

---

# **lanvs**

***Release v0.1***

**KubeEdge SIG AI**

**Sep 20, 2023**



# INTRODUCTION

<b>1</b>	<b>Distributed Synergy AI Benchmarking</b>	<b>3</b>
<b>2</b>	<b>Quick Start</b>	<b>7</b>
<b>3</b>	<b>What is next</b>	<b>11</b>
<b>4</b>	<b>How to install Ianvs</b>	<b>13</b>
<b>5</b>	<b>How to test algorithms with Ianvs</b>	<b>15</b>
<b>6</b>	<b>Industrial defect detection: the PCB-AoI dataset</b>	<b>23</b>
<b>7</b>	<b>Single task learning: FPN</b>	<b>27</b>
<b>8</b>	<b>Incremental learning: BasicIL-FPN</b>	<b>33</b>
<b>9</b>	<b>How to config algorithm</b>	<b>37</b>
<b>10</b>	<b>How to config testenv</b>	<b>43</b>
<b>11</b>	<b>How to config benchmarkingjob</b>	<b>47</b>
<b>12</b>	<b>How to use Ianvs command line</b>	<b>53</b>
<b>13</b>	<b>Leaderboard of single task learning</b>	<b>55</b>
<b>14</b>	<b>Leaderboard of incremental learning</b>	<b>57</b>
<b>15</b>	<b>Testing single task learning in industrial defect detection</b>	<b>59</b>
<b>16</b>	<b>Testing incremental learning in industrial defect detection</b>	<b>63</b>
<b>17</b>	<b>How to contribute test environments</b>	<b>71</b>
<b>18</b>	<b>How to contribute an algorithm to Ianvs</b>	<b>73</b>
<b>19</b>	<b>How to contribute test reports or leaderboards</b>	<b>75</b>
<b>20</b>	<b>Roadmap</b>	<b>77</b>
<b>21</b>	<b>Ianvs v0.1.0 release</b>	<b>79</b>
<b>22</b>	<b>RELATED LINKs</b>	<b>81</b>



Ianvs is a distributed synergy AI benchmarking project incubated in KubeEdge SIG AI. According to the landing challenge survey 2022 in KubeEdge SIG AI, when it comes to the landing of distributed synergy AI projects, developers suffer from the lack of support on related datasets and algorithms; while end users are lost in the sea of mismatched solutions. That limits the wide application of related techniques and hinders a prosperous ecosystem of distributed synergy AI.

Confronted with these challenges, Ianvs aims to test the performance of distributed synergy AI solutions following recognized standards, in order to facilitate more efficient and effective development. More detailedly, Ianvs prepares not only test cases with datasets and corresponding algorithms, but also benchmarking tools including simulation and hyper-parameter searching. Ianvs also reveals best practices for developers and end users with presentation tools including leaderboards and test reports.

The scope of Ianvs is mainly two folds.

First, Ianvs aims to provide end-to-end benchmark toolkits across devices, edge nodes, and cloud nodes based on typical distributed-synergy AI paradigms and applications.

- Tools to manage test environment. For example, it would be necessary to support the CRUD (Create, Read, Update, and Delete) actions in test environments. Elements of such test environments include algorithm-wise and system-wise configuration.
- Tools to control test cases. Typical examples include paradigm templates, simulation tools, and hyper-parameter-based assistant tools.
- Tools to manage benchmark presentation, e.g., leaderboard and test report generation.

Second, Ianvs also cooperates with other organizations or communities, e.g., in KubeEdge SIG AI, to establish comprehensive benchmarks and developed related applications, which can include but are not limited to

- Dataset collection, re-organization, and publication
- Formalized specifications, e.g., standards
- Holding competitions or coding events, e.g., open source promotion plan
- Maintaining solution leaderboards or certifications for commercial usage

Start your journey on Ianvs with the following links:



## DISTRIBUTED SYNERGY AI BENCHMARKING

Edge computing emerges as a promising technical framework to overcome the challenges in cloud computing. In this machine-learning era, the AI application becomes one of the most critical types of applications on the edge. Driven by the increasing computation power of edge devices and the increasing amount of data generated from the edge, edge-cloud synergy AI and distributed synergy AI techniques have received more and more attention for the sake of device, edge, and cloud intelligence enhancement.

Nevertheless, distributed synergy AI is at its initial stage. For the time being, the comprehensive evaluation standard is not yet available for scenarios with various AI paradigms on all three layers of edge computing systems. According to the landing challenge survey 2022, developers suffer from the lack of support on related datasets and algorithms; while end users are lost in the sea of mismatched solutions. That limits the wide application of related techniques and hinders a prosperous ecosystem of distributed synergy AI. A comprehensive end-to-end distributed synergy AI benchmark suite is thus needed to measure and optimize the systems and applications.

Ianvs thus provides a basic benchmark suite for distributed synergy AI, so that AI developers and end users can benefit from efficient development support and best practice discovery.

### 1.1 Goals

For developers or end users of distributed synergy AI solutions, the goals of the distributed synergy AI framework are:

- Facilitating efficient development for developers by preparing
  - test cases including dataset and corresponding tools
  - benchmarking tools including simulation and hyper-parameter searching
- Revealing best practices for developers and end users
  - presentation tools including leaderboards and test reports

### 1.2 Scope

The distributed synergy AI benchmarking ianvs aims to test the performance of distributed synergy AI solutions following recognized standards, in order to facilitate more efficient and effective development.

The scope of ianvs includes

- Providing end-to-end benchmark toolkits across devices, edge nodes, and cloud nodes based on typical distributed-synergy AI paradigms and applications.
  - Tools to manage test environment. For example, it would be necessary to support the CRUD (Create, Read, Update, and Delete) actions in test environments. Elements of such test environments include algorithm-wise and system-wise configuration.

- Tools to control test cases. Typical examples include paradigm templates, simulation tools, and hyper-parameter-based assistant tools.
- Tools to manage benchmark presentation, e.g., leaderboard and test report generation.
- Cooperation with other organizations or communities, e.g., in KubeEdge SIG AI, to establish comprehensive benchmarks and developed related applications, which can include but are not limited to
  - Dataset collection, re-organization, and publication
  - Formalized specifications, e.g., standards
  - Holding competitions or coding events, e.g., open source promotion plan
  - Maintaining solution leaderboards or certifications for commercial usage

Targeting users

- Developers: Build and publish edge-cloud collaborative AI solutions efficiently from scratch
- End users: view and compare distributed synergy AI capabilities of solutions

The scope of ianvs does NOT include to

- Re-invent existing edge platform, i.e., kubeedge, etc.
- Re-invent existing AI frameworks, i.e., tensorflow, pytorch, mindspore, etc.
- Re-invent existing distributed synergy AI framework, i.e., kubeedge-sedna, etc.
- Re-invent existing UI or GUI toolkits, i.e., prometheus, grafana, matplotlib, etc.

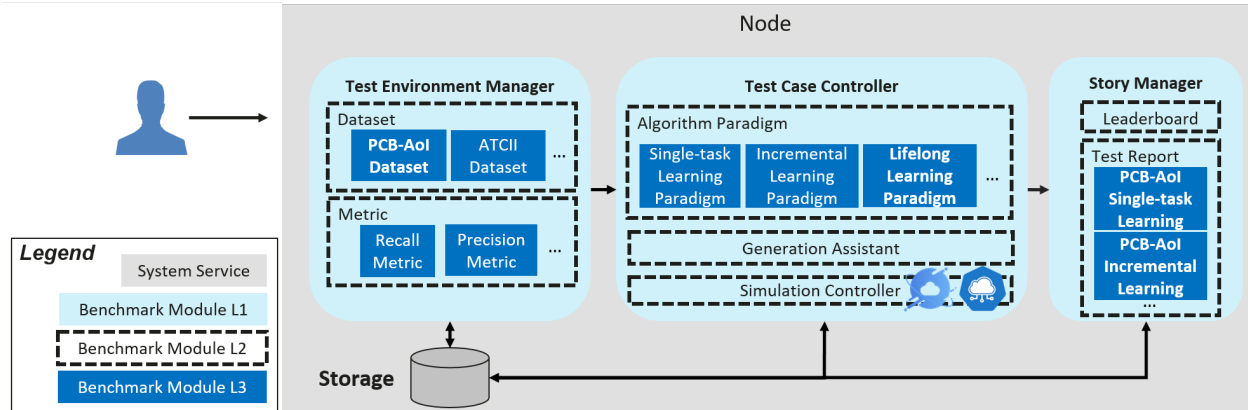
## 1.3 Design Details

### 1.3.1 Architecture and Modules

The architectures and related concepts are shown in the below figure. The ianvs is designed to run within a single node. Critical components include

- **Test Environment Manager:** the CRUD of test environments serving for global usage
- **Test Case Controller:** control the runtime behavior of test cases like instance generation and vanish
  - **Generation Assistant:** assist users to generate test cases based on certain rules or constraints, e.g., the range of parameters
  - **Simulation Controller:** control the simulation process of edge-cloud synergy AI, including the instance generation and vanishment of simulation containers
- **Story Manager:** the output management and presentation of the test case, e.g., leaderboards





Ianvs includes Test-Environment Management, Test-case Controller, and Story Manager in the Distributed Synergy AI benchmarking toolkits, where

1. Test-Environment Manager basically includes

- Algorithm-wise configuration
  - Public datasets
  - Pre-processing algorithms
  - Feature engineering algorithms
  - Post-processing algorithms like metric computation
- System-wise configuration
  - Overall architecture
  - System constraints or budgets
    - \* End-to-end cross-node
    - \* Per node

2. Test-case Controller includes but is not limited to the following components

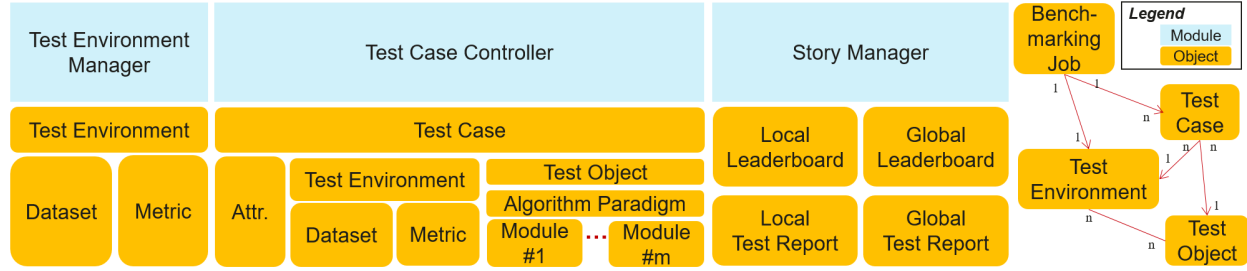
- Templates of common distributed-synergy-AI paradigms, which can help the developer to prepare their test case without too much effort. Such paradigms include edge-cloud synergy joint inference, incremental learning, federated learning, and lifelong learning.
- Simulation tools. Develop simulated test environments for test cases
  - Note that simulation tools are not yet available in early versions until v0.5
  - It is NOT in the scope of this open-sourced Ianvs to simulate different hardware devices, e.g., simulating NPU with GPU and even CPU
- Other tools to assist test-case generation. For instance, prepare test cases based on a given range of hyper-parameters.

3. Story Manager includes but is not limited to the following components

- Leaderboard generation
- Test report generation

### 1.3.2 Definitions of Objects

Quite a few terms exist in ianvs, which include the detailed modules and objects. To facilitate easier concept understanding, we show a hierarchical table of terms in the following figures, where the top item contains the items below it.



The concept definition of modules has been shown in the Architecture Section. In the following, we introduce the concepts of objects for easier understanding.

- **Benchmark:** standardized evaluation process recognized by the academic or industry.
- **Benchmarking Job:** the serving instance for an individual benchmarking with ianvs, which takes charge of the lifetime management of all possible ianvs components.
  - Besides components, a benchmarking job includes instances of a test environment, one or more test cases, a leaderboard, or a test report.
  - Different test environments lead to different benchmarking jobs and leaderboards. A benchmarking job can include multiple test cases
- **Test Object:** the targeted instance under benchmark testing. A typical example would be a particular algorithm or system.
- **Test Environment:** setups or configurations for benchmarking, typically excluding the test object.
  - It can include algorithm-wise and system-wise configurations.
  - It serves as the unique descriptor of a benchmarking job. Different test environments thus lead to different benchmarking jobs.
- **Test Case:** the executable instance to evaluate the performance of the test object under a particular test environment. Thus, the test case is usually generated with a particular test environment and outputs testing results if executed.
  - It is the atomic unit of a benchmark. That is, a benchmarking job can include quite a few test cases.
- **Attribute (Attr.) of Test Case:** Attributes or descriptors of a test case, e.g., id, name, and time stamp.
- **Algorithm Paradigm:** acknowledged AI process which usually includes quite a few modules that can be implemented with replaceable algorithms, e.g., federated learning which includes modules of local train and global aggregation.
- **Algorithm Module:** the component of the algorithm paradigm, e.g., the global aggregation module of the federated learning paradigm.
- **Leaderboard:** the ranking of the test object under a specific test environment.
  - The local node holds the local leaderboard for private usage.
  - The global leaderboard is shared (e.g., via GitHub) by the acknowledged organization.
- **Test Report:** the manuscript recording how the testing is conducted.

## QUICK START

Welcome to Ianvs! Ianvs aims to test the performance of distributed synergy AI solutions following recognized standards, in order to facilitate more efficient and effective development. Quick start helps you to test your algorithm on Ianvs with a simple example of industrial defect detection. You can reduce manual procedures to just a few steps so that you can build and start your distributed synergy AI solution development within minutes.

