna.common.file_ops.FileOps` class method), 174
`http_request()` (in module `lib.sedna.service.client`), 227
`IBTFilter` (class in `lib.sedna.algorithms.hard_example_mining.hard_example_mining`), 127
`id` (`lib.sedna.service.server.knowledgeBase.model.Samples` attribute), 223
`id` (`lib.sedna.service.server.knowledgeBase.model.TaskGrp` attribute), 222
`id` (`lib.sedna.service.server.knowledgeBase.model.TaskModel` attribute), 222

[id \(lib.sedna.service.server.knowledgeBase.model.TaskRelation attribute\), 223](#)
[id \(lib.sedna.service.server.knowledgeBase.model.Tasks attribute\), 222](#)
[id \(lib.sedna.service.server.knowledgeBase.model.TaskSample attribute\), 223](#)
[in_risk_zone\(\) \(lib.sedna.algorithms.reid.close_contact_estimation.ContactTracker method\), 137](#)
[INCREMENTAL_JOB \(lib.sedna.common.constant.K8sResourceKind attribute\), 172](#)
[IncrementalLearning \(class in lib.sedna.core.incremental_learning\), 181](#)
[IncrementalLearning \(class in lib.sedna.core.incremental_learning.incremental_learning\), 179](#)
[inference\(\) \(lib.sedna.core.base.JobBase method\), 209](#)
[inference\(\) \(lib.sedna.core.incremental_learning.incremental_learning.IncrementalLearning method\), 180](#)
[inference\(\) \(lib.sedna.core.incremental_learning.IncrementalLearning method\), 183](#)
[inference\(\) \(lib.sedna.core.joint_inference.BigModelService method\), 188](#)
[inference\(\) \(lib.sedna.core.joint_inference.joint_inference.BigModelService method\), 184](#)
[inference\(\) \(lib.sedna.core.joint_inference.joint_inference.JointInference method\), 186](#)
[inference\(\) \(lib.sedna.core.joint_inference.JointInference method\), 187](#)
[inference\(\) \(lib.sedna.core.lifelong_learning.lifelong_learning.LifelongLearning method\), 195](#)
[inference\(\) \(lib.sedna.core.lifelong_learning.LifelongLearning method\), 198](#)
[inference\(\) \(lib.sedna.core.multi_edge_inference.plugins.PluggableModelClient method\), 206](#)
[inference\(\) \(lib.sedna.service.client.ModelClient method\), 228](#)
[InferenceServer \(class in lib.sedna.service.server\), 226](#)
[InferenceServer \(class in lib.sedna.service.server.inference\), 226](#)
[init_db\(\) \(in module lib.sedna.service.server.knowledgeBase.model\), 223](#)
[install_signal_handlers\(\) \(lib.sedna.service.server.base.Server method\), 225](#)
[is_current \(lib.sedna.service.server.knowledgeBase.model.TaskModel attribute\), 222](#)
[is_exists\(\) \(lib.sedna.common.class_factory.ClassFactory class method\), 170](#)
[is_local\(\) \(lib.sedna.common.file_ops.FileOps class method\), 174](#)
[is_remote\(\) \(lib.sedna.common.file_ops.FileOps class method\), 174](#)
[is_test_data \(lib.sedna.datasources.BaseDataSource property\), 213](#)
[job_name \(lib.sedna.common.config.BaseConfig attribute\), 170](#)
[JobBase \(class in lib.sedna.core.base\), 209](#)
[join_path\(\) \(lib.sedna.common.file_ops.FileOps class method\), 173](#)
[JOINT_INFERENCE_SERVICE \(lib.sedna.common.constant.K8sResourceKind attribute\), 172](#)
[JointInference \(class in lib.sedna.core.joint_inference\), 186](#)
[JointInference \(class in lib.sedna.core.joint_inference.joint_inference\), 186](#)
[JSONDataParse \(class in lib.sedna.datasources\), 213](#)
K
[K8sResourceKind \(class in lib.sedna.common.constant\), 171](#)
[K8sResourceKindStatus \(class in lib.sedna.common.constant\), 172](#)
[KafkaConsumerThread \(class in lib.sedna.datasources.kafka.kafka_manager\), 211](#)
[KafkaProducer \(class in lib.sedna.datasources.kafka.kafka_manager\), 210](#)
[KB_INDEX_NAME \(lib.sedna.common.constant.KBResourceConstant attribute\), 172](#)
[KBClient \(class in lib.sedna.service.client\), 228](#)
[KBResourceConstant \(class in lib.sedna.common.constant\), 172](#)
[KBServer \(class in lib.sedna.service.server.knowledgeBase.server\), 224](#)
[KBUpdateResult \(class in lib.sedna.service.server.knowledgeBase.server\), 224](#)
[KerasBackend \(class in lib.sedna.backend.tensorflow\), 167](#)
[KM \(lib.sedna.common.class_factory.ClassType attribute\), 169](#)
L
[LCClient \(class in lib.sedna.service.client\), 228](#)
[LCReporter \(class in lib.sedna.service.client\), 227](#)
[lib.sedna module, 123](#)
[lib.sedna.__version__](#)

