KubeEdge-AI Intro
Edge AI Challenges

- Geo-distributed dataset across edges
- Few-shot samples per edge for training, cold booting, hard to converge
- Non-I.I.D data across edges, the performance of universal AI model degraded on edge
- Resource constrained on edge
**What we propose:**

1. an edge-cloud collaborative ML framework based on KubeEdge
2. with embed collaborative training and joint inferencing algorithm, which can
3. working with existing AI framework like Tensorflow, etc

**3 Features:**

1. joint inferencing
2. incremental learning
3. collaborative training (aka federated learning)

**Targeting Users:**

1. Domain-specific AI Developers: build and publish edge-cloud collaborative AI services/functions easily
2. Application Developers: use edge-cloud collaborative AI capabilities.

We are NOT:

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

**Workers:**
- do inferencing 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:**
- expose the Edge AI features to applications, i.e. training or inferencing programs.

**GlobalCoordinator**
- uniportal of EdgeAI,
- across-edges coordination

**LocalController**
- local controller
- manage local dataset and models

**Platform:**
- KubeEdge@Edge + Arm Server(Atlas), x86 Server, smart camera(Hilense), etc
Edge-cloud Collaborative JOINT INFRINGEMENT
Improve the inference performance, when edge resources are limited.

1. AI developer: provides training data to generate deep and shallow models.
2. Service developers: invoke collaboration models through the library and deploy the models to the edge.
3. Inference based on the shallow model on the edge side. If the confidence requirement is met, the result is returned.
4. Otherwise, the data is sent to the cloud for deep model inference.
Edge-cloud Collaborative INCREMENTAL LEARNING

The more models are used, the smarter they are.

1. App developers: Use the Edge AI library during development to integrate the edge-cloud collaborative incremental learning function.

2. Deploy the app and start incremental learning.

3. The hard sample detection algorithm in the Edge AI library identify the samples with low inference confidence, and upload them to the cloud labeling service.

4. Manually and periodically label samples.

5. The system automatically performs incremental training based on the preset policy to generate a new model.

6. Updating the new model to Edge
Edge-cloud collaborative FEDERATED LEARNING

Raw data is not transmitted out of the edge, and the model is generated by knowledge aggregation.

- **Multi-task detection:** Divide non-IID sample sets and work with the cloud to identify similar tasks.
- **Local training:** Model parameters are uploaded to the cloud, and the cross-edge transferring and model aggregation algorithms are running on the cloud.

1. Developer: Import the Edge AI library and develop the edge-cloud collaborative federated learning program.
2. Start the federated learning task and deploy the training program to the edge.
Developer perspective: JOINT INFERENCEx

Design Objectives: Try not to change the existing code of developers and do not require developers to learn new frameworks, reducing learning costs.

How To Use:

- **Importing the Edge AI library**: Developers use the familiar ML framework (such as TensorFlow) to import the edge AI library (solar_corona library in the figure).

- **JOINT INFERENCE**: Replace the original load model object part, configure and generate the edge-cloud synergy model, and the background automatically generates a large model on the cloud. Developers do not need to change other parts of the code.

```
import hllens
import solar_corona

def pre_fun():
    ...# Preprocessing function may be, for example, image rotation or resize.
    return

def post_fun():
    ...# Post-processing function may be, for example, a post-processing module NMS in the object detection framework.
    return

def run():
    # endpoint may be: 1. Services started by users' own models 2. Existing cloud services
    big_model_endpoint = solar_corona.JOINT_INFERENCEx.get_big_model_endpoint()  # deep model on cloud
    ibt = solar_corona.JOINT_INFERENCEx.ibt(upload_ratio=0.5)  # "transfer to cloud" algorithm
    model_path = solar_corona.context.get_model_path()

    # Configure the edge-cloud model, including:
    # the local path of the model,
    # local pre-processing,
    # post-processing methods,
    # cloud migration algorithm,
    # cloud migration endpoint.

    model = solar_corona.load_model(model_path, pre_fun, post_fun,
                                    cloud_offload_algorithm=ibt,
                                    big_model_endpoint=big_model_endpoint)

    # Service parameters settings.
    camera_address = solar_corona.context.get_parameters('ip_camera_address')
    camera = hllens.VideoCapture(camera_address)

    # Service related code.
    while True:
        image = read_one_frame_from_camera(camera)
        predictions = model.predict(image)

    if __name__ == '__main__':
        run()
```
Developer perspective: FEDERATED LEARNING

Design Objectives: Try not to change the existing code of developers and do not require developers to learn new frameworks, reducing learning costs.

How To Use:

- Importing the Edge AI library: Developers use the familiar ML framework (such as TensorFlow) to import the edge AI library (solar_corona library in the figure).

- FEDERATED LEARNING: Import the local training loss function, optimizer, and the collaborative_train function from the solar_corona library.

```python
def main():
    # load dataset.
    (x, y) = solar_corona.load_train_dataset()
    (x_test, y_test) = solar_corona.load_eval_dataset()
    x, x_test = normalize(x, x_test)

    # read parameters from deployment config.
    epochs = solar_corona.context.get_parameters('epochs')
    batch_size = solar_corona.context.get_parameters('batch_size')
    aggregation_algorithm = solar_corona.context.get_parameters('aggregation_algorithm')

    train_loader = tf.data.Dataset.from_tensor_slices((x, y))
    train_loader = train_loader.map(prepare_cifar).shuffle(10000).batch(batch_size)
    test_loader = test_loader.map(prepare_cifar).shuffle(10000).batch(batch_size)
    model = VGG16([32, 32, 3])

    # you can use the loss/metric function defined in keras
    # loss = keras.losses.CategoricalCrossentropy(from_logits=True)
    # metric = keras.metrics.CategoricalAccuracy()
    # you can also use the loss/metric function defined in solar_corona
    loss = solar_corona.losses.ADifferentCategoricalCrossentropy()
    metric = solar_corona.metrics.ADifferentCategoricalAccuracy()

    # optimizer = keras.optimizers.Adam(learning_rate=0.0001)
    optimizer = solar_corona.optimizers.ADifferentAdam(learning_rate=0.0001)

    model = solar_corona.collaborative_training.fit(train_loader=train_loader,
                                                    test_loader=test_loader,
                                                    model=model,
                                                    loss=loss,
                                                    metric=metric,
                                                    optimizer=optimizer,
                                                    batch_size=batch_size,
                                                    epochs=epochs,
                                                    # config the aggregation_algorithm
                                                    aggregation_algorithm=aggregation_algorithm)

    # save the model based on the config.
    solar_corona.save_model(model)
```

FEDERATED LEARNING code example (Based on TensorFlow)
Thank you