Before using Ianvs, you might want to have the device ready:

- One machine is all you need, i.e., a laptop or a virtual machine is sufficient and a cluster is not necessary
- 2 CPUs or more
- 4GB+ free memory, depends on algorithm and simulation setting
- 10GB+ free disk space
- Internet connection for GitHub and pip, etc
- Python 3.6+ installed

In this example, we are using the Linux platform with Python 3.6.9. If you are using Windows, most steps should still apply but a few commands and package requirements might be different.

### 2.1 Step 1. ianvs Preparation

First, we download the code of Ianvs. Assuming that we are using `/ianvs` as workspace, Ianvs can be cloned with Git as:

```
mkdir /ianvs
cd /ianvs #One might use another path preferred

mkdir project
cd project
git clone https://github.com/kubeedge/ianvs.git
```

Then, we install third-party dependencies for ianvs.

```
sudo apt-get update
sudo apt-get install libgl1-mesa-glx -y
python -m pip install --upgrade pip

cd ianvs
python -m pip install ./examples/resources/third_party/*
python -m pip install -r requirements.txt
```

We are now ready to install Ianvs.

```
python setup.py install
```

Note: If you want to use a separate space to do work, you may select the following method:

```
python -m pip install --pre envd
envd bootstrap

cd /ianvs/project/ianvs
envd build build.envd
envd up
```

refer to the ML tool [envd](#).

## 2.2 Step 2. Dataset and Model Preparation

Datasets and models can be large. To avoid over-size projects in the GitHub repository of Ianvs, the Ianvs code base does not include origin datasets and models. Then developers do not need to download non-necessary datasets and models for a quick start.

First, the user needs to prepare the dataset according to the targeted scenario, from source links (e.g., from Cloud Service or Kaggle) provided by Ianvs. All scenarios with datasets are available [Links of scenarios](#). As an example in this document, we are using [the PCB-AoI Public Dataset](#) put on Kaggle. The dataset is released by KubeEdge Ianvs and prepared by KubeEdge SIG AI members. See [Details of PCB-AoI dataset](#) for more information.

```
cd /ianvs #One might use another path preferred
mkdir dataset
cd dataset
wget https://kubedge.obs.cn-north-1.myhuaweicloud.com:443/ianvs/pcb-aoi/dataset.zip
unzip dataset.zip
```

The URL address of this dataset then should be filled in the configuration file `testenv.yaml`. In this quick start, we have done that for you and the interested readers can refer to [testenv.yaml](#) for more details.

Then we may Develop the targeted algorithm as usual. In this quick start, Ianvs has prepared an initial model for benchmarking. One can find the model at [FPN-model](#).

```
cd /ianvs #One might use another path preferred
mkdir initial_model
cd initial_model
wget https://kubedge.obs.cn-north-1.myhuaweicloud.com:443/ianvs/pcb-aoi/model.zip
```

Related algorithm is also ready as a wheel in this quick start.

```
cd /ianvs/project/ianvs/
python -m pip install examples/resources/algorithms/FPN_TensorFlow-0.1-py3-none-any.whl
```

The URL address of this algorithm then should be filled in the configuration file `algorithm.yaml`. In this quick start, we have done that for you and the interested readers can refer to the [algorithm.yaml](#) for more details.

## 2.3 Step 3. ianvs Execution and Presentation

We are now ready to run the ianvs for benchmarking on the PCB-AoI dataset.

```
ianvs -f ./examples/pcb-aoi/singletask_learning_bench/benchmarkingjob.yaml
```

Finally, the user can check the result of benchmarking on the console and also in the output path( e.g. `/ianvs/singletask_learning_bench/workspace`) defined in the benchmarking config file ( e.g. `benchmarkingjob.yaml`). In this quick start, we have done all configurations for you and the interested readers can refer to [benchmarkingJob.yaml](#) for more details.

The final output might look like this:

rank	algo- rithm	f1_score	paradigm	base- model	learn- ing_rate	mo- men- tum	time	url
1	fpn_single	0.8396	single-task learning	FPN	0.1	0.5	2022-07-07 20:33:53	/ianvs/pcb-aoi/singletask_learning_bench/workspace/benchmarkingfdf0-11ec-8d5d-fa163eaa99d5
2	fpn_single	0.8353	single-task learning	FPN	0.1	0.95	2022-07-07 20:31:08	/ianvs/pcb-aoi/singletask_learning_bench/workspace/benchmarkingfdf0-11ec-8d5d-fa163eaa99d5

This ends the quick start experiment.



## WHAT IS NEXT

If the reader is ready to explore more on Ianvs, e.g., after the quick start, the following links might help:

[How to test algorithms](#)

[How to contribute algorithms](#)

[How to contribute test environments](#)

[Links of scenarios](#)

[Details of PCB-AoI dataset](#)

If any problems happen, the user can refer to [the issue page on Github](#) for help and are also welcome to raise any new issue.

Enjoy your journey on Ianvs!





## HOW TO INSTALL IANVS

It is recommended to use Ianvs on a Linux machine. But for quick algorithm development, the Windows platform is also planned to support, to reduce the configuration cost of the development environment.

This guide covers how to install Ianvs on a Linux environment.

### 4.1 Prerequisites

- One machine is all you need, i.e., a laptop or a virtual machine is sufficient and a cluster is not necessary
- 2 CPUs or more
- 4GB+ free memory, depends on algorithm and simulation setting
- 10GB+ free disk space
- Internet connection for GitHub and pip, etc
- Python 3.6+ installed

you can check the python version by the following command:

```
python -V
```

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

```
Python 3.6.9
```

### 4.2 Install ianvs on Linux

#### 4.2.1 Create virtualenv

```
sudo apt-get install -y virtualenv
mkdir ~/venv
virtualenv -p python3 ~/venv/ianvs
source ~/venv/ianvs/bin/activate
```

If you prefer conda, you can create a python environment by referring to the [creating steps](#) provided by conda.

### 4.2.2 Download ianvs project

```
cd ~  
git clone https://github.com/kubeedge/ianvs.git
```

### 4.2.3 Install third-party dependencies

```
sudo apt-get update  
sudo apt-get install libgl1-mesa-glx -y  
python -m pip install --upgrade pip  
  
cd ~/ianvs  
python -m pip install ./examples/resources/third_party/*  
python -m pip install -r requirements.txt
```

### 4.2.4 Install ianvs

```
python setup.py install
```

### 4.2.5 Check the installation

```
ianvs -v
```

If the version information is printed, Ianvs is installed successfully.

## 4.3 About Windows

At the time being, the package requirements of Ianvs are only applicable for Linux, to ensure comprehensive support from the Linux ecosystem and to ease the burden of manual installation for users in Windows.

If you are more used to developing on Windows, you can still do so with remote connections like SSH from Windows connecting to a Linux machine with ianvs installed. Such remote connection is already supported in common Python coding tools like VScode, Pycharm, etc. By doing so, it helps to provide efficient installation and robust functionality of Ianvs.

## HOW TO TEST ALGORITHMS WITH IANVS

With Ianvs installed and the related environment prepared, an algorithm developer is then able to test his/her own targeted algorithm using the following steps.

Note that:

- If you are testing an algorithm submitted in the Ianvs repository, e.g., FPN for single task learning, the test environment and the test case are both ready to use and you can directly refer to Quick Start.
- Otherwise, if the user has a test algorithm that is new to the Ianvs repository, i.e., the test environment and the test case are not ready for the targeted algorithm, you might test the algorithm in Ianvs following the next steps from scratch

### 5.1 Step 1. Test Environment Preparation

First, the user needs to prepare the dataset according to the targeted scenario, from source links (e.g., from Kaggle) provided by Ianvs. Scenarios with datasets are available Links of scenarios. As an example in this document, we are using the PCB-AoI Public Dataset released by KubeEdge SIG AI members on Kaggle. See details of PCB-AoI dataset for more information on this dataset.

You might wonder why not put the dataset on the GitHub repository of Ianvs: Datasets can be large. To avoid over-size projects in the GitHub repository of Ianvs, the Ianvs code base does not include origin datasets and developers might want to download unneeded datasets. The URL address of this dataset then should be filled in the configuration file `testenv.yaml`.

The URL address of this dataset then should be filled in the configuration file `testenv.yaml`.

```
# testenv.yaml
testenv:
  # dataset configuration
  dataset:
    # the url address of train dataset index; string type;
    train_url: "/ianvs/dataset/train_data/index.txt"
    # the url address of test dataset index; string type;
    test_url: "/ianvs/dataset/test_data/index.txt"

  # model eval configuration of incremental learning;
  model_eval:
    # metric used for model evaluation
    model_metric:
      # metric name; string type;
      name: "f1_score"
```

(continues on next page)

(continued from previous page)

```

# the url address of python file
url: "./examples/pcb-aoi/incremental_learning_bench/testenv/fl_score.py"

# condition of triggering inference model to update
# threshold of the condition; types are float/int
threshold: 0.01
# operator of the condition; string type;
# values are ">=", ">", "<=", "<" and "=";
operator: ">="

# metrics configuration for test case's evaluation; list type;
metrics:
  # metric name; string type;
  - name: "fl_score"
  # the url address of python file
  url: "./examples/pcb-aoi/incremental_learning_bench/testenv/fl_score.py"
  - name: "samples_transfer_ratio"

# incremental rounds setting for incremental learning paradigm.; int type; default
↪ value is 2;
incremental_rounds: 2

```

The URL address of this test environment, i.e., testenv.yaml, then should be filled in the configuration file in the following Step 3. For example,

```

# benchmarkingJob.yaml
testenv: "./examples/pcb-aoi/benchmarkingjob/testenv/testenv.yaml"

```

## 5.2 Step 2. Test Case Preparation

Note that the tested algorithm should follow the ianvs interface to ensure functional benchmarking. That is, when a new algorithm is needed for testing, it should be extended based on the basic classes, i.e., class\_factory.py. The class factory helps to make the algorithm pluggable in Ianvs and two classes are defined in class\_factory.py, namely ClassType, and ClassFactory. ClassFactory can register the modules you want to reuse through decorators. The user may develop the targeted algorithm, as usual, using the algorithm interface in the class factory. Currently, Ianvs is using the class\_factory.py defined in KubeEdge SIG AI (source link). If you want to contribute a new type of module to KubeEdge SIG AI, i.e., a new class type, please refer to the guide on how to contribute algorithms.

Currently, Ianvs is using the class\_factory.py defined in KubeEdge SIG AI (source link). If you want to contribute a new type of modules to KubeEdge SIG AI, i.e., a new classtype, please refer to the guide of [how to contribute algorithms](#).

### 5.2.1 Example 1. Testing a hard-example-mining algorithm in incremental learning

As the first example, we describe how to test an algorithm Threshold-based-HEM for HEM (Hard Example Mining) module in incremental learning. For this new algorithm in `ClassType.HEM`, the code in the algorithm file is as follows:

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

    def __call__(self, infer_result=None):
        return Threshold-based-HEM(infer_result)
```

With the above algorithm interface, one may develop the targeted algorithm as usual in the same algorithm file:

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

    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
```

### 5.2.2 Example 2. Testing a neural-network-based modeling algorithm in incremental learning

As the second example, we describe how to test a neural network FPN for HEM (Hard Example Mining) module in incremental learning. For this new algorithm in `ClassType.GENERAL`, the code in the algorithm file is as follows:

```
@ClassFactory.register(ClassType.GENERAL, alias="FPN")
class BaseModel:

    def __init__(self, **kwargs):
        """
        initialize logging configuration
        """

        self.has_fast_rcnn_predict = False

        self._init_tf_graph()

        self.temp_dir = tempfile.mkdtemp()
        if not os.path.isdir(self.temp_dir):
            mkdir(self.temp_dir)

        os.environ["MODEL_NAME"] = "model.zip"
```

(continues on next page)

(continued from previous page)

```

cfigs.LR = kwargs.get("learning_rate", 0.0001)
cfigs.MOMENTUM = kwargs.get("momentum", 0.9)
cfigs.MAX_ITERATION = kwargs.get("max_iteration", 5)

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

    if train_data is None or train_data.x is None or train_data.y is None:
        raise Exception("Train data is None.")

    with tf.Graph().as_default():

        img_name_batch, train_data, gtboxes_and_label_batch, num_objects_batch, data_
↪ num = \
            next_batch_for_tasks(
                (train_data.x, train_data.y),
                dataset_name=cfigs.DATASET_NAME,
                batch_size=cfigs.BATCH_SIZE,
                shortside_len=cfigs.SHORT_SIDE_LEN,
                is_training=True,
                save_name="train"
            )

        # ... ..
        # several lines are omitted here.

    return self.checkpoint_path

def save(self, model_path):
    if not model_path:
        raise Exception("model path is None.")

    model_dir, model_name = os.path.split(self.checkpoint_path)
    models = [model for model in os.listdir(model_dir) if model_name in model]

    if os.path.splitext(model_path)[-1] != ".zip":
        model_path = os.path.join(model_path, "model.zip")

    if not os.path.isdir(os.path.dirname(model_path)):
        os.makedirs(os.path.dirname(model_path))

    with zipfile.ZipFile(model_path, "w") as f:
        for model_file in models:
            model_file_path = os.path.join(model_dir, model_file)
            f.write(model_file_path, model_file, compress_type=zipfile.ZIP_DEFLATED)

    return model_path

def predict(self, data, input_shape=None, **kwargs):
    if data is None:
        raise Exception("Predict data is None")

    inference_output_dir = os.getenv("RESULT_SAVED_URL")

```

(continues on next page)

(continued from previous page)

```

with self.tf_graph.as_default():
    if not self.has_fast_rcnn_predict:
        self._fast_rcnn_predict()
        self.has_fast_rcnn_predict = True

    restorer = self._get_restorer()

    config = tf.ConfigProto()
    init_op = tf.group(
        tf.global_variables_initializer(),
        tf.local_variables_initializer()
    )

    with tf.Session(config=config) as sess:
        sess.run(init_op)

    # ... ..
    # several lines are omitted here.

    return predict_dict

def load(self, model_url=None):
    if model_url:
        model_dir = os.path.split(model_url)[0]
        with zipfile.ZipFile(model_url, "r") as f:
            f.extractall(path=model_dir)
            ckpt_name = os.path.basename(f.namelist()[0])
            index = ckpt_name.find("ckpt")
            ckpt_name = ckpt_name[:index + 4]
            self.checkpoint_path = os.path.join(model_dir, ckpt_name)

    else:
        raise Exception(f"model url is None")

    return self.checkpoint_path

def evaluate(self, data, model_path, **kwargs):
    if data is None or data.x is None or data.y is None:
        raise Exception("Prediction data is None")

    self.load(model_path)
    predict_dict = self.predict(data.x)
    metric_name, metric_func = kwargs.get("metric")
    if callable(metric_func):
        return {"f1_score": metric_func(data.y, predict_dict)}
    else:
        raise Exception(f"not found model metric func(name={metric_name}) in model_
↪eval phase")

```

With the above algorithm interface, one may develop the targeted algorithm of FPN as usual in the same algorithm file. The FPN\_TensorFlow is also open sourced. For those interested in FPN\_TensorFlow, an example implementation is available [here](#) and extended with the algorithm interface [here](#).