module, 229	module, 140
lib.sedna.algorithms	lib.sedna.algorithms.seen_task_learning.task_allocation.ba
module, 123	module, 140
lib.sedna.algorithms.aggregation	lib.sedna.algorithms.seen_task_learning.task_allocation.ta
module, 123	module, 141
lib.sedna.algorithms.aggregation.aggregation	lib.sedna.algorithms.seen_task_learning.task_allocation.ta
module, 123	module, 142
lib.sedna.algorithms.client_choose	lib.sedna.algorithms.seen_task_learning.task_allocation.ta
module, 125	module, 142
lib.sedna.algorithms.client_choose.client_choose	lib.sedna.algorithms.seen_task_learning.task_definition
module, 125	module, 143
lib.sedna.algorithms.hard_example_mining	lib.sedna.algorithms.seen_task_learning.task_definition.ba
module, 126	module, 143
lib.sedna.algorithms.hard_example_mining.hard_example_mining	lib.sedna.algorithms.seen_task_learning.task_definition.ta
module, 126	module, 143
lib.sedna.algorithms.multi_task_learning	lib.sedna.algorithms.seen_task_learning.task_definition.ta
module, 128	module, 144
lib.sedna.algorithms.multi_task_learning.multitask_learning	lib.sedna.algorithms.seen_task_learning.task_relation_disc
module, 132	module, 145
lib.sedna.algorithms.multi_task_learning.task_job	lib.sedna.algorithms.seen_task_learning.task_relation_disc
module, 128	module, 145
lib.sedna.algorithms.multi_task_learning.task_job_sedna_factory	lib.sedna.algorithms.seen_task_learning.task_relation_disc
module, 128	module, 145
lib.sedna.algorithms.multi_task_learning.task_job_sedna_factory_integrate	lib.sedna.algorithms.seen_task_learning.task_remodeling
module, 128	module, 146
lib.sedna.algorithms.multi_task_learning.task_job_sedna_factory_integrate	lib.sedna.algorithms.seen_task_learning.task_remodeling.ba
module, 129	module, 146
lib.sedna.algorithms.multi_task_learning.task_job_sedna_factory_integrate	lib.sedna.algorithms.seen_task_learning.task_remodeling.ta
module, 130	module, 146
lib.sedna.algorithms.multi_task_learning.task_job_sedna_factory_integrate	lib.sedna.algorithms.seen_task_learning.task_update_decisi
module, 130	module, 147
lib.sedna.algorithms.multi_task_learning.task_job_sedna_factory_integrate	lib.sedna.algorithms.seen_task_learning.task_update_decisi
module, 131	module, 147
lib.sedna.algorithms.optical_flow	lib.sedna.algorithms.seen_task_learning.task_update_decisi
module, 137	module, 148
lib.sedna.algorithms.reid	lib.sedna.algorithms.transmitter
module, 137	module, 154
lib.sedna.algorithms.reid.close_contact_estimation	lib.sedna.algorithms.transmitter.transmitter
module, 137	module, 154
lib.sedna.algorithms.reid.multi_img_matching	lib.sedna.algorithms.unseen_task_detect
module, 138	module, 155
lib.sedna.algorithms.seen_task_learning	lib.sedna.algorithms.unseen_task_detect.unseen_task_detect
module, 139	module, 155
lib.sedna.algorithms.seen_task_learning.artifact	lib.sedna.algorithms.unseen_task_detection
module, 148	module, 157
lib.sedna.algorithms.seen_task_learning.inference	lib.sedna.algorithms.unseen_task_detection.unseen_sample_r
module, 139	module, 157
lib.sedna.algorithms.seen_task_learning.inference_base_inference_integrate	lib.sedna.algorithms.unseen_task_detection.unseen_sample_r
module, 139	module, 157
lib.sedna.algorithms.seen_task_learning.inference_base_inference_integrate	lib.sedna.algorithms.unseen_task_detection.unseen_sample_r
module, 139	module, 158
lib.sedna.algorithms.seen_task_learning.seen_task_sedna_algorithm	lib.sedna.algorithms.unseen_task_detection.unseen_sample_r
module, 148	module, 159
lib.sedna.algorithms.seen_task_learning.task_allocation	lib.sedna.algorithms.unseen_task_detection.unseen_sample_r

module, 159	module, 184
lib.sedna.algorithms.unseen_task_detection.unseen_task_detection	lib.sedna.core.cofn lifelong_unseen_sample_detection
module, 160	module, 188
lib.sedna.algorithms.unseen_task_detection.unseen_task_detection	lib.sedna.core.cofn lifelong_unseen_sample_hedge_management
module, 160	module, 188
lib.sedna.algorithms.unseen_task_processing	lib.sedna.core.lifelong_learning.knowledge_management.base
module, 161	module, 188
lib.sedna.algorithms.unseen_task_processing.unseen_task_processing	lib.sedna.core.lifelong_learning.knowledge_management.cloud
module, 161	module, 189
lib.sedna.algorithms.unseen_task_processing.unseen_task_processing	lib.sedna.core.lifelong_learning.knowledge_management.edge
module, 161	module, 190
lib.sedna.algorithms.unseen_task_processing.unseen_task_processing	lib.sedna.core.lifelong_learning.knowledge_management.edge_inference
module, 162	module, 192
lib.sedna.algorithms.unseen_task_processing.unseen_task_processing	lib.sedna.core.lifelong_learning.knowledge_management.edge_inference
module, 162	module, 198
lib.sedna.backend	lib.sedna.core.multi_edge_inference.components
module, 165	module, 198
lib.sedna.backend.base	lib.sedna.core.multi_edge_inference.components.detector
module, 167	module, 198
lib.sedna.backend.mindspore	lib.sedna.core.multi_edge_inference.components.feature_ext
module, 165	module, 200
lib.sedna.backend.tensorflow	lib.sedna.core.multi_edge_inference.components.reid
module, 166	module, 201
lib.sedna.backend.torch	lib.sedna.core.multi_edge_inference.data_classes
module, 167	module, 207
lib.sedna.common	lib.sedna.core.multi_edge_inference.plugins
module, 168	module, 204
lib.sedna.common.class_factory	lib.sedna.core.multi_edge_inference.plugins.registered
module, 168	module, 204
lib.sedna.common.config	lib.sedna.core.multi_edge_inference.utils
module, 170	module, 208
lib.sedna.common.constant	lib.sedna.datasources
module, 171	module, 210
lib.sedna.common.file_ops	lib.sedna.datasources.kafka
module, 173	module, 210
lib.sedna.common.log	lib.sedna.datasources.kafka.consumer
module, 175	module, 210
lib.sedna.common.utils	lib.sedna.datasources.kafka.kafka_manager
module, 176	module, 210
lib.sedna.core	lib.sedna.datasources.kafka.producer
module, 176	module, 211
lib.sedna.core.base	lib.sedna.datasources.obs
module, 209	module, 212
lib.sedna.core.federated_learning	lib.sedna.datasources.obs.connector
module, 176	module, 212
lib.sedna.core.federated_learning.federated_learning	lib.sedna.service
module, 176	module, 214
lib.sedna.core.incremental_learning	lib.sedna.service.client
module, 179	module, 227
lib.sedna.core.incremental_learning.incremental_learning	lib.sedna.service.multi_edge_inference
module, 179	module, 214
lib.sedna.core.joint_inference	lib.sedna.service.multi_edge_inference.interface
module, 184	module, 214
lib.sedna.core.joint_inference.joint_inference	lib.sedna.service.multi_edge_inference.interface.detection