Then we can fill in the `algorithm.yaml`:

```
algorithm:
  # paradigm type; string type;
  # currently the options of value are as follows:
  #   1> "singletasklearning"
  #   2> "incrementallearning"
  paradigm_type: "incrementallearning"
  incremental_learning_data_setting:
    # ratio of training dataset; float type;
    # the default value is 0.8.
    train_ratio: 0.8
    # the method of splitting dataset; string type; optional;
    # currently the options of value are as follows:
    #   1> "default": the dataset is evenly divided based train_ratio;
    splitting_method: "default"
  # the url address of initial model for model pre-training; string url;
  initial_model_url: "/ianvs/initial_model/model.zip"

  # algorithm module configuration in the paradigm; list type;
  modules:
    # type of algorithm module; string type;
    # currently the options of value are as follows:
    #   1> "basemodel": contains important interfaces such as train, eval, predict and
    ↪ more; required module;
    - type: "basemodel"
      # name of python module; string type;
      # example: basemodel.py has BaseModel module that the alias is "FPN" for this
    ↪ benchmarking;
      name: "FPN"
      # the url address of python module; string type;
      url: "./examples/pcb-aoi/incremental_learning_bench/testalgorithms/fpn/basemodel.py"
    ↪ ""

    # hyperparameters configuration for the python module; list type;
    hyperparameters:
      # name of the hyperparameter; string type;
      - momentum:
          # values of the hyperparameter; list type;
          # types of the value are string/int/float/boolean/list/dictionary
          values:
            - 0.95
            - 0.5
      - learning_rate:
          values:
            - 0.1
      # 2> "hard_example_mining": check hard example when predict ; optional module;
      - type: "hard_example_mining"
        # name of python module; string type;
        name: "IBT"
        # the url address of python module; string type;
        url: "./examples/pcb-aoi/incremental_learning_bench/testalgorithms/fpn/hard_
    ↪ example_mining.py"
```

(continues on next page)



(continued from previous page)

```

# hyperparameters configuration for the python module; list type;
hyperparameters:
  # name of the hyperparameter; string type;
  # threshold of image; value is [0, 1]
  - threshold_img:
      values:
        - 0.9
  # predict box of image; value is [0, 1]
  - threshold_box:
      values:
        - 0.9

```

The URL address of this algorithm then should be filled in the configuration file of benchmarkingJob.yaml in the following Step 3. Two examples are as follows:

```

# the configuration of test object
test_object:
  # test type; string type;
  # currently the option of value is "algorithms", the others will be added in succession.
  type: "algorithms"
  # test algorithm configuration files; list type;
  algorithms:
    # algorithm name; string type;
    - name: "fpn_incremental_learning"
      # the url address of test algorithm configuration file; string type;
      # the file format supports yaml/yml
      url: "./examples/pcb-aoi/incremental_learning_bench/testalgorithms/fpn/fpn_
↪algorithm.yaml"

```

or

```

# the configuration of test object
test_object:
  # test type; string type;
  # currently the option of value is "algorithms", the others will be added in succession.
  type: "algorithms"
  # test algorithm configuration files; list type;
  algorithms:
    # algorithm name; string type;
    - name: "fpn_singletask_learning"
      # the url address of test algorithm configuration file; string type;
      # the file format supports yaml/yml;
      url: "./examples/pcb-aoi/singletask_learning_bench/testalgorithms/fpn/fpn_
↪algorithm.yaml"

```

## 5.3 Step 3. ianvs Configuration

Now we come to the final configuration on `benchmarkingJob.yaml` before running ianvs.

First, the user can configure the workspace to reserve the output of tests.

```
# benchmarkingJob.yaml
workspace: "/ianvs/pcb-aoi/workspace/"
```

Then, the user fill in the test environment and algorithm configured in previous steps.

```
# benchmarkingJob.yaml
testenv: ".examples/pcb-aoi/benchmarkingjob/testenv/testenv.yaml"
```

```
algorithms:
- name: "fpm_incremental_learning"
  url: "./examples/pcb-aoi/benchmarkingjob/testalgorithms/fpm_incremental_learning/fpm_
  ↪algorithm.yaml"
```

As the final leaderboard, the user can configure how to rank the leaderboard with the specific metric and order.

```
# benchmarkingJob.yaml
rank:
  sort_by: [ { "f1_score": "descend" } ]
```

There are quite a few possible data items in the leaderboard. Not all of them can be shown simultaneously on the screen. In the leaderboard, we provide the `selected_only` mode for the user to configure what is shown or is not shown. The user can add his/her interested data items in terms of `paradigms`, `modules`, `hyperparameters`, and `metrics` so that the selected columns will be shown.

```
visualization:
  mode: "selected_only"
  method: "print_table"

selected_dataitem:
  paradigms: [ "all" ]
  modules: [ "all" ]
  hyperparameters: [ "all" ]
  metrics: [ "f1_score" ]

save_mode: "selected_and_all"
```

## 5.4 Step 4. Execution and Presentation

Finally, the user can run ianvs for benchmarking.

The benchmarking result of the targeted algorithms will be shown after the evaluation is done. Leaderboard examples can be found [here](#).

## INDUSTRIAL DEFECT DETECTION: THE PCB-AOI DATASET

Download link: [Kaggle](#), [Huawei OBS](#)

### 6.1 Authors

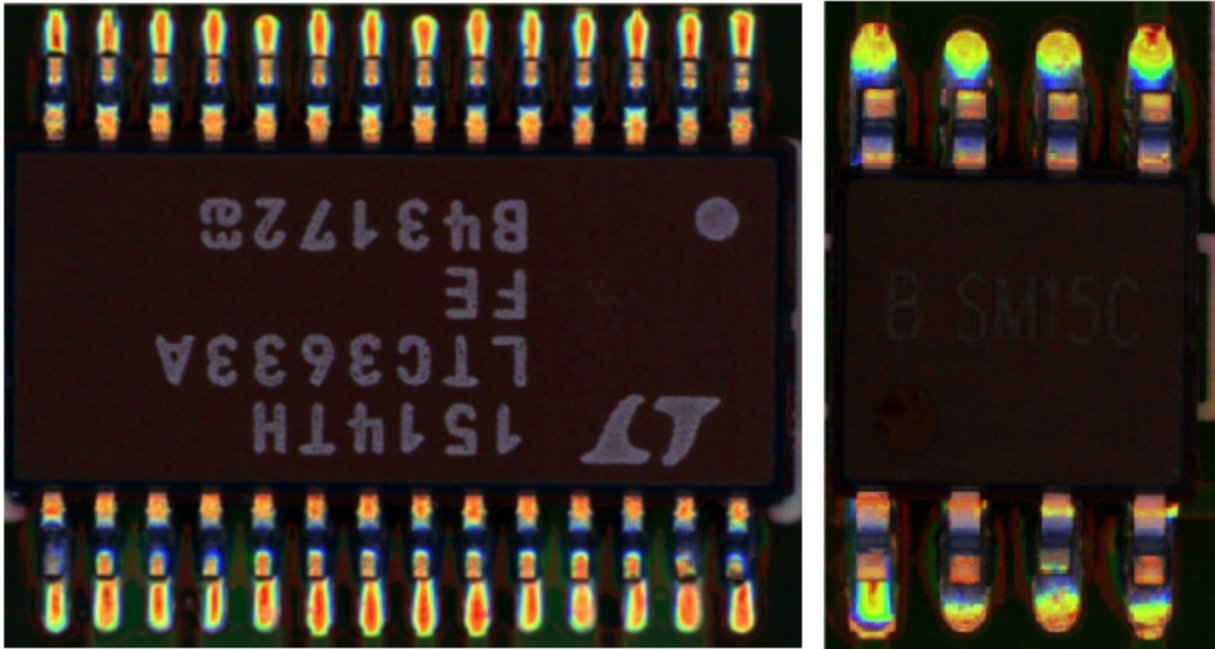
- China Telcom Research Institute: Dongdong Li, Dan Liu, Yun Shen, Yaqi Song
- Raisecom Technology Co.,ltd.: Liangliang Luo

### 6.2 Background

Surface-mount technology (SMT) is a technology that automates electronic circuits production in which components are mounted or placed onto the surface of printed circuit boards. Solder paste printing (SPP) is the most delicate stage in SMT. It prints solder paste on the pads of an electronic circuit panel. Thus, SPP is followed by a solder paste inspection (SPI) stage to detect defects. SPI scans the printed circuit board for missing/less paste, bridging between pads, miss alignments, and so forth. Boards with anomaly must be detected, and boards in good condition should not be disposed of. Thus SPI requires high precision and a high recall.

The PCB-AoI dataset is a part of the open-source distributed synergy AI benchmarking project KubeEdge-Ianvs. Ianvs is honored to be the First site that this dataset is released and the Ianvs working group put it on Kaggle as [The PCB-AoI public dataset](#). It is released by KubeEdge SIG AI members from China Telecom and Raisecom Technology.

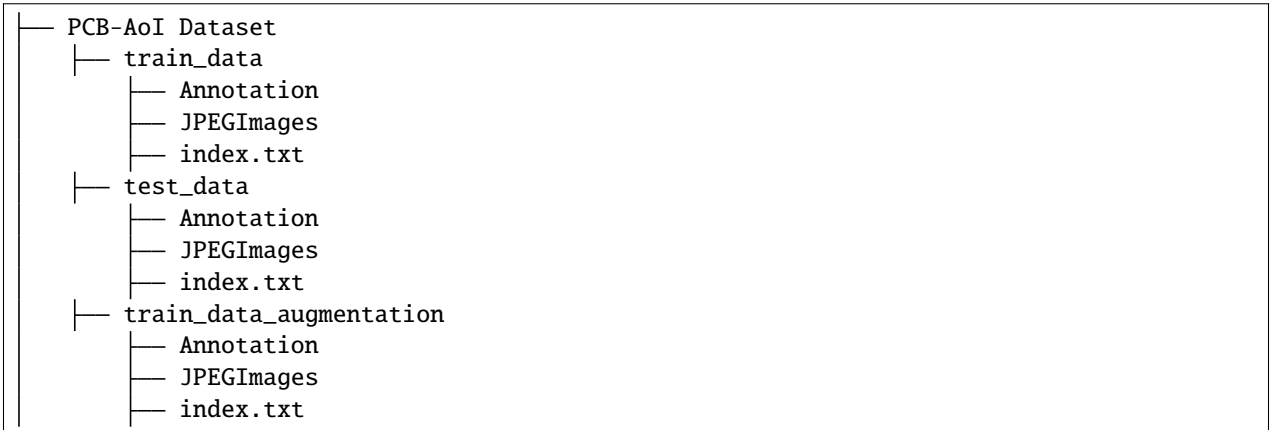
Below shows two example figures in the dataset.



## 6.3 Data Explorer

In this dataset, more than 230 boards are collected and the number of images is enhanced to more than 1200. Detailedly, the dataset include two parts, i.e., the train and the test set. The train set includes 173 boards while the test set includes 60 boards. That is, the train-test ratio is around 3:1 in terms of PCB boards. Data augmentation is conducted, boosting the train-test ratio to 1211:60 (about 20:1) in term of images. Both directories of train\_data and test\_data include the index file which recodes the mapping between the raw images and the label of annotation.

The directories of this dataset is as follows:



The following is part of index.txt:

```

./JPEGImages/20161019-SPI-AOI-1.jpeg ./Annotations/20161019-SPI-AOI-1.xml
./JPEGImages/20161020-SPI-AOI-5.jpeg ./Annotations/20161020-SPI-AOI-5.xml
./JPEGImages/20161021-SPI-AOI-13.jpeg ./Annotations/20161021-SPI-AOI-13.xml

```

(continues on next page)

(continued from previous page)

./JPEGImages/20161021-SPI-AOI-14.jpeg	./Annotations/20161021-SPI-AOI-14.xml
./JPEGImages/20161021-SPI-AOI-15.jpeg	./Annotations/20161021-SPI-AOI-15.xml

Column 1 stands for the file path of the raw image, and column 2 is the file path of corresponding annotation file. In this dataset, the xml annotation follows Pascal VOC XML format. you can find more description of Pascal VOC XML at [here](#).



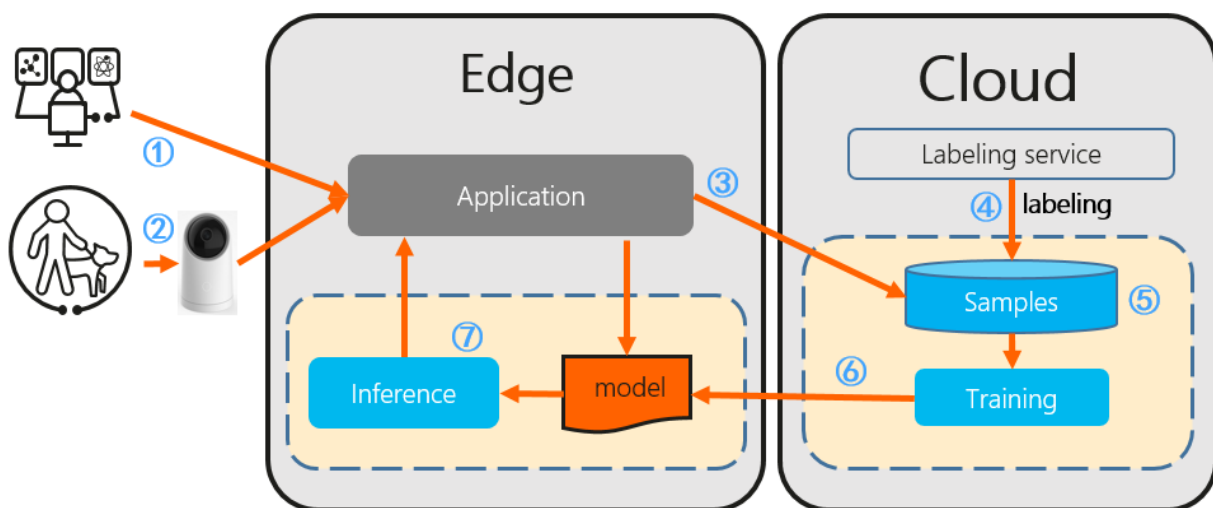
## SINGLE TASK LEARNING: FPN

Pre-trained model: [Huawei OBS](#)

Single task learning is a traditional learning pooling all data together to train a single model. It typically includes a specialist model laser-focused on a single task and requires large amounts of task-specific labeled data, which is not always available on early stage of a distributed synergy AI project.

As shown in the following figure, the single task learning works as procedures below:

1. Developer implements and deploys the application based on single task learning.
2. The application runs and launches single task learning.
3. The application uploads samples to the cloud.
4. Labeling service labels the uploaded samples.
5. Training learns the samples to generate a new model.
6. The system updates the model on the edge.
7. The model conducts inference given test samples where the inference result is send to the application which ends the process.



As for the base model of single task learning, in this report we are using FPN\_TensorFlow. It is a tensorflow re-implementation of Feature Pyramid Networks for Object Detection, which is based on Faster-RCNN. More detailedly, feature pyramids are a basic component in recognition systems for detecting objects at different scales. But recent deep learning object detectors have avoided pyramid representations, in part because they are compute and memory intensive. Researchers have exploited the inherent multi-scale, pyramidal hierarchy of deep convolutional networks to

construct feature pyramids with marginal extra cost. A top-down architecture with lateral connections is developed for building high-level semantic feature maps at all scales. The architecture, called a Feature Pyramid Network (FPN), shows significant improvement as a generic feature extractor in several applications. Using FPN in a basic Faster R-CNN system, the method achieves state-of-the-art single-model results on the COCO detection benchmark without bells and whistles, surpassing all existing single-task entries including those from the COCO 2016 challenge winners. In addition, FPN can run at 5 FPS on a GPU and thus is a practical and accurate solution to multi-scale object detection.

The FPN\_TensorFlow is also open sourced and completed by YangXue and YangJirui. For those interested in details of FPN\_TensorFlow, an example implementation is available [here](#) and is extended with the Ianvs algorithm interface [here](#).

## 7.1 Implementation

Here we also show how to implement a single task learning algorithm for testing in ianvs, based on an opensource algorithm [FPN](#).

When testing your own algorithm, of course, FPN is not necessary. It can be replaced with any algorithm complying the requirement of ianvs interface.

Ianvs testing algorithm development, at present, are using Sedna Lib. The following is recommended development workflow:

1. Algorithm Development: put the algorithm implementation to ianvs [examples directory](#) locally, for testing.
2. Algorithm Submission: submit the algorithm implementation to [Sedna repository](#), for sharing, then everyone can test and use your algorithm.

## 7.2 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 **single task learning algorithm**, you only need to add a line of `ClassFactory.register(ClassType.GENERAL)` to complete the registration.

The following code is just to show the overall structure of a basicIL-fpn BaseModel, not the complete version. The complete code can be found [here](#).

```
@ClassFactory.register(ClassType.GENERAL, alias="FPN")
class BaseModel:

    def __init__(self, **kwargs):
        """
        initialize logging configuration
        """

        self.has_fast_rcnn_predict = False

        self._init_tf_graph()

        self.temp_dir = tempfile.mkdtemp()
```

(continues on next page)



(continued from previous page)

```

    if not os.path.isdir(self.temp_dir):
        mkdir(self.temp_dir)

    os.environ["MODEL_NAME"] = "model.zip"
    cfgs.LR = kwargs.get("learning_rate", 0.0001)
    cfgs.MOMENTUM = kwargs.get("momentum", 0.9)
    cfgs.MAX_ITERATION = kwargs.get("max_iteration", 5)

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

        if train_data is None or train_data.x is None or train_data.y is None:
            raise Exception("Train data is None.")

        with tf.Graph().as_default():

            img_name_batch, train_data, gtboxes_and_label_batch, num_objects_batch, data_
↪ num = \
                next_batch_for_tasks(
                    (train_data.x, train_data.y),
                    dataset_name=cfgs.DATASET_NAME,
                    batch_size=cfgs.BATCH_SIZE,
                    shortside_len=cfgs.SHORT_SIDE_LEN,
                    is_training=True,
                    save_name="train"
                )

            # ...
            # several lines are omitted here.

        return self.checkpoint_path

    def save(self, model_path):
        if not model_path:
            raise Exception("model path is None.")

        model_dir, model_name = os.path.split(self.checkpoint_path)
        models = [model for model in os.listdir(model_dir) if model_name in model]

        if os.path.splitext(model_path)[-1] != ".zip":
            model_path = os.path.join(model_path, "model.zip")

        if not os.path.isdir(os.path.dirname(model_path)):
            os.makedirs(os.path.dirname(model_path))

        with zipfile.ZipFile(model_path, "w") as f:
            for model_file in models:
                model_file_path = os.path.join(model_dir, model_file)
                f.write(model_file_path, model_file, compress_type=zipfile.ZIP_DEFLATED)

        return model_path

    def predict(self, data, input_shape=None, **kwargs):

```

(continues on next page)

(continued from previous page)

```

    if data is None:
        raise Exception("Predict data is None")

    inference_output_dir = os.getenv("RESULT_SAVED_URL")

    with self.tf_graph.as_default():
        if not self.has_fast_rcnn_predict:
            self._fast_rcnn_predict()
            self.has_fast_rcnn_predict = True

        restorer = self._get_restorer()

        config = tf.ConfigProto()
        init_op = tf.group(
            tf.global_variables_initializer(),
            tf.local_variables_initializer()
        )

        with tf.Session(config=config) as sess:
            sess.run(init_op)

        # ...
        # several lines are omitted here.

    return predict_dict

def load(self, model_url=None):
    if model_url:
        model_dir = os.path.split(model_url)[0]
        with zipfile.ZipFile(model_url, "r") as f:
            f.extractall(path=model_dir)
            ckpt_name = os.path.basename(f.namelist()[0])
            index = ckpt_name.find("ckpt")
            ckpt_name = ckpt_name[:index + 4]
            self.checkpoint_path = os.path.join(model_dir, ckpt_name)

    else:
        raise Exception(f"model url is None")

    return self.checkpoint_path

def evaluate(self, data, model_path, **kwargs):
    if data is None or data.x is None or data.y is None:
        raise Exception("Prediction data is None")

    self.load(model_path)
    predict_dict = self.predict(data.x)
    metric_name, metric_func = kwargs.get("metric")
    if callable(metric_func):
        return {"f1_score": metric_func(data.y, predict_dict)}
    else:
        raise Exception(f"not found model metric func(name={metric_name}) in model_

```

(continues on next page)

(continued from previous page)

```
↪eval phase")
```

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



## INCREMENTAL LEARNING: BASICIL-FPN

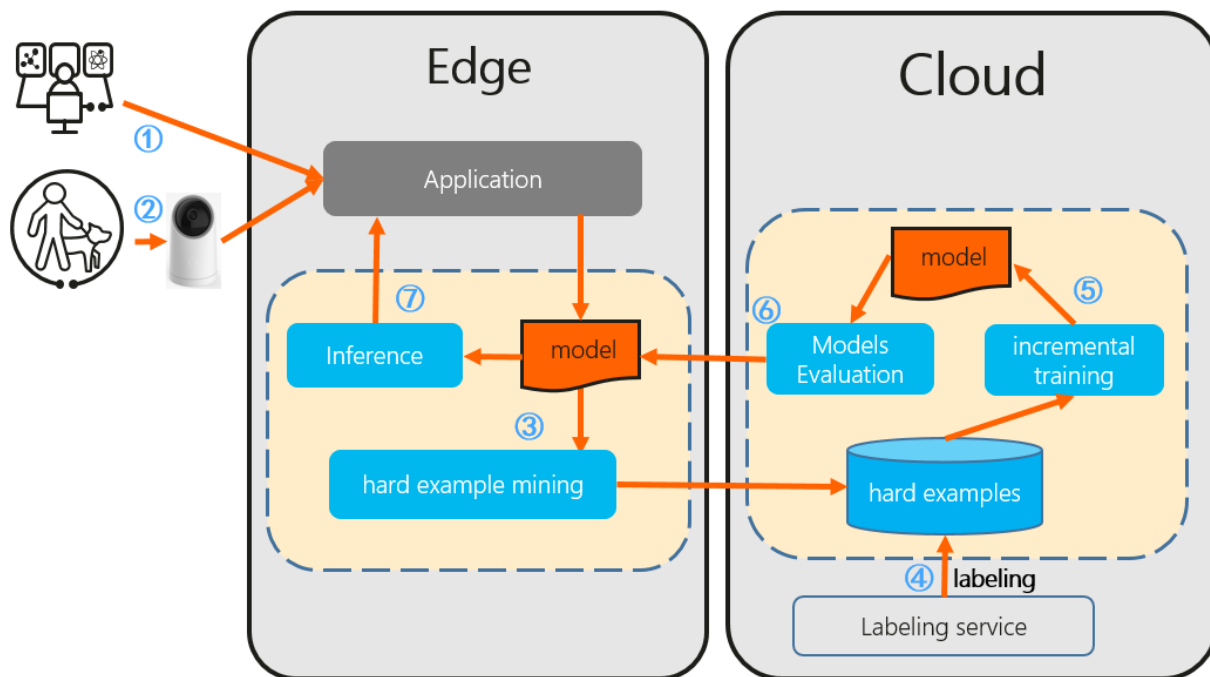
Initial model: Huawei OBS

Traditionally, the data is collected manually and periodically retrained on the cloud to improve the model effect. However, data is continuously generated on the edge side. Traditional 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.

Its goals include:

- 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 example discovering of unlabeled data, for reducing the manual labeling workload.



As shown in the above figure, the incremental learning works as following procedures:

1. Developer implements and deploys the application based on incremental learning.

2. The application runs and launches incremental learning. It can also return the inference result to the application.
3. The system detects hard examples and uploads hard examples to the cloud.
4. Labeling service labels the hard examples.
5. Incremental training online learns the hard examples to generate a new model.
6. Model evaluation is conducted and updates the model if qualified.
7. The model outputs the inference result given test samples and continue as Step 3.

## 8.1 Implementation

Here we will show how to implement a single task learning algorithm for testing in ianvs, based on an opensource algorithm [FPN](#).

For test of your own algorithm, the base model of FPN is not necessary: It can be replaced with any algorithm complying the requirement of ianvs interface.

Ianvs testing algorithm development, at present, are using Sedna Lib. The following is recommended development workflow:

1. Algorithm Development: put the algorithm implementation to ianvs [examples directory](#) locally, for testing.
2. Algorithm Submission: submit the algorithm implementation to [Sedna repository](#), for sharing, then everyone can test and use your 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 **single task learning algorithm**, you only need to add a line of `ClassFactory.register(ClassType.GENERAL)` to complete the registration.