module, 214
 lib.sedna.service.multi_edge_inference.interface.feature_extractor
 module, 214
 lib.sedna.service.multi_edge_inference.interface.reid_endpoint
 module, 215
 lib.sedna.service.multi_edge_inference.server
 module, 216
 lib.sedna.service.multi_edge_inference.server.detection
 module, 216
 lib.sedna.service.multi_edge_inference.server.feature_extractor
 module, 217
 lib.sedna.service.multi_edge_inference.server.load_weights()
 module, 218
 lib.sedna.service.run_kb
 module, 228
 lib.sedna.service.server
 module, 221
 lib.sedna.service.server.aggregation
 module, 224
 lib.sedna.service.server.base
 module, 225
 lib.sedna.service.server.inference
 module, 226
 lib.sedna.service.server.knowledgeBase
 module, 221
 lib.sedna.service.server.knowledgeBase.database
 module, 221
 lib.sedna.service.server.knowledgeBase.model
 module, 221
 lib.sedna.service.server.knowledgeBase.server
 module, 224
 LIFELONG_JOB (lib.sedna.common.constant.K8sResourceKind
 attribute), 172
 LifelongLearning (class in lib.sedna.core.lifelong_learning), 195
 LifelongLearning (class in lib.sedna.core.lifelong_learning.lifelong_learning),
 192
 list_objects() (lib.sedna.datasources.obs.connector.OBSClientWrapper
 method), 212
 load() (lib.sedna.algorithms.multi_task_learning.MulTaskLearning
 method), 136
 load() (lib.sedna.algorithms.multi_task_learning.multi_task_learning.MultiTaskLearning
 method), 133
 load() (lib.sedna.algorithms.seen_task_learning.seen_task_learning.SeenTaskLearning
 method), 150
 load() (lib.sedna.algorithms.seen_task_learning.SeenTaskLearning
 method), 153
 load() (lib.sedna.algorithms.unseen_task_processing.unseen_task_processing.UnseenTaskProcessing
 method), 164
 load() (lib.sedna.algorithms.unseen_task_processing.UnseenTaskProcessing
 method), 165
 load() (lib.sedna.backend.base.BackendBase method),
 168
 load() (lib.sedna.backend.torch.TorchBackend method),
 167
 load() (lib.sedna.common.file_ops.FileOps class
 method), 174
 load() (lib.sedna.core.multi_edge_inference.plugins.PluggableModel
 method), 206
 load_anno_from_ids()
 lib.sedna.datasources.JSONDataParse
 method), 213
 load_weights() (lib.sedna.common.file_ops.FileOps
 class method), 174
 load_weights() (lib.sedna.backend.mindspore.MSBackend
 method), 166
 load_weights() (lib.sedna.backend.tensorflow.TFBackend
 method), 166
 LOG_LEVEL (in module lib.sedna.common.log), 175
 log_level (lib.sedna.common.config.BaseConfig
 attribute), 171
 Logger (class in lib.sedna.common.log), 175
 LOGGER (in module lib.sedna.common.log), 175
M
 main() (in module lib.sedna.service.run_kb), 228
 make_base_dir() (lib.sedna.common.file_ops.FileOps
 class method), 173
 make_dir() (lib.sedna.common.file_ops.FileOps class
 method), 173
 match_query_to_targets() (in module
 lib.sedna.algorithms.reid.multi_img_matching),
 138
 max_size (lib.sedna.service.client.AggregationClient at-
 tribute), 228
 MIN_TRAIN_SAMPLE (lib.sedna.common.constant.KBResourceConstant
 attribute), 172
 MistNet (class in lib.sedna.algorithms.aggregation), 124
 Model (class in lib.sedna.algorithms.multi_task_learning.task_jobs.artifact),
 128
 Model (class in lib.sedna.algorithms.seen_task_learning.artifact),
 128
 model_info() (lib.sedna.backend.base.BackendBase
 method), 168
 model_info() (lib.sedna.backend.tensorflow.TFBackend
 method), 166
 model_info() (lib.sedna.service.server.inference.InferenceServer
 method), 226
 model_info() (lib.sedna.service.server.InferenceServer
 method), 226
 model_name (lib.sedna.backend.base.BackendBase
 property), 168
 model_name (lib.sedna.common.config.BaseConfig at-
 tribute), 171
 model_name (lib.sedna.core.multi_edge_inference.plugins.PluggableModel
 property), 206

MODEL_NOT_FOUND	(in module lib.sedna.core.multi_edge_inference.plugins), 205	138 lib.sedna.algorithms.seen_task_learning, 139
model_path	(lib.sedna.core.base.JobBase property), 209	lib.sedna.algorithms.seen_task_learning.artifact,
model_path	(lib.sedna.core.multi_edge_inference.plugins.PluggableModel property), 206	lib.sedna.algorithms.seen_task_learning.inference_inte
model_url	(lib.sedna.common.config.BaseConfig attribute), 171	139 lib.sedna.algorithms.seen_task_learning.inference_inte
model_url	(lib.sedna.service.server.knowledgeBase.model.TaskModel attribute), 222	139 lib.sedna.algorithms.seen_task_learning.inference_inte
ModelClient	(class in lib.sedna.service.client), 228	139
ModelProbeFilter	(class in lib.sedna.algorithms.unseen_task_detect), 156	lib.sedna.algorithms.seen_task_learning.seen_task_lear 148 lib.sedna.algorithms.seen_task_learning.task_allocatio
ModelProbeFilter	(class in lib.sedna.algorithms.unseen_task_detect.unseen_task_ 156	140 lib.sedna.algorithms.seen_task_learning.task_allocatio
module		lib.sedna.algorithms.seen_task_learning.task_allocatio
lib.sedna	123	141
lib.sedna.__version__	229	lib.sedna.algorithms.seen_task_learning.task_allocatio
lib.sedna.algorithms	123	142
lib.sedna.algorithms.aggregation	123	lib.sedna.algorithms.seen_task_learning.task_allocatio
lib.sedna.algorithms.aggregation.aggregation	123	142 lib.sedna.algorithms.seen_task_learning.task_definitio
lib.sedna.algorithms.client_choose	125	143
lib.sedna.algorithms.client_choose.client_choose	125	lib.sedna.algorithms.seen_task_learning.task_definitio 143
lib.sedna.algorithms.hard_example_mining	126	lib.sedna.algorithms.seen_task_learning.task_definitio 143
lib.sedna.algorithms.hard_example_mining.hard_example_mining	126	lib.sedna.algorithms.seen_task_learning.task_definitio 144
lib.sedna.algorithms.multi_task_learning	128	lib.sedna.algorithms.seen_task_learning.task_relation_ 145
lib.sedna.algorithms.multi_task_learning.multitasklearning	132	lib.sedna.algorithms.seen_task_learning.task_relation_ 145
lib.sedna.algorithms.multi_task_learning.task_jobs	128	lib.sedna.algorithms.seen_task_learning.task_relation_ 145
lib.sedna.algorithms.multi_task_learning.task_jobs.edta_factory	128	lib.sedna.algorithms.seen_task_learning.task_remodelin 146
lib.sedna.algorithms.multi_task_learning.task_jobs.inference_inte	128	lib.sedna.algorithms.seen_task_learning.task_remodelin 146
lib.sedna.algorithms.multi_task_learning.task_jobs.task_definition	129	lib.sedna.algorithms.seen_task_learning.task_remodelin 146
lib.sedna.algorithms.multi_task_learning.task_jobs.task_aligning	130	lib.sedna.algorithms.seen_task_learning.task_update_de 147
lib.sedna.algorithms.multi_task_learning.task_jobs.task_algorithm_discover	130	lib.sedna.algorithms.seen_task_learning.task_update_de 147
lib.sedna.algorithms.multi_task_learning.task_jobs.task_algorithming	131	lib.sedna.algorithms.seen_task_learning.task_update_de 148
lib.sedna.algorithms.optical_flow	137	lib.sedna.algorithms.transmitter, 154
lib.sedna.algorithms.reid	137	lib.sedna.algorithms.transmitter.transmitter,
lib.sedna.algorithms.reid.close_contact_estimation	137	154
lib.sedna.algorithms.reid.multi_img_matching		lib.sedna.algorithms.unseen_task_detect, 155