The following code is just to show the overall structure of a basicIL-fpn `BaseModel`, not the complete version. The complete code can be found [here](#).

```
@ClassFactory.register(ClassType.GENERAL, alias="FPN")
class BaseModel:

    def __init__(self, **kwargs):
        """
        initialize logging configuration
        """

        self.has_fast_rcnn_predict = False

        self._init_tf_graph()

        self.temp_dir = tempfile.mkdtemp()
        if not os.path.isdir(self.temp_dir):
            mkdir(self.temp_dir)

        os.environ["MODEL_NAME"] = "model.zip"
        cfgs.LR = kwargs.get("learning_rate", 0.0001)
```

(continues on next page)

(continued from previous page)

```

        cfgs.MOMENTUM = kwargs.get("momentum", 0.9)
        cfgs.MAX_ITERATION = kwargs.get("max_iteration", 5)

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

        if train_data is None or train_data.x is None or train_data.y is None:
            raise Exception("Train data is None.")

        with tf.Graph().as_default():

            img_name_batch, train_data, gtboxes_and_label_batch, num_objects_batch, data_
↪ num = \
                next_batch_for_tasks(
                    (train_data.x, train_data.y),
                    dataset_name=cfgs.DATASET_NAME,
                    batch_size=cfgs.BATCH_SIZE,
                    shortside_len=cfgs.SHORT_SIDE_LEN,
                    is_training=True,
                    save_name="train"
                )

            # ... ...
            # several lines are omitted here.

        return self.checkpoint_path

    def save(self, model_path):
        if not model_path:
            raise Exception("model path is None.")

        model_dir, model_name = os.path.split(self.checkpoint_path)
        models = [model for model in os.listdir(model_dir) if model_name in model]

        if os.path.splitext(model_path)[-1] != ".zip":
            model_path = os.path.join(model_path, "model.zip")

        if not os.path.isdir(os.path.dirname(model_path)):
            os.makedirs(os.path.dirname(model_path))

        with zipfile.ZipFile(model_path, "w") as f:
            for model_file in models:
                model_file_path = os.path.join(model_dir, model_file)
                f.write(model_file_path, model_file, compress_type=zipfile.ZIP_DEFLATED)

        return model_path

    def predict(self, data, input_shape=None, **kwargs):
        if data is None:
            raise Exception("Predict data is None")

        inference_output_dir = os.getenv("RESULT_SAVED_URL")

```

(continues on next page)

(continued from previous page)

```

with self.tf_graph.as_default():
    if not self.has_fast_rcnn_predict:
        self._fast_rcnn_predict()
        self.has_fast_rcnn_predict = True

    restorer = self._get_restorer()

    config = tf.ConfigProto()
    init_op = tf.group(
        tf.global_variables_initializer(),
        tf.local_variables_initializer()
    )

    with tf.Session(config=config) as sess:
        sess.run(init_op)

    # ... ..
    # several lines are omitted here.

    return predict_dict

def load(self, model_url=None):
    if model_url:
        model_dir = os.path.split(model_url)[0]
        with zipfile.ZipFile(model_url, "r") as f:
            f.extractall(path=model_dir)
            ckpt_name = os.path.basename(f.namelist()[0])
            index = ckpt_name.find("ckpt")
            ckpt_name = ckpt_name[:index + 4]
            self.checkpoint_path = os.path.join(model_dir, ckpt_name)

    else:
        raise Exception(f"model url is None")

    return self.checkpoint_path

def evaluate(self, data, model_path, **kwargs):
    if data is None or data.x is None or data.y is None:
        raise Exception("Prediction data is None")

    self.load(model_path)
    predict_dict = self.predict(data.x)
    metric_name, metric_func = kwargs.get("metric")
    if callable(metric_func):
        return {"f1_score": metric_func(data.y, predict_dict)}
    else:
        raise Exception(f"not found model metric func(name={metric_name}) in model_
↪eval phase")

```

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



## HOW TO CONFIG ALGORITHM

The algorithm developer is able to test his/her own targeted algorithm and configs the algorithm using the following configuration.

### 9.1 The configuration of algorithm

Property	Re-quired	Description
paradigm_type	yes	Paradigm name; Type: string; Value Constraint: Currently the options of value are as follows: 1> singletasklearning 2> incrementallearning
inremen-tal_learning_data_setting	no	Data setting for incremental learning paradigm.the configuration of incremental_learning_data_setting
initial_model_url	no	The url address of initial model for model pre-training; Type: string
modules	yes	The algorithm modules for paradigm; Type: list; Value Constraint: the list of the configuration of module

For example:

```
algorithm:
# paradigm type; string type;
# currently the options of value are as follows:
# 1> "singletasklearning"
# 2> "incrementallearning"
paradigm_type: "incrementallearning"
incremental_learning_data_setting:
...
# the url address of initial model for model pre-training; string url;
initial_model_url: "/ianvs/initial_model/model.zip"

# algorithm module configuration in the paradigm; list type;
modules:
...
```

### 9.1.1 The configuration of incremental\_learning\_data\_setting

Property	Required	Description
train_ratio	no	Ratio of training dataset; Type: float; Default value: 0.8; Value Constraint: the value is greater than 0 and less than 1.
splitting_method	no	The method of splitting dataset; Type: string; Default value: default; Value Constraint: Currently the options of value are as follows: 1> default: the dataset is evenly divided based train_ratio.

For example:

```
incremental_learning_data_setting:
# ratio of training dataset; float type;
# the default value is 0.8.
train_ratio: 0.8
# the method of splitting dataset; string type; optional;
# currently the options of value are as follows:
# 1> "default": the dataset is evenly divided based train_ratio;
splitting_method: "default"
```

### 9.1.2 The configuration of module

Property	Required	Description
type	yes	Algorithm module type; Type: string; Value Constraint: Currently the options of value are as follows: 1> basemodel: the algorithm module contains important interfaces such as train, eval, predict and more.it's required module. 2> hard_example_mining: the module checks hard example when predict. it's optional module and often used for incremental learning paradigm.
name	yes	Algorithm module name; Type: string; Value Constraint: a python module name
url	yes	The url address of python module file; Type: string
hyperparameters	no	the configuration of hyperparameters

For example:

```
# algorithm module configuration in the paradigm; list type;
modules:
# type of algorithm module; string type;
# currently the options of value are as follows:
# 1> "basemodel": contains important interfaces such as train eval predict and more;
↪required module;
- type: "basemodel"
  # name of python module; string type;
  # example: basemodel.py has BaseModel module that the alias is "FPN" for this;
↪benchmarking;
```

(continues on next page)

(continued from previous page)

```

name: "FPN"
# the url address of python module; string type;
url: "./examples/pcb-aoi/incremental_learning_bench/testalgorithms/fpn/basemodel.py"

# hyperparameters configuration for the python module; list type;
hyperparameters:
    ...
    # 2> "hard_example_mining": check hard example when predict ; optional module;
- type: "hard_example_mining"
# name of python module; string type;
name: "IBT"
# the url address of python module; string type;
url: "./examples/pcb-aoi/incremental_learning_bench/testalgorithms/fpn/hard_example_
↪mining.py"
# hyperparameters configuration for the python module; list type;
hyperparameters:
    ...

```

### 9.1.3 The configuration of hyperparameters

The following is an example of hyperparameters configuration:

```

# hyperparameters configuration for the python module; list type;
hyperparameters:
# name of the hyperparameter; string type;
- momentum:
# values of the hyperparameter; list type;
# types of the value are string/int/float/boolean/list/dictionary
values:
- 0.95
- 0.5
- learning_rate:
values:
- 0.1
- 0.2

```

Ianvs will test for all the hyperparameter combination, that means it will run all the following 4 test:

Num	learning_rate	momentum
1	0.1	0.95
2	0.1	0.5
3	0.2	0.95
4	0.2	0.5

Currently, Ianvs is not restricted to validity of the hyperparameter combination. That might lead to some invalid parameter combination, and it is controlled by the user himself. In the further version of Ianvs, it will support excluding invalid parameter combinations to improve efficiency.

## 9.2 Show example

```
# fpn_algorithm.yaml
algorithm:
  # paradigm type; string type;
  # currently the options of value are as follows:
  # 1> "singletasklearning"
  # 2> "incrementallearning"
  paradigm_type: "incrementallearning"
  incremental_learning_data_setting:
    # ratio of training dataset; float type;
    # the default value is 0.8.
    train_ratio: 0.8
    # the method of splitting dataset; string type; optional;
    # currently the options of value are as follows:
    # 1> "default": the dataset is evenly divided based train_ratio;
    splitting_method: "default"
  # the url address of initial model for model pre-training; string url;
  initial_model_url: "/ianvs/initial_model/model.zip"

  # algorithm module configuration in the paradigm; list type;
  modules:
    # type of algorithm module; string type;
    # currently the options of value are as follows:
    # 1> "basemodel": contains important interfaces such as train eval predict and
    ↪ more; required module;
    - type: "basemodel"
      # name of python module; string type;
      # example: basemodel.py has BaseModel module that the alias is "FPN" for this
    ↪ benchmarking;
      name: "FPN"
      # the url address of python module; string type;
      url: "./examples/pcb-aoi/incremental_learning_bench/testalgorithms/fpn/basemodel.py"
    ↪ ""

    # hyperparameters configuration for the python module; list type;
    hyperparameters:
      # name of the hyperparameter; string type;
      - momentum:
          # values of the hyperparameter; list type;
          # types of the value are string/int/float/boolean/list/dictionary
          values:
            - 0.95
            - 0.5
      - learning_rate:
          values:
            - 0.1
      # 2> "hard_example_mining": check hard example when predict ; optional module;
      - type: "hard_example_mining"
        # name of python module; string type;
        name: "IBT"
        # the url address of python module; string type;
```

(continues on next page)

(continued from previous page)

```
url: "./examples/pcb-aoi/incremental_learning_bench/testalgorithms/fpn/hard_
↪example_mining.py"
# hyperparameters configuration for the python module; list type;
hyperparameters:
  # name of the hyperparameter; string type;
  # threshold of image; value is [0, 1]
  - threshold_img:
    values:
      - 0.9
  # predict box of image; value is [0, 1]
  - threshold_box:
    values:
      - 0.9
```



## HOW TO CONFIG TESTENV

The algorithm developer is able to test his/her own targeted algorithm, he/she should prepare the test environment. how to config test environment, please to refer to the following configuration information.

### 10.1 The configuration of testenv

Property	Re-quired	Description
dataset	yes	The configuration of dataset
model_eval	no	The configuration of model_eval
metrics	yes	The metrics used for test case's evaluation; Type: list; Value Constraint: the list of the configuration of metric.
incremental_rounds	no	Incremental rounds setting for incremental learning paradigm; Type: int; Default value: 2; Value Constraint: the value must be not less than 2.

For example:

```
testenv:
# dataset configuration
dataset:
...
# model eval configuration of incremental learning;
model_eval:
...
# metrics configuration for test case's evaluation; list type;
metrics:
...
# incremental rounds setting for incremental learning paradigm; int type; default
↪value is 2;
# the value must be not less than 2;
incremental_rounds: 2
```

### 10.1.1 The configuration of dataset

Property	Required	Description
train_url	yes	The url address of train dataset index; Type: string
test_url	yes	The url address of test dataset index; Type: string

For example:

```
# dataset configuration
dataset:
  # the url address of train dataset index; string type;
  train_url: "/ianvs/dataset/train_data/index.txt"
  # the url address of test dataset index; string type;
  test_url: "/ianvs/dataset/test_data/index.txt"
```

### 10.1.2 The configuration of model\_eval

Property	Required	Description
model_metric	yes	The Metric used for model evaluation; <a href="#">The configuration of metric</a> .
threshold	yes	Threshold of condition for triggering inference model to update; Type: float/int
operator	yes	Operator of condition for triggering inference model to update; Type: string; Value Constraint: the values are ">=", ">", "<=", "<" and "=".

For example:

```
# model eval configuration of incremental learning;
model_eval:
  # metric used for model evaluation
  model_metric:
    ...
  # condition of triggering inference model to update
  # threshold of the condition; types are float/int
  threshold: 0.01
  # operator of the condition; string type;
  # values are ">=", ">", "<=", "<" and "=";
  operator: ">="
```

### 10.1.3 The configuration of metric

Property	Required	Description
name	yes	Metric name; Type: string; Value Constraint: a python module name
url	no	The url address of python module file; Type: string.

For example:



```
# metric used for model evaluation
model_metric:
  # metric name; string type;
  name: "f1_score"
  # the url address of python file
  url: "./examples/pcb-aoi/incremental_learning_bench/testenv/f1_score.py"
```

## 10.2 Show example

```
# testenv.yaml
testenv:
  # dataset configuration
  dataset:
    # the url address of train dataset index; string type;
    train_url: "/ianvs/dataset/train_data/index.txt"
    # the url address of test dataset index; string type;
    test_url: "/ianvs/dataset/test_data/index.txt"

  # model eval configuration of incremental learning;
  model_eval:
    # metric used for model evaluation
    model_metric:
      # metric name; string type;
      name: "f1_score"
      # the url address of python file
      url: "./examples/pcb-aoi/incremental_learning_bench/testenv/f1_score.py"

    # condition of triggering inference model to update
    # threshold of the condition; types are float/int
    threshold: 0.01
    # operator of the condition; string type;
    # values are ">=", ">", "<=", "<" and "=";
    operator: ">="

  # metrics configuration for test case's evaluation; list type;
  metrics:
    # metric name; string type;
    - name: "f1_score"
      # the url address of python file
      url: "./examples/pcb-aoi/incremental_learning_bench/testenv/f1_score.py"
    - name: "samples_transfer_ratio"

  # incremental rounds setting for incremental learning paradigm; int type; default_
  ↪ value is 2;
  incremental_rounds: 2
```



## HOW TO CONFIG BENCHMARKINGJOB

The algorithm developer is able to test his/her own targeted algorithm using the following configuration information.