lib.sedna.algorithms.unseen_task_detect.unseen_task_detection,	lib.sedna.core.lifelong_learning.knowledge_management,
155	188
lib.sedna.algorithms.unseen_task_detection,	lib.sedna.core.lifelong_learning.knowledge_management,
157	189
lib.sedna.algorithms.unseen_task_detection.unseen_sample_recognition,	lib.sedna.core.lifelong_learning.knowledge_management,
157	190
lib.sedna.algorithms.unseen_task_detection.unseen_sample_recognition,	lib.sedna.core.lifelong_learning.knowledge_management,
157	192
lib.sedna.algorithms.unseen_task_detection.unseen_sample_recognition,	lib.sedna.core.lifelong_learning.knowledge_management,
158	198
lib.sedna.algorithms.unseen_task_detection.unseen_sample_recognition,	lib.sedna.core.lifelong_learning.knowledge_management,
159	198
lib.sedna.algorithms.unseen_task_detection.unseen_sample_recognition,	lib.sedna.core.lifelong_learning.knowledge_management,
159	198
lib.sedna.algorithms.unseen_task_detection.unseen_sample_recognition,	lib.sedna.core.lifelong_learning.knowledge_management,
160	198
lib.sedna.algorithms.unseen_task_detection.unseen_sample_recognition,	lib.sedna.core.lifelong_learning.knowledge_management,
160	198
lib.sedna.algorithms.unseen_task_processing,	lib.sedna.core.lifelong_learning.knowledge_management,
161	198
lib.sedna.algorithms.unseen_task_processing.unseen_task_allocation,	lib.sedna.core.lifelong_learning.knowledge_management,
161	198
lib.sedna.algorithms.unseen_task_processing.unseen_task_allocation,	lib.sedna.core.lifelong_learning.knowledge_management,
161	198
lib.sedna.algorithms.unseen_task_processing.unseen_task_allocation,	lib.sedna.core.lifelong_learning.knowledge_management,
162	198
lib.sedna.algorithms.unseen_task_processing.unseen_task_allocation,	lib.sedna.core.lifelong_learning.knowledge_management,
162	198
lib.sedna.backend,	lib.sedna.core.lifelong_learning.knowledge_management,
165	198
lib.sedna.backend.base,	lib.sedna.core.lifelong_learning.knowledge_management,
167	198
lib.sedna.backend.mindspore,	lib.sedna.core.lifelong_learning.knowledge_management,
165	198
lib.sedna.backend.tensorflow,	lib.sedna.core.lifelong_learning.knowledge_management,
166	198
lib.sedna.backend.torch,	lib.sedna.core.lifelong_learning.knowledge_management,
167	198
lib.sedna.common,	lib.sedna.core.lifelong_learning.knowledge_management,
168	198
lib.sedna.common.class_factory,	lib.sedna.core.lifelong_learning.knowledge_management,
168	198
lib.sedna.common.config,	lib.sedna.core.lifelong_learning.knowledge_management,
170	198
lib.sedna.common.constant,	lib.sedna.core.lifelong_learning.knowledge_management,
171	198
lib.sedna.common.file_ops,	lib.sedna.core.lifelong_learning.knowledge_management,
173	198
lib.sedna.common.log,	lib.sedna.core.lifelong_learning.knowledge_management,
175	198
lib.sedna.common.utils,	lib.sedna.core.lifelong_learning.knowledge_management,
176	198
lib.sedna.core,	lib.sedna.core.lifelong_learning.knowledge_management,
176	198
lib.sedna.core.base,	lib.sedna.core.lifelong_learning.knowledge_management,
209	198
lib.sedna.core.federated_learning,	lib.sedna.core.lifelong_learning.knowledge_management,
176	198
lib.sedna.core.federated_learning.federated_learning,	lib.sedna.core.lifelong_learning.knowledge_management,
176	198
lib.sedna.core.incremental_learning,	lib.sedna.core.lifelong_learning.knowledge_management,
179	198
lib.sedna.core.incremental_learning.incremental_learning,	lib.sedna.core.lifelong_learning.knowledge_management,
179	198
lib.sedna.core.joint_inference,	lib.sedna.core.lifelong_learning.knowledge_management,
184	198
lib.sedna.core.joint_inference.joint_inference,	lib.sedna.core.lifelong_learning.knowledge_management,
184	198
lib.sedna.core.lifelong_learning,	lib.sedna.core.lifelong_learning.knowledge_management,
188	198
lib.sedna.core.lifelong_learning.knowledge_management,	lib.sedna.core.lifelong_learning.knowledge_management,
188	198

lib.sedna.service.server, 221
 lib.sedna.service.server.aggregation, 224
 lib.sedna.service.server.base, 225
 lib.sedna.service.server.inference, 226
 lib.sedna.service.server.knowledgeBase, 221
 lib.sedna.service.server.knowledgeBase.database, 221
 lib.sedna.service.server.knowledgeBase.model, 221
 lib.sedna.service.server.knowledgeBase.server, 224
 MSBackend (class in lib.sedna.backend.mindspore), 165
 MTL (lib.sedna.common.class_factory.ClassType attribute), 169
 MulTaskLearning (class in lib.sedna.algorithms.multi_task_learning), 134
 MulTaskLearning (class in lib.sedna.algorithms.multi_task_learning.multi_task_learning), 132
N
 name (lib.sedna.service.server.knowledgeBase.model.TaskGrp attribute), 222
 name (lib.sedna.service.server.knowledgeBase.model.Tasks attribute), 222
 namespace (lib.sedna.common.config.BaseConfig attribute), 171
 NOOP (lib.sedna.core.multi_edge_inference.data_classes.OP_MODE attribute), 207
 num_examples() (lib.sedna.datasources.BaseDataSource method), 213
 num_samples (lib.sedna.algorithms.aggregation.AggrClient attribute), 125
 num_samples (lib.sedna.algorithms.aggregation.aggregation_client attribute), 123
O
 obj_to_pickle_string() (lib.sedna.common.file_ops.FileOps class method), 175
 ObjectDetector (class in lib.sedna.core.multi_edge_inference.components.detector), 198
 OBSClientWrapper (class in lib.sedna.datasources.obs.connector), 212
 OF (lib.sedna.common.class_factory.ClassType attribute), 169
 on_send_error() (lib.sedna.datasources.kafka.producer.Producer method), 211
 on_send_success() (lib.sedna.datasources.kafka.producer.Producer method), 211
 OP_MODE (class in lib.sedna.core.multi_edge_inference.data_classes), 207
 original_dataset_url (lib.sedna.common.config.BaseConfig attribute), 171
P
 parameters (lib.sedna.common.config.BaseConfig attribute), 171
 parameters (lib.sedna.common.config.Context attribute), 171
 parameters (lib.sedna.core.base.JobBase attribute), 209
 parse() (lib.sedna.datasources.BaseDataSource method), 213
 parse() (lib.sedna.datasources.CSVDataParse method), 213
 parse() (lib.sedna.datasources.JSONDataParse method), 213
 parse() (lib.sedna.datasources.TxtDataParse method), 213
 parse_json() (lib.sedna.datasources.CSVDataParse static method), 213
 parse_kwargs() (lib.sedna.backend.base.BackendBase static method), 168
 pause() (lib.sedna.datasources.kafka.consumer.Consumer method), 210
 pickle_string_to_obj() (lib.sedna.common.file_ops.FileOps class method), 175
 PluggableModel (class in lib.sedna.core.multi_edge_inference.plugins), 206
 PluggableNetworkService (class in lib.sedna.core.multi_edge_inference.plugins), 206
 PLUGGABLE_NETWORK_SERVICE (class in lib.sedna.core.multi_edge_inference.plugins), 205
 POLL_INTERVAL (in module lib.sedna.core.multi_edge_inference), 208
 POLL_INTERVAL (in module lib.sedna.core.multi_edge_inference.components), 203
 predict() (lib.sedna.algorithms.multi_task_learning.MulTaskLearning method), 136
 predict() (lib.sedna.algorithms.multi_task_learning.multi_task_learning method), 134
 predict() (lib.sedna.algorithms.seen_task_learning.seen_task_learning method), 150
 predict() (lib.sedna.algorithms.seen_task_learning.SeenTaskLearning method), 153
 predict() (lib.sedna.algorithms.unseen_task_processing.unseen_task_processing method), 163
 predict() (lib.sedna.algorithms.unseen_task_processing.UnseenTaskProcessing method), 165

`predict()` (*lib.sedna.backend.base.BackendBase* method), 168
`predict()` (*lib.sedna.backend.mindspore.MSBackend* method), 165
`predict()` (*lib.sedna.backend.tensorflow.TFBackend* method), 166
`predict()` (*lib.sedna.backend.torch.TorchBackend* method), 167
`predict()` (*lib.sedna.service.server.inference.InferenceServer* method), 226
`predict()` (*lib.sedna.service.server.InferenceServer* method), 226
`predict_proba()` (*lib.sedna.backend.base.BackendBase* method), 168
`prep_homography()` (*lib.sedna.algorithms.reid.close_contact_estimation.close_contact_tracker* method), 137
`preprocess()` (*lib.sedna.core.multi_edge_inference.BaseService* class method), 209
`preprocess()` (*lib.sedna.core.multi_edge_inference.components.BaseService* method), 203
`preprocess()` (*lib.sedna.core.multi_edge_inference.components.detection_object_detector* method), 200
`pretrained_model_url` (*lib.sedna.common.config.BaseConfig* attribute), 171
`process_data()` (*lib.sedna.core.multi_edge_inference.BaseService* class method), 209
`process_data()` (*lib.sedna.core.multi_edge_inference.components.BaseService* method), 203
`process_data()` (*lib.sedna.core.multi_edge_inference.components.detection_object_detector* method), 200
`process_data()` (*lib.sedna.core.multi_edge_inference.components.federated_learning.FEService* method), 201
`process_data()` (*lib.sedna.core.multi_edge_inference.components.reid.ReID* method), 202
`Producer` (class in *lib.sedna.datasources.kafka.producer*), 211
`publish_data_asynchronous()` (*lib.sedna.datasources.kafka.producer.Producer* method), 211
`publish_data_synchronous()` (*lib.sedna.datasources.kafka.producer.Producer* method), 211
`put()` (*lib.sedna.core.multi_edge_inference.BaseService* method), 208
`put()` (*lib.sedna.core.multi_edge_inference.components.BaseService* method), 203

Q

`query()` (*lib.sedna.service.server.knowledgeBase.server.KBServer* method), 224

R

`read_from_disk()` (*lib.sedna.core.multi_edge_inference.components.FileOperations* method), 204
`read_from_disk()` (*lib.sedna.core.multi_edge_inference.FileOperations* method), 209
`recv()` (*lib.sedna.algorithms.transmitter.S3Transmitter* method), 155
`recv()` (*lib.sedna.algorithms.transmitter.transmitter.AbstractTransmitter* method), 154
`recv()` (*lib.sedna.algorithms.transmitter.transmitter.S3Transmitter* method), 155
`recv()` (*lib.sedna.algorithms.transmitter.transmitter.WSTransmitter* method), 154
`recv()` (*lib.sedna.algorithms.transmitter.WSTransmitter* method), 155
`recv()` (*lib.sedna.service.client.AggregationClient* method), 228
`register()` (*lib.sedna.common.class_factory.ClassFactory* class method), 169
`register()` (*lib.sedna.core.federated_learning.federated_learning.FederatedLearning* method), 177
`register()` (*lib.sedna.core.federated_learning.FederatedLearning* method), 177
`register_cls()` (*lib.sedna.common.class_factory.ClassFactory* class method), 169
`register_from_package()` (*lib.sedna.common.class_factory.ClassFactory* class method), 170
`ReID` (class in *lib.sedna.core.multi_edge_inference.components.reid*), 203
`REID` (*lib.sedna.core.multi_edge_inference.plugins.PLUGIN* attribute), 206
`reid()` (*lib.sedna.service.multi_edge_inference.ReIDServer* method), 219
`reid()` (*lib.sedna.service.multi_edge_inference.server.reid.ReIDServer* method), 219
`reid()` (*lib.sedna.service.multi_edge_inference.server.ReIDServer* method), 219
`ReID_Endpoint` (class in *lib.sedna.service.multi_edge_inference*), 220
`ReID_Endpoint` (class in *lib.sedna.service.multi_edge_inference.interface*), 216
`ReID_Endpoint` (class in *lib.sedna.service.multi_edge_inference.interface.reid_endpoint*), 215
`ReID_I` (class in *lib.sedna.core.multi_edge_inference.plugins.registered*), 204
`REID_I` (*lib.sedna.core.multi_edge_inference.plugins.PLUGIN* attribute), 206
`REID_JOB` (*lib.sedna.common.constant.K8sResourceKind* attribute), 172
`REID_MANAGER` (*lib.sedna.core.multi_edge_inference.plugins.PLUGIN* attribute), 206
`REID_MANAGER_I` (*lib.sedna.core.multi_edge_inference.plugins.PLUGIN* attribute), 206