### 11.1 The configuration of benchmarkingjob

Prop-erty	Re-quired	Description
name	yes	Job name of benchmarking; Type: string
workspace	no	The url address of job workspace that will reserve the output of tests; Type: string; Default value: <code>./workspace</code>
testenv	yes	The url address of test environment configuration file; Type: string; Value Constraint: The file format supports yaml/yml.
test_object	yes	The configuration of test_object
rank	yes	The configuration of ranking leaderboard

For example:

```
benchmarkingjob:
  # job name of benchmarking; string type;
  name: "benchmarkingjob"
  # the url address of job workspace that will reserve the output of tests; string type;
  # default value: "./workspace"
  workspace: "/ianvs/incremental_learning_bench/workspace"

  # the url address of test environment configuration file; string type;
  # the file format supports yaml/yml;
  testenv: "./examples/pcb-aoi/incremental_learning_bench/testenv/testenv.yaml"
  # the configuration of test object
  test_object:
    ...
  # the configuration of ranking leaderboard
  rank:
    ...
```

### 11.1.1 The configuration of test\_object

Property	Required	Description
type	yes	Type of test object; Type: string; Value Constraint: Currently the option of value is “algorithms”,the others will be added in succession.
algorithms	no	Test algorithm configuration; Type: list

For example:

```
# the configuration of test object
test_object:
  # test type; string type;
  # currently the option of value is "algorithms",the others will be added in succession.
  type: "algorithms"
  # test algorithm configuration files; list type;
  algorithms:
    ...
```

### 11.1.2 The configuration of algorithms

Property	Required	Description
name	yes	Algorithm name; Type: string
url	yes	The url address of test algorithm configuration file; Type: string; Value Constraint: The file format supports yaml/yml.

For example:

```
# test algorithm configuration files; list type;
algorithms:
  # algorithm name; string type;
  - name: "fpn_incremental_learning"
  # the url address of test algorithm configuration file; string type;
  # the file format supports yaml/yml
  url: "./examples/pcb-aoi/incremental_learning_bench/testalgorithms/fpn/fpn_algorithm.
↪yaml"
```

### 11.1.3 The configuration of rank

Property	Required	Description
sort_by	yes	Rank leaderboard with metric of test case's evaluation and order; Type: list; Value Constraint: The sorting priority is based on the sequence of metrics in the list from front to back.
visualization	yes	The configuration of visualization
selected_dataitem	yes	The configuration of selected_dataitem; The user can add his/her interested dataitems in terms of "paradigms", "modules", "hyperparameters" and "metrics", so that the selected columns will be shown.
save_mode	yes	save mode of selected and all dataitems in workspace ./rank; Type: string; Value Constraint: Currently the options of value are as follows: 1> "selected_and_all": save selected and all dataitems. 2> "selected_only": save selected dataitems.

For example:

```
# the configuration of ranking leaderboard
rank:
  # rank leaderboard with metric of test case's evaluation and order ; list type;
  # the sorting priority is based on the sequence of metrics in the list from front to
  ↪back;
  sort_by: [ { "f1_score": "descend" }, { "samples_transfer_ratio": "ascend" } ]
  # visualization configuration
  visualization:
    ...
  # selected dataitem configuration
  # The user can add his/her interested dataitems in terms of "paradigms", "modules",
  ↪"hyperparameters" and "metrics",
  # so that the selected columns will be shown.
  selected_dataitem:
    ...
  # save mode of selected and all dataitems in workspace `./rank` ; string type;
  # currently the options of value are as follows:
  # 1> "selected_and_all": save selected and all dataitems;
  # 2> "selected_only": save selected dataitems;
  save_mode: "selected_and_all"
```

### 11.1.4 The configuration of visualization

Property	Required	Description
mode	no	Mode of visualization in the leaderboard. There are quite a few possible dataitems in the leaderboard. Not all of them can be shown simultaneously on the screen; Type: string; Default value: selected_only
method	no	Method of visualization for selected dataitems; Type: string; Value Constraint: Currently the options of value are as follows: 1> "print_table": print selected dataitems.

For example:

```
# visualization configuration
visualization:
# mode of visualization in the leaderboard; string type;
# There are quite a few possible dataitems in the leaderboard. Not all of them can be
↳ shown simultaneously on the screen.
# In the leaderboard, we provide the "selected_only" mode for the user to configure what
↳ is shown or is not shown.
mode: "selected_only"
# method of visualization for selected dataitems; string type;
# currently the options of value are as follows:
# 1> "print_table": print selected dataitems;
method: "print_table"
```

### 11.1.5 The configuration of selected\_dataitem

Prop-erty	Re-quired	Description
paradigms	yes	Select paradigms in the leaderboard; Type: list; Default value: ["all"]; Value Constraint: Currently the options of value are as follows: 1> "all": select all paradigms in the leaderboard. 2> paradigms in the leaderboard, e.g., "singletasklearning".
mod-ules	yes	Select modules in the leaderboard; Type: list; Default value: ["all"]; Value Constraint: Currently the options of value are as follows: 1> "all": select all hyperparameters in the leaderboard. 2> hyperparameters in the leaderboard, e.g., "momentum".
hyper-pa-rame-ters	yes	Select hyperparameters in the leaderboard; Type: list; Default value: ["all"]; Value Constraint: Currently the options of value are as follows: 1> "all": select all hyperparameters in the leaderboard. 2> hyperparameters in the leaderboard, e.g., "momentum".
met-rics	yes	Select metrics in the leaderboard; Type: list; Default value: ["all"]; Value Constraint: Currently the options of value are as follows: 1> "all": select all metrics in the leaderboard. 2> metrics in the leaderboard, e.g., "f1_score".

```
# selected dataitem configuration
# The user can add his/her interested dataitems in terms of "paradigms", "modules",
↳ "hyperparameters" and "metrics",
# so that the selected columns will be shown.
selected_dataitem:
# currently the options of value are as follows:
# 1> "all": select all paradigms in the leaderboard;
# 2> paradigms in the leaderboard, e.g., "singletasklearning"
paradigms: [ "all" ]
# currently the options of value are as follows:
# 1> "all": select all modules in the leaderboard;
# 2> modules in the leaderboard, e.g., "basemodel"
modules: [ "all" ]
# currently the options of value are as follows:
# 1> "all": select all hyperparameters in the leaderboard;
# 2> hyperparameters in the leaderboard, e.g., "momentum"
hyperparameters: [ "all" ]
# currently the options of value are as follows:
# 1> "all": select all metrics in the leaderboard;
```

(continues on next page)

(continued from previous page)

```
# 2> metrics in the leaderboard, e.g., "F1_SCORE"
metrics: [ "f1_score", "samples_transfer_ratio" ]
```

## 11.2 Show the example

```
benchmarkingjob:
  # job name of benchmarking; string type;
  name: "benchmarkingjob"
  # the url address of job workspace that will reserve the output of tests; string type;
  # default value: "./workspace"
  workspace: "/ianvs/incremental_learning_bench/workspace"

  # the url address of test environment configuration file; string type;
  # the file format supports yaml/yml;
  testenv: "./examples/pcb-aoi/incremental_learning_bench/testenv/testenv.yaml"

  # the configuration of test object
  test_object:
    # test type; string type;
    # currently the option of value is "algorithms", the others will be added in_
    ↪ succession.
    type: "algorithms"
    # test algorithm configuration files; list type;
    algorithms:
      # algorithm name; string type;
      - name: "fpn_incremental_learning"
      # the url address of test algorithm configuration file; string type;
      # the file format supports yaml/yml
      url: "./examples/pcb-aoi/incremental_learning_bench/testalgorithms/fpn/fpn_
    ↪ algorithm.yaml"

  # the configuration of ranking leaderboard
  rank:
    # rank leaderboard with metric of test case's evaluation and order ; list type;
    # the sorting priority is based on the sequence of metrics in the list from front to_
    ↪ back;
    sort_by: [ { "f1_score": "descend" }, { "samples_transfer_ratio": "ascend" } ]

  # visualization configuration
  visualization:
    # mode of visualization in the leaderboard; string type;
    # There are quite a few possible dataitems in the leaderboard. Not all of them can_
    ↪ be shown simultaneously on the screen.
    # In the leaderboard, we provide the "selected_only" mode for the user to_
    ↪ configure what is shown or is not shown.
    mode: "selected_only"
    # method of visualization for selected dataitems; string type;
    # currently the options of value are as follows:
    # 1> "print_table": print selected dataitems;
    method: "print_table"
```

(continues on next page)

(continued from previous page)

```

# selected dataitem configuration
# The user can add his/her interested dataitems in terms of "paradigms", "modules",
↪ "hyperparameters" and "metrics",
# so that the selected columns will be shown.
selected_dataitem:
# currently the options of value are as follows:
# 1> "all": select all paradigms in the leaderboard;
# 2> paradigms in the leaderboard, e.g., "singletasklearning"
paradigms: [ "all" ]
# currently the options of value are as follows:
# 1> "all": select all modules in the leaderboard;
# 2> modules in the leaderboard, e.g., "basemodel"
modules: [ "all" ]
# currently the options of value are as follows:
# 1> "all": select all hyperparameters in the leaderboard;
# 2> hyperparameters in the leaderboard, e.g., "momentum"
hyperparameters: [ "all" ]
# currently the options of value are as follows:
# 1> "all": select all metrics in the leaderboard;
# 2> metrics in the leaderboard, e.g., "f1_score"
metrics: [ "f1_score", "samples_transfer_ratio" ]

# save mode of selected and all dataitems in workspace `./rank` ; string type;
# currently the options of value are as follows:
# 1> "selected_and_all": save selected and all dataitems;
# 2> "selected_only": save selected dataitems;
save_mode: "selected_and_all"

```



## HOW TO USE IANVS COMMAND LINE

### 12.1 List available commands

Command line: `ianvs -h` For example:

```
$ ianvs -h
usage: ianvs [-h] [-f [BENCHMARKING_CONFIG_FILE]] [-v]

AI Benchmarking Tool

optional arguments:
  -h, --help            show this help message and exit
  -f [BENCHMARKING_CONFIG_FILE], --benchmarking_config_file [BENCHMARKING_CONFIG_FILE]
                        run a benchmarking job, and the benchmarking config
                        file must be yaml/yml file.
  -v, --version          show program version info and exit.
```

### 12.2 Show the version of ianvs

Command line: `ianvs -v` For example:

```
$ ianvs -v
0.1.0
```

### 12.3 Run a benchmarking job

Command line: `ianvs -f [BENCHMARKING_CONFIG_FILE]` For example:

```
ianvs -f examples/pcb-aoi/singletask_learning_bench/benchmarkingjob.yaml
```

The final output might look like:

rank	algo- rithm	f1_score	paradigm	base- model	learn- ing_rate	mo- men- tum	time	url
1	fpn_single	0.8396	single- task learn- ing	FPN	0.1	0.5	2022-07-07 20:33:53	/ianvs/pcb-aoi/singletask_learning_bench/workspace/benchmarking/fdf0-11ec-8d5d-fa163eaa99d5
2	fpn_single	0.8353	single- task learn- ing	FPN	0.1	0.95	2022-07-07 20:31:08	/ianvs/pcb-aoi/singletask_learning_bench/workspace/benchmarking/fdf0-11ec-8d5d-fa163eaa99d5

Refer to [details of example](#).

## LEADERBOARD OF SINGLE TASK LEARNING

rank	algorithm	f1_score	paradigm	base-model	learning_rate	momentum	time
1	fpn_singletask_learning	0.8396	single-tasklearning	FPN	0.1	0.5	2022-07-07 20:33:53
2	fpn_singletask_learning	0.8353	single-tasklearning	FPN	0.1	0.95	2022-07-07 20:31:08



## LEADERBOARD OF INCREMENTAL LEARNING

rank	algorithm	f1_score	samples_transfer_ratio	paradigm	base-model	learning_rate	momentum	thresh-old_img	thresh-old_box	time
1	fpn_incremental_learning	0.9572	0.5263	incremental-learning	FPN	0.1	0.95	0.9	0.9	2022-07-07 20:14:12
2	fpn_incremental_learning	0.9441	0.5789	incremental-learning	FPN	0.1	0.5	0.9	0.9	2022-07-07 20:20:57



## TESTING SINGLE TASK LEARNING IN INDUSTRIAL DEFECT DETECTION

### 15.1 About Industrial Defect Detection

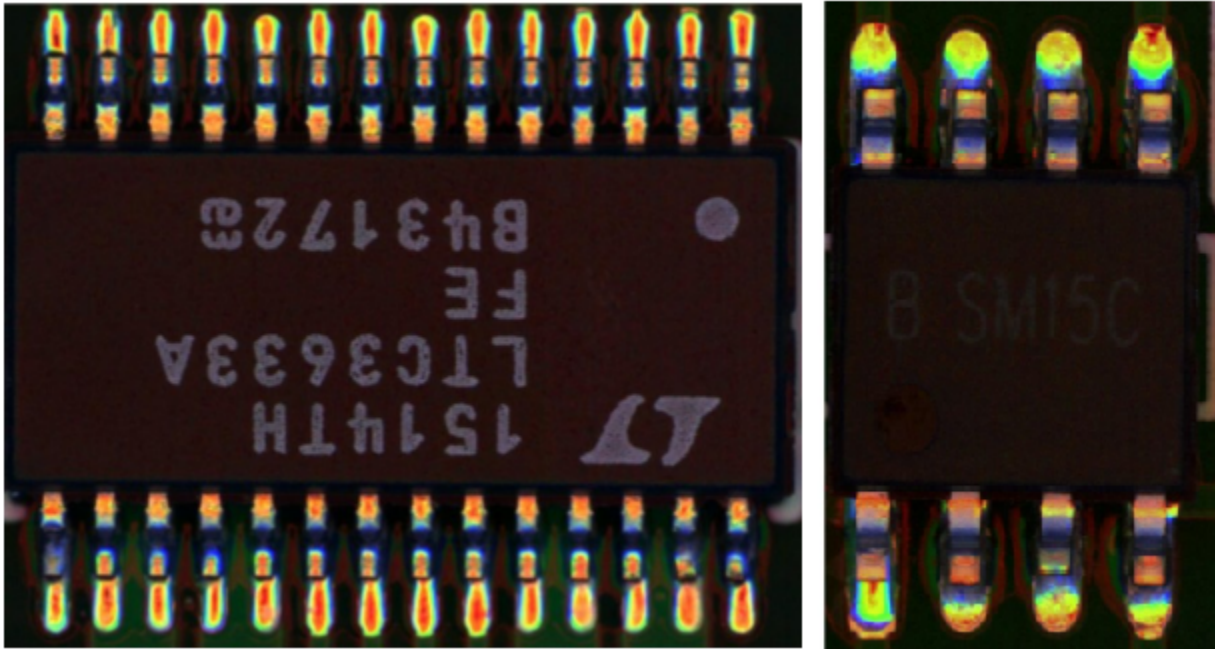
In recent years, the manufacturing process is moving towards a higher degree of automation and improved manufacturing efficiency. During this development, smart manufacturing increasingly employs computing technologies, for example, with a higher degree of automation, there is also a higher risk of product defects; thus, a number of machine learning models have been developed to detect defectives in the manufacturing process.