attribute), 206
 ReID_Server (class in Samples (class in lib.sedna.service.server.knowledgeBase.model),
 lib.sedna.core.multi_edge_inference.plugins.registered), 223
 204 save() (lib.sedna.backend.base.BackendBase method),
 ReIDServer (class in lib.sedna.service.multi_edge_inference), 168
 220 save() (lib.sedna.datasources.BaseDataSource
 ReIDServer (class in lib.sedna.service.multi_edge_inference.server), method), 213
 219 save_task_index() (lib.sedna.core.lifelong_learning.knowledge_manage
 ReIDServer (class in lib.sedna.service.multi_edge_inference.server.run), method), 189
 218 save_task_index() (lib.sedna.core.lifelong_learning.knowledge_manage
 remove_path_prefix() method), 191
 (lib.sedna.common.file_ops.FileOps class save_task_index() (lib.sedna.core.lifelong_learning.knowledge_manage
 method), 173 method), 189
 report_task_info() (lib.sedna.core.base.JobBase save_task_index() (lib.sedna.core.lifelong_learning.knowledge_manage
 method), 209 method), 191
 resume() (lib.sedna.datasources.kafka.consumer.Consumer save_task_index() (lib.sedna.core.lifelong_learning.knowledge_manage
 method), 210 method), 190
 run() (lib.sedna.algorithms.unseen_task_detection.unseen_sample_task_index() (lib.sedna.core.lifelong_learning.unseen_sample_detection_manage
 method), 160 method), 192
 run() (lib.sedna.datasources.kafka.kafka_manager.KafkaConsumer.unseen_samples() (lib.sedna.core.lifelong_learning.knowledge_management.edge_k
 method), 211 method), 190
 run() (lib.sedna.service.client.LCReporter method), 227
 run() (lib.sedna.service.server.base.BaseServer save_unseen_samples() (lib.sedna.core.lifelong_learning.knowledge_management.EdgeK
 method), 225 method), 192
 run_in_thread() (lib.sedna.service.server.base.Server
 method), 225
 RUNNING (lib.sedna.common.constant.K8sResourceKindStatus attribute), 171
 attribute), 172 SEEN_TASK (lib.sedna.common.constant.KBResourceConstant
 attribute), 172
S SeenTaskLearning (class in
 lib.sedna.algorithms.seen_task_learning),
 s3_download() (lib.sedna.common.file_ops.FileOps 151
 class method), 174
 s3_endpoint_url (lib.sedna.common.config.BaseConfig SeenTaskLearning (class in
 attribute), 171 lib.sedna.algorithms.seen_task_learning.seen_task_learning),
 s3_upload() (lib.sedna.common.file_ops.FileOps class 148
 method), 174
 S3Transmitter (class in
 lib.sedna.algorithms.transmitter), 155
 S3Transmitter (class in
 lib.sedna.algorithms.transmitter.transmitter), 154
 154
 sample (lib.sedna.service.server.knowledgeBase.model.TaskSample send() (lib.sedna.algorithms.transmitter.transmitter.WSTransmitter
 attribute), 223 method), 154
 sample_id (lib.sedna.service.server.knowledgeBase.model.TaskSample send() (lib.sedna.algorithms.transmitter.WSTransmitter
 attribute), 223 method), 155
 sample_num (lib.sedna.service.server.knowledgeBase.model.TaskSample send() (lib.sedna.service.client.AggregationClient
 attribute), 223 method), 228
 sample_num (lib.sedna.service.server.knowledgeBase.model.TaskSample send() (lib.sedna.service.client.LCClient class method),
 attribute), 222 228
 SampleRegonitionDefault (class in Server (class in lib.sedna.service.server.base), 225
 lib.sedna.algorithms.unseen_task_detection.unseen_sample_recognition.unseen_sample_recognition),
 160
 SampleReRegonitionDefault (class in SessionLocal (in module
 lib.sedna.algorithms.unseen_task_detection.unseen_sample_recognition.unseen_sample_recognition),
 160

221

`set_backend()` (in module `lib.sedna.backend`), 168

`set_session()` (`lib.sedna.backend.tensorflow.KerasBackend` method), 167

`set_weights()` (`lib.sedna.backend.base.BackendBase` method), 168

`set_weights()` (`lib.sedna.backend.mindspore.MSBackend` method), 166

`set_weights()` (`lib.sedna.backend.tensorflow.KerasBackend` method), 167

`set_weights()` (`lib.sedna.backend.tensorflow.TFBackend` method), 166

`SimpleClientChoose` (class in `lib.sedna.algorithms.client_choose`), 126

`SimpleClientChoose` (class in `lib.sedna.algorithms.client_choose.client_choose`), 125

`singleton()` (in module `lib.sedna.common.utils`), 176

`SQLALCHEMY_DATABASE_URL` (in module `lib.sedna.service.server.knowledgeBase.database`), 221

`start()` (`lib.sedna.core.joint_inference.BigModelService` method), 188

`start()` (`lib.sedna.core.joint_inference.joint_inference.BigModelService` method), 184

`start()` (`lib.sedna.service.multi_edge_inference.DetectionServer` method), 220

`start()` (`lib.sedna.service.multi_edge_inference.FEServer` method), 220

`start()` (`lib.sedna.service.multi_edge_inference.ReIDServer` method), 220

`start()` (`lib.sedna.service.multi_edge_inference.server.detection.DetectionServer` method), 217

`start()` (`lib.sedna.service.multi_edge_inference.server.DetectionServer` method), 218

`start()` (`lib.sedna.service.multi_edge_inference.server.feature_extraction.FEServer` method), 217

`start()` (`lib.sedna.service.multi_edge_inference.server.FEServer` method), 219

`start()` (`lib.sedna.service.multi_edge_inference.server.reid.ReIDServer` method), 218

`start()` (`lib.sedna.service.multi_edge_inference.server.ReIDServer` method), 219

`start()` (`lib.sedna.service.server.aggregation.AggregationServer` method), 225

`start()` (`lib.sedna.service.server.aggregation.AggregationServerV2` method), 225

`start()` (`lib.sedna.service.server.AggregationServer` method), 227

`start()` (`lib.sedna.service.server.AggregationServerV2` method), 227

`start()` (`lib.sedna.service.server.inference.InferenceServer` method), 226

`start()` (`lib.sedna.service.server.InferenceServer` method), 226

`start()` (`lib.sedna.service.server.knowledgeBase.server.KBServer` method), 224

`start_services()` (`lib.sedna.core.lifelong_learning.knowledge_management` method), 190

`start_services()` (`lib.sedna.core.lifelong_learning.knowledge_management` method), 192

`status` (`lib.sedna.service.server.knowledgeBase.server.KBUpdateResult` attribute), 224

`status()` (`lib.sedna.service.multi_edge_inference.DetectionServer` method), 220

`status()` (`lib.sedna.service.multi_edge_inference.FEServer` method), 220

`status()` (`lib.sedna.service.multi_edge_inference.ReIDServer` method), 221

`status()` (`lib.sedna.service.multi_edge_inference.server.detection.DetectionServer` method), 217

`status()` (`lib.sedna.service.multi_edge_inference.server.DetectionServer` method), 218

`status()` (`lib.sedna.service.multi_edge_inference.server.feature_extraction.FEServer` method), 217

`status()` (`lib.sedna.service.multi_edge_inference.server.FEServer` method), 219

`status()` (`lib.sedna.service.multi_edge_inference.server.reid.ReIDServer` method), 218

`status()` (`lib.sedna.service.multi_edge_inference.server.ReIDServer` method), 219

`STP` (`lib.sedna.common.class_factory.ClassType` attribute), 169

`subscribe()` (`lib.sedna.datasources.kafka.consumer.Consumer` method), 210

T

`target` (class in `lib.sedna.core.multi_edge_inference.data_classes`), 207

`target_images` (`lib.sedna.core.multi_edge_inference.data_classes`), 207

`Task` (class in `lib.sedna.algorithms.multi_task_learning.task_jobs.artifact`), 128

`Task` (class in `lib.sedna.algorithms.seen_task_learning.artifact`), 148

`task` (`lib.sedna.service.server.knowledgeBase.model.TaskModel` attribute), 222

`task` (`lib.sedna.service.server.knowledgeBase.model.TaskRelation` attribute), 223

`task` (`lib.sedna.service.server.knowledgeBase.model.TaskSample` attribute), 223

`task_attr` (`lib.sedna.service.server.knowledgeBase.model.Tasks` attribute), 222

`TASK_EXTRACTOR_NAME` (`lib.sedna.common.constant.KBResourceConstant` attribute), 172

TASK_GROUPS (*lib.sedna.common.constant.KBResourceConstant* attribute), 172
task_id (*lib.sedna.service.server.knowledgeBase.model.TaskModel* attribute), 222
task_id (*lib.sedna.service.server.knowledgeBase.model.TaskRelation* attribute), 223
task_id (*lib.sedna.service.server.knowledgeBase.model.TaskModel* attribute), 223
task_num (*lib.sedna.service.server.knowledgeBase.model.TaskRelation* attribute), 222
TaskAllocationByDataAttr (class in *lib.sedna.algorithms.seen_task_learning.task_allocation*), 141
TaskAllocationByOrigin (class in *lib.sedna.algorithms.seen_task_learning.task_allocation.task_allocation_by_origin*), 142
TaskAllocationBySVC (class in *lib.sedna.algorithms.seen_task_learning.task_allocation*), 141
TaskAllocationDefault (class in *lib.sedna.algorithms.seen_task_learning.task_allocation.task_allocation*), 142
TaskAllocationStream (class in *lib.sedna.algorithms.seen_task_learning.task_allocation.task_allocation_stream*), 142
TaskAttrFilter (class in *lib.sedna.algorithms.unseen_task_detect*), 157
TaskAttrFilter (class in *lib.sedna.algorithms.unseen_task_detect.unseen_task_rectangle*), 156
TaskDefinitionByDataAttr (class in *lib.sedna.algorithms.multi_task_learning.task_definition*), 129
TaskDefinitionByDataAttr (class in *lib.sedna.algorithms.seen_task_learning.task_definition*), 144
TaskDefinitionByOrigin (class in *lib.sedna.algorithms.seen_task_learning.task_definition*), 144
TaskDefinitionBySVC (class in *lib.sedna.algorithms.seen_task_learning.task_definition*), 129
TaskDefinitionBySVC (class in *lib.sedna.algorithms.seen_task_learning.task_definition*), 144
TaskGroup (class in *lib.sedna.algorithms.multi_task_learning.task_group*), 128
TaskGroup (class in *lib.sedna.algorithms.seen_task_learning.task_group*), 148
TaskGrp (class in *lib.sedna.service.server.knowledgeBase.model*), 222
TaskItem (class in *lib.sedna.service.server.knowledgeBase.model*), 224