Defects are an unwanted thing in the manufacturing industry. There are many types of defects in manufacturing like blow holes, pinholes, burr, shrinkage defects, mould material defects, pouring metal defects, metallurgical defects, etc. For removing this defective product all industry have their defect detection department. But the main problem is this inspection process is carried out manually. It is a very time-consuming process and due to human accuracy, this is not 100% accurate. This can be because of the rejection of the whole order. So it creates a big loss for the company.

### 15.2 About Dataset

The printed circuit board (PCB) industry is not different. Surface-mount technology (SMT) is a technology that automates PCB production in which components are mounted or placed onto the surface of printed circuit boards. Solder paste printing (SPP) is the most delicate stage in SMT. It prints solder paste on the pads of an electronic circuit panel. Thus, SPP is followed by a solder paste inspection (SPI) stage to detect defects. SPI scans the printed circuit board for missing/less paste, bridging between pads, miss alignments, and so forth. Boards with anomaly must be detected, and boards in good condition should not be disposed of. Thus SPI requires high precision and a high recall.

As an example in this document, we are using [the PCB-AoI dataset](#) released by KubeEdge SIG AI members on Kaggle. See [this link](#) for more information on this dataset. Below also shows two example figures in the dataset.



## 15.3 About Single Task Learning

Single task learning is a traditional learning pooling all data together to train a single model. It typically includes a specialist model laser-focused on a single task and requires large amounts of task-specific labeled data, which is not always available in the early stage of a distributed synergy AI project.

This report is testing the single task learning algorithm based on FPN\_TensorFlow. It is a Tensorflow re-implementation of Feature Pyramid Networks for Object Detection, which is based on Faster-RCNN. More detailedly, feature pyramids are a basic component in recognition systems for detecting objects at different scales. But recent deep learning object detectors have avoided pyramid representations, in part because they are compute and memory intensive. Researchers have exploited the inherent multi-scale, pyramidal hierarchy of deep convolutional networks to construct feature pyramids with marginal extra cost. A top-down architecture with lateral connections is developed for building high-level semantic feature maps at all scales. The architecture, called a Feature Pyramid Network (FPN), shows significant improvement as a generic feature extractor in several applications. Using FPN in a basic Faster R-CNN system, the method achieves state-of-the-art single-model results on the COCO detection benchmark without bells and whistles, surpassing all existing single-task entries including those from the COCO 2016 challenge winners. In addition, FPN can run at 5 FPS on a GPU and thus is a practical and accurate solution to multi-scale object detection. The FPN\_TensorFlow is also open sourced and completed by YangXue and YangJirui. For those interested in details of FPN\_TensorFlow, an example implementation is available [here](#) and is extended with the lanvs algorithm interface [here](#). Interested readers can refer to [the FPN](#) for more details.



## 15.4 Benchmark Setting

Key settings of the test environment to single task learning are as follows:

```
# testenv.yaml
testenv:
  # dataset configuration
  dataset:
    # the url address of train dataset index; string type;
    train_url: "/ianvs/dataset/train_data/index.txt"
    # the url address of test dataset index; string type;
    test_url: "/ianvs/dataset/test_data/index.txt"

  # metrics configuration for test case's evaluation; list type;
  metrics:
    # metric name; string type;
    - name: "f1_score"
    # the url address of python file
    url: "./examples/pcb-aoi/singletask_learning_bench/testenv/f1_score.py"
```

Key settings of the algorithm to single learning are as follows:

```
# algorithm.yaml
algorithm:
  # paradigm type; string type;
  # currently the options of value are as follows:
  # 1> "singletasklearning"
  # 2> "incrementallearning"
  paradigm_type: "singletasklearning"
  # the url address of initial model; string type; optional;
  initial_model_url: "/ianvs/initial_model/model.zip"

  # algorithm module configuration in the paradigm; list type;
  modules:
    # kind of algorithm module; string type;
    # currently the options of value are as follows:
    # 1> "basemodel"
    - type: "basemodel"
    # name of python module; string type;
    # example: basemodel.py has BaseModel module that the alias is "FPN" for this.
    ↪ benchmarking;
    name: "FPN"
    # the url address of python module; string type;
    url: "./examples/pcb-aoi/singletask_learning_bench/testalgorithms/fpn/basemodel.py"

  # hyperparameters configuration for the python module; list type;
  hyperparameters:
    # name of the hyperparameter; string type;
    - momentum:
      # values of the hyperparameter; list type;
      # types of the value are string/int/float/boolean/list/dictionary
      values:
        - 0.95
```

(continues on next page)

(continued from previous page)

```
- 0.5
# hyperparameters configuration files; dictionary type;
- other_hyperparameters:
  # the url addresses of hyperparameters configuration files; list type;
  # type of the value is string;
  values:
    - "/examples/pcb-aoi/singletask_learning_bench/testalgorithms/fpn/fpn_
↪hyperparameter.yaml"
```

## 15.5 Benchmark Result

We release the leaderboard [here](#).

## TESTING INCREMENTAL LEARNING IN INDUSTRIAL DEFECT DETECTION

### 16.1 About Industrial Defect Detection

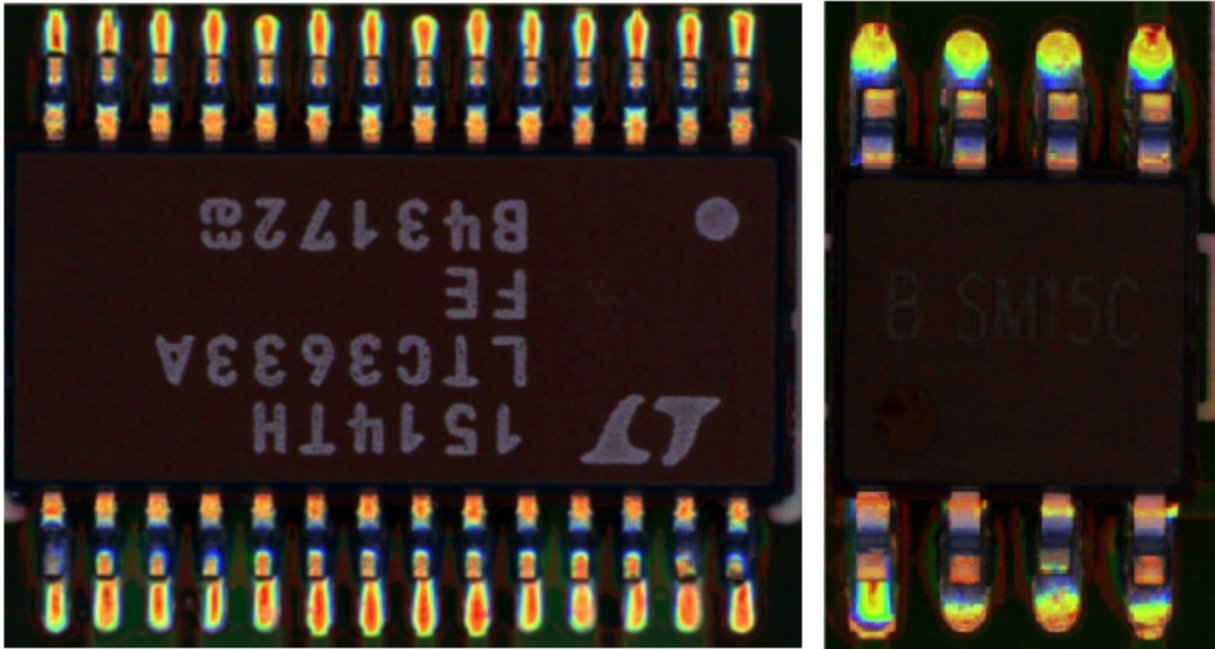
In recent years, the manufacturing process is moving towards a higher degree of automation and improved manufacturing efficiency. During this development, smart manufacturing increasingly employs computing technologies, for example, with a higher degree of automation, there is also a higher risk of product defects; thus, a number of machine learning models have been developed to detect defectives in the manufacturing process.

Defects are an unwanted thing in the manufacturing industry. There are many types of defects in manufacturing like blow holes, pinholes, burr, shrinkage defects, mould material defects, pouring metal defects, metallurgical defects, etc. For removing this defective product all industry have their defect detection department. But the main problem is this inspection process is carried out manually. It is a very time-consuming process and due to human accuracy, this is not 100% accurate. This can be because of the rejection of the whole order. So it creates a big loss for the company.

### 16.2 About Dataset

The printed circuit board (PCB) industry is not different. Surface-mount technology (SMT) is a technology that automates PCB production in which components are mounted or placed onto the surface of printed circuit boards. Solder paste printing (SPP) is the most delicate stage in SMT. It prints solder paste on the pads of an electronic circuit panel. Thus, SPP is followed by a solder paste inspection (SPI) stage to detect defects. SPI scans the printed circuit board for missing/less paste, bridging between pads, miss alignments, and so forth. Boards with anomaly must be detected, and boards in good condition should not be disposed of. Thus SPI requires high precision and a high recall.

As an example in this document, we are using [the PCB-AoI dataset](#) released by KubeEdge SIG AI members on Kaggle. See [this link](#) for more information on this dataset. Below also shows two example figures in the dataset.



## 16.3 About Incremental Learning

Traditionally, the data is collected manually and periodically retrained on the cloud to improve the model effect. However, data is continuously generated on the edge side. The traditional 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.

Its goals include:

- 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 example discovering of unlabeled data, for reducing the manual labeling workload.

This report is testing the basic incremental algorithm based on FPN and interested readers can refer to [the basicIL-fpn](#) for more details.

## 16.4 Benchmark Setting

Key settings of the test environment for incremental learning are as follows:

```
# testenv.yaml
testenv:
  # dataset configuration
  dataset:
    # the url address of train dataset index; string type;
    train_url: "/ianvs/dataset/train_data/index.txt"
    # the url address of test dataset index; string type;
    test_url: "/ianvs/dataset/test_data/index.txt"

  # model eval configuration of incremental learning;
  model_eval:
    # metric used for model evaluation
    model_metric:
      # metric name; string type;
      name: "f1_score"
      # the url address of python file
      url: "./examples/pcb-aoi/incremental_learning_bench/testenv/f1_score.py"

    # condition of triggering inference model to update
    # threshold of the condition; types are float/int
    threshold: 0.01
    # operator of the condition; string type;
    # values are ">=", ">", "<=", "<" and "=";
    operator: ">="

  # metrics configuration for test case's evaluation; list type;
  metrics:
    # metric name; string type;
    - name: "f1_score"
      # the url address of python file
      url: "./examples/pcb-aoi/incremental_learning_bench/testenv/f1_score.py"
    - name: "samples_transfer_ratio"

  # incremental rounds setting for incremental learning paradigm.; int type; default_
  ↪value is 2;
  incremental_rounds: 2
```

Key settings of the algorithm to incremental learning are as follows:

```
# algorithm.yaml
algorithm:
  # paradigm type; string type;
  # currently the options of value are as follows:
  # 1> "singletasklearning"
  # 2> "incrementallearning"
  paradigm_type: "incrementallearning"
  incremental_learning_data_setting:
    # ratio of training dataset; float type.
    # the default value is 0.8.
```

(continues on next page)

(continued from previous page)

```

train_ratio: 0.8
# the method of splitting dataset; string type; optional;
# currently the options of value are as follows:
# 1> "default": the dataset is evenly divided based train_ratio;
splitting_method: "default"
# the url address of initial model for model pre-training; string url;
initial_model_url: "/ianvs/initial_model/model.zip"

# algorithm module configuration in the paradigm; list type;
modules:
# type of algorithm module; string type;
# currently the options of value are as follows:
# 1> "basemodel": contains important interfaces such as train eval predict and
↳more; required module;
- type: "basemodel"
# name of python module; string type;
# example: basemodel.py has BaseModel module that the alias is "FPN" for this
↳benchmarking;
name: "FPN"
# the url address of python module; string type;
url: "./examples/pcb-aoi/incremental_learning_bench/testalgorithms/fpn/basemodel.py
↳"

# hyperparameters configuration for the python module; list type;
hyperparameters:
# name of the hyperparameter; string type;
- momentum:
# values of the hyperparameter; list type;
# types of the value are string/int/float/boolean/list/dictionary
values:
- 0.95
- 0.5
# hyperparameters configuration files; dictionary type;
- other_hyperparameters:
# the url addresses of hyperparameters configuration files; list type;
# type of the value is string;
values:
- "./examples/pcb-aoi/incremental_learning_bench/testalgorithms/fpn/fpn_
↳hyperparameter.yaml"
# 2> "hard_example_mining": check hard example when predict ; optional module;
- type: "hard_example_mining"
# name of python module; string type;
name: "IBT"
# the url address of python module; string type;
url: "./examples/pcb-aoi/incremental_learning_bench/testalgorithms/fpn/hard_
↳example_mining.py"
# hyperparameters configuration for the python module; list type;
hyperparameters:
# name of the hyperparameter; string type;
# threshold of image; value is [0, 1]
- threshold_img:
values:

```

(continues on next page)

(continued from previous page)

```
- 0.9
# predict box of image; value is [0, 1]
- threshold_box:
  values:
    - 0.9
```

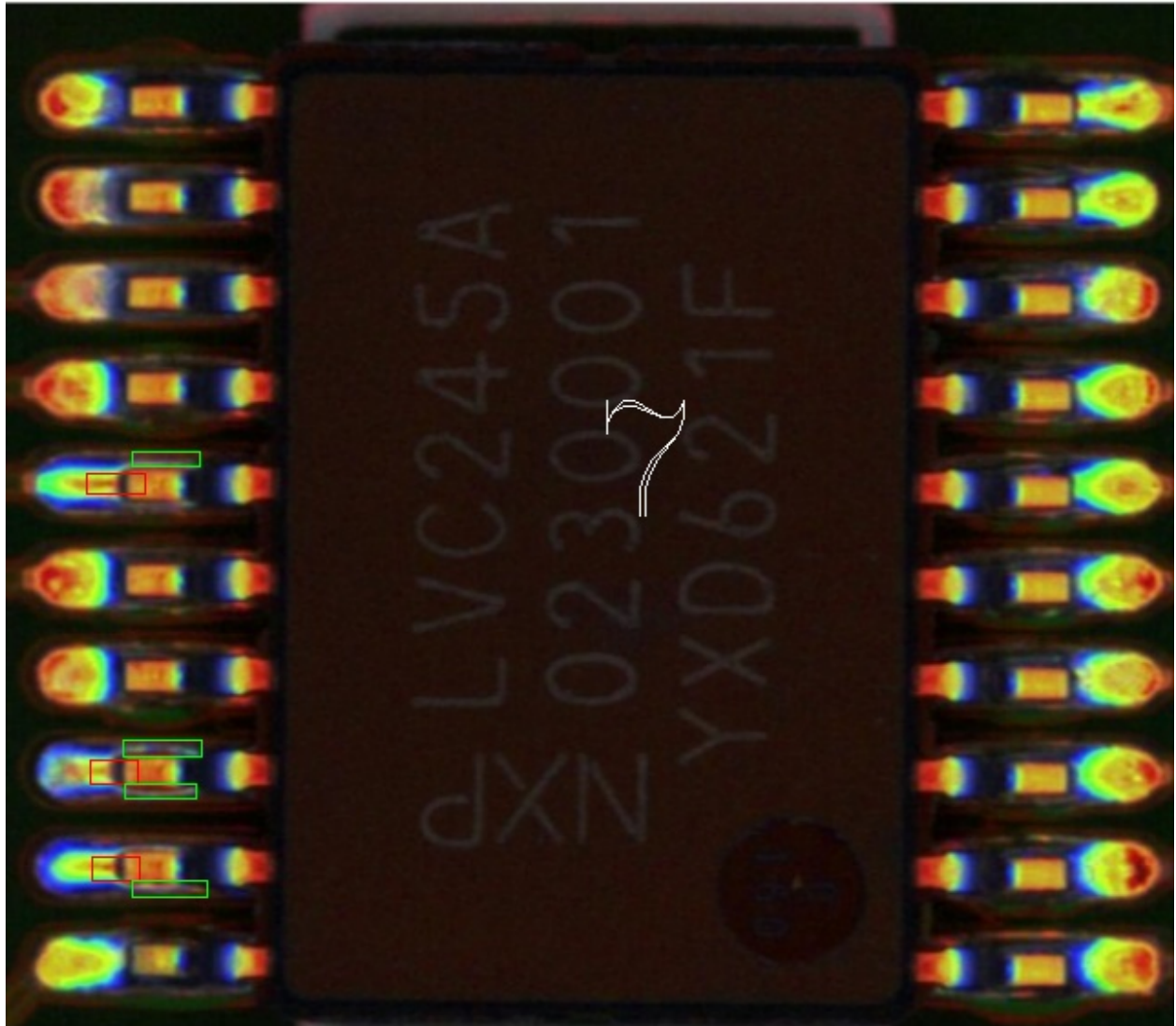
## 16.5 Benchmark Result

We release the leaderboard [here](#) .

## 16.6 Effect Display

The pcb image has 8 bad defects. See [label file](#) for details.

- Before incremental learning, 7 the bad defects have been detected.



- After incremental learning, 8 the bad defects have been detected.







## HOW TO CONTRIBUTE TEST ENVIRONMENTS

### 17.1 Overall contribution workflow

1. Apply for a topic. Once you have a new idea about the test environment, you can apply for a topic to discuss it on [SIG AI weekly meeting](#).
2. Submit proposal. After the idea is fully discussed, the former proposal PR is needed to submit to the [Ianvs repository](#).
3. Fix proposal review comments. If other Ianvs maintainers leave review comments to the PR, you need to fix them and get at least 2 reviewers' /lgtm, and 1 approver's /approve.
4. Submit code. Then you can implement your code, and a good code style is encouraged.
5. Fix code review comments. Besides the merge requirements of the proposal, CI passing is needed before reviewing this step.

The following is a typical testenv:

```
testenv:
# dataset configuration
dataset:
# the url address of train dataset index; string type;
train_url: "/ianvs/dataset/train_data/index.txt"
# the url address of test dataset index; string type;
test_url: "/ianvs/dataset/test_data/index.txt"

# model eval configuration of incremental learning;
model_eval:
# metric used for model evaluation
model_metric:
# metric name; string type;
name: "f1_score"
# the url address of python file
url: "./examples/pcb-aoi/incremental_learning_bench/testenv/f1_score.py"

# condition of triggering inference model to update
# threshold of the condition; types are float/int
threshold: 0.01
# operator of the condition; string type;
# values are ">=", ">", "<=", "<" and "=";
operator: ">="
```

(continues on next page)

(continued from previous page)

```
# metrics configuration for test case's evaluation; list type;
metrics:
  # metric name; string type;
  - name: "f1_score"
  # the url address of python file
  url: "./examples/pcb-aoi/incremental_learning_bench/testenv/f1_score.py"
  - name: "samples_transfer_ratio"

# incremental rounds setting for incremental learning paradigm.; int type; default_
↪value is 2;
incremental_rounds: 2
```

It can be found that for a test, we need to set up the three fields:

- dataset
- model\_eval
- metrics

That means, if you want to test on a different dataset, different model, or different metrics, you need a new test environment.

## 17.2 Add a new test environment

Please refer to the examples directory, `pcb-aoi` is a scenario for testing. We can regard it as a subject for a student that needs to take an exam, the test env is like an examination paper, and the test job is like the student.

For a subject `pcb-aoi`, a new examination paper could be added to the subdirectory, on the same level as a `benchmarking_job`. The detailed steps could be the following:

1. Copy `benchmarking_job` and name `benchmarking_job_2` or any other intuitive name.
2. Add new algorithms to test algorithms, or Keep the useful algorithm. It can refer to contribute algorithm section to develop your own algorithm.
3. Copy `testenv/testnev.yaml`, and modify it based on what you need to test, with different datasets, models, metrics, and so on.

If all things have been done, and you think that would be a nice “examination paper”, you can create PR to `ianvs`, to publish your paper.

Interested “students” from our community will take the exam.

## HOW TO CONTRIBUTRBUTE AN ALGORITHM TO IANVS

Ianvs serves as testing tools for test objects, e.g., algorithms. Ianvs does NOT include code directly on the test object. Algorithms serve as typical test objects in Ianvs and detailed algorithms are thus NOT included in this Ianvs python file. As for the details of example test objects, e.g., algorithms, please refer to third party packages in the Ianvs example. For example, for AI workflow and interface please refer to sedna and for module implementation please refer to third party packages like FPN\_TensorFlow and Sedna IBT algorithm.

For algorithm contributors, you can:

1. Release a repo independent of ianvs, but the interface should still follow the SIG AI algorithm interface to launch ianvs. Here are two examples showing how to develop an algorithm for testing in Ianvs. Here are two examples show how to development algorithm for testing in Ianvs.
  - [incremental-learning](#)
  - [single-task-learning](#)
2. Integrated the targeted algorithm into sedna so that ianvs can use it directly. in this case, you can connect with sedna owners for help.

Also, if a new algorithm has already been integrated into Sedna, it can be used in Ianvs directly.



## HOW TO CONTRIBUTE TEST REPORTS OR LEADERBOARDS

This document helps you to contribute stories, i.e., test reports or leaderboards, for Ianvs. If you follow this guide and find some problem, it is appreciated to submit an issue to update this file.

### 19.1 Test Reports

Everyone is welcome to submit and share your own test report to the community.

#### 19.1.1 1. Setup and Testing

Ianvs 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
```

Clone the Ianvs repo.:

```
git clone http://github.com/kubeedge/ianvs.git
```

Please follow the [Ianvs setup](#) to install Ianvs, and then run your own algorithm to output test reports.

#### 19.1.2 2. Declare your grades

You may want to compare your testing result and those results on the [leaderboard](#).

Test reports are welcome after benchmarking. It can be submitted [here](#) for further review.

### 19.2 Leaderboards

Leaderboards, i.e., rankings of the test object, are public for everyone to visit. Example: [leaderboard](#).

Except for [Ianvs Owners](#), there are mainly two roles for a leaderboard publication:

1. Developer: submit the test object for benchmarking, including but not limited to materials like algorithm, test case following Ianvs settings, and interfaces.
2. Maintainer: testing materials provided by developers and releasing the updated leaderboard to the public.

For potential developers,

- Develop your algorithm with ianvs and choose the algorithm to submit.
- Make sure the submitted test object runs properly under the latest version of Ianvs before submission. Maintainers are not responsible to debug for the submitted objects.
- Do NOT need to submit the new leaderboard. Maintainers are responsible to make the test environment consistent for all test objects under the same leaderboard and execute the test object to generate a new leaderboard.
- If the test object is ready, you are welcome to contact [Ianvs Owners](#). Ianvs owners will connect you and maintainers, in order to receive your test object. Note that when developers submit the test object, developers give maintainers the right to test them.

For potential maintainers,

- To maintain the consistency of test environments and test objects, the [leaderboard](#) submission is at present calling for acknowledged organizations to apply in charge. Please contact
- Maintainers should be responsible for the result submitted.
- Maintainers should update the leaderboard in a monthly manner.
- Maintainers are NOT allowed to use the test object in purpose out of Ianvs benchmarking without formal authorization from developers.
- Besides submitted objects, maintainers are suggested to test objects released in KubeEdge SIG AI or other classic solutions released in public.



## ROADMAP

Upon the release of ianvs, the roadmap would be as follows

- AUG 2022: Release Another Use Case and Advanced Algorithm Paradigm - Non-structured lifelong learning paradigm in ianvs
- SEP 2022: Release Another Use Case, Dataset, and Algorithm Paradigm - Another structured dataset and lifelong learning paradigm in ianvs
- OCT 2022: Release Advanced Benchmark Presentation - shared space for story manager to present your work in public
- NOV 2022: Release Advanced Algorithm Paradigm - Re-ID with Multi-edge Synergy Inference in ianvs
- DEC 2022: Release Simulation Tools
- JUN 2023: More datasets, algorithms, and test cases with ianvs
- DEC 2023: Standards, coding events, and competitions with ianvs



## **IANVS V0.1.0 RELEASE**

### **21.1 1. Release the ianvs distributed synergy AI benchmarking framework.**

- a) Release test environment management and configuration.
- b) Release test case management and configuration.
- c) Release test story management and configuration.
- d) Release the open-source test case generation tool: Use hyperparameter enumeration to fill in one configuration file to generate multiple test cases.

### **21.2 2. Release the PCB-AoI public dataset.**

Release the PCB-AoI public dataset, its corresponding preprocessing, and baseline algorithm projects. Ianvs is the first open-source site for that dataset.

### **21.3 3. Support two new paradigms in test environments and test cases.**

- a) Test environments and test cases that support the single-task learning paradigm.
- b) Test environments and test cases that support the incremental learning paradigm.

### **21.4 4. Release PCB-AoI benchmark cases based on the two new paradigms.**

- a) Release PCB-AoI benchmark cases based on single-task learning, including leaderboards and test reports.
- b) Release PCB-AoI benchmark cases based on incremental learning, including leaderboards and test reports.



## RELATED LINKS

### 22.1 Release

[KubeEdge AI SIG](#)

### 22.2 Meetup and Conference

[HDC.Cloud 2021: AI KubeEdge Sedna AI50%](#)

### 22.3 Distributed Synergy AI Toolkit: Sedna

[Sedna0.4.0](#) [KubeEdgeSedna 0.3.0](#) [AIKubeEdgeSedna 0.1](#) [AIKubeEdgeSedna](#)



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`