train()	(lib.sedna.core.base.JobBase method), 209	UnseenTaskAllocationDefault	(class in lib.sedna.algorithms.unseen_task_processing.unseen_task_allocation), 160
train()	(lib.sedna.core.federated_learning.federated_learning.FederatedLearningV2 method), 177	UnseenTaskProcessing	(class in lib.sedna.algorithms.unseen_task_processing), 164
train()	(lib.sedna.core.federated_learning.FederatedLearning method), 178	UnseenTaskProcessing	(class in lib.sedna.algorithms.unseen_task_processing.unseen_task_processing), 164
train()	(lib.sedna.core.federated_learning.FederatedLearning method), 179	UnseenTaskProcessing	(class in lib.sedna.algorithms.unseen_task_processing.unseen_task_processing), 164
train()	(lib.sedna.core.incremental_learning.incremental_learning.IncrementalLearning method), 180	update()	(lib.sedna.algorithms.seen_task_learning.seen_task_learning.SeenTaskLearning method), 150
train()	(lib.sedna.core.incremental_learning.IncrementalLearning method), 182	update()	(lib.sedna.algorithms.seen_task_learning.SeenTaskLearning method), 153
train()	(lib.sedna.core.joint_inference.BigModelService method), 188	update()	(lib.sedna.algorithms.unseen_task_processing.unseen_task_processing), 163
train()	(lib.sedna.core.joint_inference.joint_inference.BigModelService method), 184	update()	(lib.sedna.algorithms.unseen_task_processing.UnseenTaskProcessing), 164
train()	(lib.sedna.core.lifelong_learning.lifelong_learning.LifelongLearning method), 194	update()	(lib.sedna.backend.base.BackendBase method), 168
train()	(lib.sedna.core.lifelong_learning.LifelongLearning method), 197	update()	(lib.sedna.core.lifelong_learning.lifelong_learning.LifelongLearning method), 194
train()	(lib.sedna.core.multi_edge_inference.plugins.PluggableModelService method), 206	update()	(lib.sedna.core.lifelong_learning.LifelongLearning method), 197
train_dataset_url	(lib.sedna.common.config.BaseConfig attribute), 171	update()	(lib.sedna.service.server.knowledgeBase.server.KBServer method), 224
transfer_radio	(lib.sedna.service.server.knowledgeBase.model.TaskRadio attribute), 223	update_db()	(lib.sedna.service.client.KBClient method), 227
transmit()	(lib.sedna.service.multi_edge_inference.Detection method), 219	update_for_collaboration_inference()	(lib.sedna.service.client.LCReporter method), 227
transmit()	(lib.sedna.service.multi_edge_inference.FE method), 220	update_for_collaboration_inference()	(lib.sedna.service.client.LCReporter method), 227
transmit()	(lib.sedna.service.multi_edge_inference.interface.Detection method), 214	update_kb()	(lib.sedna.core.lifelong_learning.knowledge_management.knowledge_management.BaseService method), 189
transmit()	(lib.sedna.service.multi_edge_inference.interface.FE method), 216	update_kb()	(lib.sedna.core.lifelong_learning.knowledge_management.knowledge_management.BaseService method), 189
transmit()	(lib.sedna.service.multi_edge_inference.interface.fe_endpoint.Endpoint method), 215	update_kb()	(lib.sedna.core.lifelong_learning.knowledge_management.knowledge_management.BaseService method), 189
transmit()	(lib.sedna.service.multi_edge_inference.interface.ReID_Endpoint method), 215	update_kb()	(lib.sedna.core.lifelong_learning.knowledge_management.knowledge_management.BaseService method), 189
transmit()	(lib.sedna.service.multi_edge_inference.interface.ReID_Endpoint method), 215	update_kb()	(lib.sedna.core.lifelong_learning.knowledge_management.knowledge_management.BaseService method), 189
transmitter	(lib.sedna.common.config.BaseConfig attribute), 171	update_operational_mode()	(lib.sedna.core.multi_edge_inference.components.BaseService method), 203
TxtDataParse	(class in lib.sedna.datasources), 213	update_operational_mode()	(lib.sedna.core.multi_edge_inference.components.BaseService method), 203
UNSEEN_TASK	(lib.sedna.common.constant.KBResourceConstant attribute), 172	update_operational_mode()	(lib.sedna.core.multi_edge_inference.components.BaseService method), 203
UnseenSampleDetection	(class in lib.sedna.algorithms.unseen_task_detection.unseen_sample_detection.unseen_sample_detection), 209	update_operational_mode()	(lib.sedna.core.multi_edge_inference.components.BaseService method), 203

method), 200
 update_operational_mode()
 (lib.sedna.core.multi_edge_inference.components.feature_extraction.FEService
 method), 201
 update_operational_mode()
 (lib.sedna.core.multi_edge_inference.components.raid.ReID
 method), 202
 update_plugin() (lib.sedna.core.multi_edge_inference.plugins.PluggableModel
 method), 206
 update_service() (lib.sedna.service.multi_edge_inference.DetectionServer
 method), 219
 update_service() (lib.sedna.service.multi_edge_inference.DetectionServer
 method), 220
 update_service() (lib.sedna.service.multi_edge_inference.FE
 method), 220
 update_service() (lib.sedna.service.multi_edge_inference.FEServer
 method), 220
 update_service() (lib.sedna.service.multi_edge_inference.interface.Detection
 method), 216
 update_service() (lib.sedna.service.multi_edge_inference.interface.detection_endpoint.Detection
 method), 214
 update_service() (lib.sedna.service.multi_edge_inference.Interface.FE
 method), 216
 update_service() (lib.sedna.service.multi_edge_inference.interface.fe_endpoint.FE
 method), 215
 update_service() (lib.sedna.service.multi_edge_inference.server.detection.DetectionServer
 method), 217
 update_service() (lib.sedna.service.multi_edge_inference.server.DetectionServer
 method), 218
 update_service() (lib.sedna.service.multi_edge_inference.server.feature_extraction.FEServer
 method), 217
 update_service() (lib.sedna.service.multi_edge_inference.server.FEServer
 method), 219
 update_status() (lib.sedna.service.server.knowledgeBase.server.KBServer
 method), 224
 update_task_status()
 (lib.sedna.service.client.KBClient method),
 228
 updated_at (lib.sedna.service.server.knowledgeBase.model.Samples
 attribute), 223
 updated_at (lib.sedna.service.server.knowledgeBase.model.Tasks
 attribute), 222
 upload() (lib.sedna.common.file_ops.FileOps class
 method), 174
 upload_file() (lib.sedna.datasources.obs.connector.OBSClientWrapper
 method), 212
 upload_file() (lib.sedna.service.client.KBClient
 method), 228
 UTD (lib.sedna.common.class_factory.ClassType at-
 tribute), 169
 UTP (lib.sedna.common.class_factory.ClassType at-
 tribute), 169

V

VIDEO_ANALYTICS (lib.sedna.core.multi_edge_inference.plugins.PLUGIN
 attribute), 206
 video_analytics() (lib.sedna.service.multi_edge_inference.DetectionSer
 method), 220
 video_analytics() (lib.sedna.service.multi_edge_inference.server.detect
 method), 217
 video_analytics() (lib.sedna.service.multi_edge_inference.server.Detect
 method), 218
 VIDEO_ANALYTICS_I (lib.sedna.core.multi_edge_inference.plugins.PLUG
 attribute), 206
 VIDEO_ANALYTICS_JOB
 (lib.sedna.common.constant.K8sResourceKind
 attribute), 172
 VideoAnalytics (class in
 lib.sedna.core.multi_edge_inference.plugins.registered),
 205
 VideoAnalytics_I (class in
 lib.sedna.core.multi_edge_inference.plugins.registered),
 205

W

wait_stop() (lib.sedna.service.server.base.BaseServer
 method), 226
 WAIT_TIME (lib.sedna.service.server.base.BaseServer at-
 tribute), 225
 weights (lib.sedna.algorithms.aggregation.AggregClient
 attribute), 125
 weights (lib.sedna.algorithms.aggregation.aggregation.AggregClient
 attribute), 125
 worker_name (lib.sedna.common.config.BaseConfig at-
 tribute), 171
 write_result() (lib.sedna.datasources.kafka.kafka_manager.KafkaProdu
 method), 211
 write_to_disk() (lib.sedna.core.multi_edge_inference.components.FileC
 method), 204
 write_to_disk() (lib.sedna.core.multi_edge_inference.FileOperations
 method), 209
 WSTransmitter (class in
 lib.sedna.algorithms.transmitter), 155
 WSTransmitter (class in
 lib.sedna.algorithms.transmitter.transmitter),
 154