mmengine

Release 0.3.0

mmengine contributors

Feb 07, 2023

GET STARTED

1 Introduction 3

2 Installation 5

3 15 minutes to get started with MMEngine 7

4 Registry 13

5 Config 15

6 Runner 17

7 Hook 19

8 Model 21

9 Evaluation 23

10 OptimWrapper 25

11 Parameter Scheduler 27

12 Data transform 29

13 BaseDataset 31

14 Abstract Data Element 33

15 Visualization 35

16 Initialization 37

17 Distribution communication 39

18 Logging 41

19 File IO 43

20 utils 45

21 Resume training 47

22 Speed up training 49

23 Save memory on GPU 51

24 Use modules from other libraries 53

25 Train a GAN 55

26 Hook 57

27 Runner 59

28 Evaluation 61

29 Visualization 63

30 Logging 65

31 Migrate Runner from MMCV to MMEngine 67

32 Migrate Hook from MMCV to MMEngine 69

33 Migrate Model from MMCV to MMEngine 71

34 Migrate Parameter Scheduler from MMCV to MMEngine 73

35 Migrate Transform from MMCV to MMEngine 75

36 mmengine.registry 77

37 mmengine.config 87

38 mmengine.runner 91

39 mmengine.hooks 119

40 mmengine.model 135

41 mmengine.optim 161

42 mmengine.evaluator 189

43 mmengine.structures 193

44 mmengine.dataset 205

45 mmengine.device 217

46 mmengine.hub 219

47 mmengine.logging 221

48 mmengine.visualization 231

49 mmengine.fileio 247

50 mmengine.dist 291

51 mmengine.utils 309

52 mmengine.utils.dl_utils 325

53 Changelog of v0.x 331

54 English 335

55 337

56 Indices and tables 339

Index 341

You can switch between Chinese and English documents in the lower-left corner of the layout.

CHAPTER

ONE

INTRODUCTION

Coming soon. Please refer to chinese documentation.

CHAPTER

TWO

INSTALLATION

2.1 Prerequisites

- Python 3.6+
- PyTorch 1.6+
- CUDA 9.2+
- GCC 5.4+

2.2 Prepare the Environment

1. Use conda and activate the environment:

```
conda create -n open-mmlab python=3.7 -y
conda activate open-mmlab
```

2. Install PyTorch

   Before installing `MMEngine`, please make sure that PyTorch has been successfully installed in the environment. You can refer to PyTorch official installation documentation. Verify the installation with the following command:

```
python -c 'import torch;print(torch.__version__)'
```

2.3 Install MMEngine

2.3.1 Install with mim

mim is a package management tool for OpenMMLab projects, which can be used to install the OpenMMLab project easily.

```
pip install -U openmim
mim install mmengine
```

2.3.2 Install with pip

```
pip install mmengine
```

2.3.3 Use docker images

1. Build the image

   ```
   docker build -t mmengine https://github.com/open-mmlab/mmengine.git#main:docker/
   ↪release
   ```

   More information can be referred from mmengine/docker.

2. Run the image

   ```
   docker run --gpus all --shm-size=8g -it mmengine
   ```

Build from source

```
# if cloning speed is too slow, you can switch the source to https://gitee.com/open-
↪mmlab/mmengine.git
git clone https://github.com/open-mmlab/mmengine.git
cd mmengine
pip install -e . -v
```

2.3.4 Verify the Installation

To verify if `MMEngine` and the necessary environment are successfully installed, we can run this command:

```
python -c 'import mmengine;print(mmengine.__version__)'
```

CHAPTER

THREE

15 MINUTES TO GET STARTED WITH MMENGINE

In this tutorial, we'll take training a ResNet-50 model on CIFAR-10 dataset as an example. We will build a complete and configurable pipeline for both training and validation in only 80 lines of code with `MMEgnine`. The whole process includes the following steps:

1. *Build a Model*
2. *Build a Dataset and DataLoader*
3. *Build a Evaluation Metrics*
4. *Build a Runner and Run the Task*

3.1 Build a Model

First, we need to build a **model**. In MMEngine, the model should inherit from `BaseModel`. Aside from parameters representing inputs from the dataset, its `forward` method needs to accept an extra argument called `mode`:

- for training, the value of `mode` is "loss," and the `forward` method should return a `dict` containing the key "loss".
- for validation, the value of `mode` is "predict", and the forward method should return results containing both predictions and labels.

```
import torch.nn.functional as F
import torchvision
from mmengine.model import BaseModel


class MMResNet50(BaseModel):
    def __init__(self):
        super().__init__()
        self.resnet = torchvision.models.resnet50()

    def forward(self, imgs, labels, mode):
        x = self.resnet(imgs)
        if mode == 'loss':
            return {'loss': F.cross_entropy(x, labels)}
        elif mode == 'predict':
            return x, labels
```

3.2 Build a Dataset and DataLoader

Next, we need to create **Dataset** and **DataLoader** for training and validation. For basic training and validation, we can simply use built-in datasets supported in TorchVision.

```
import torchvision.transforms as transforms
from torch.utils.data import DataLoader

norm_cfg = dict(mean=[0.491, 0.482, 0.447], std=[0.202, 0.199, 0.201])
train_dataloader = DataLoader(batch_size=32,
                              shuffle=True,
                              dataset=torchvision.datasets.CIFAR10(
                                  'data/cifar10',
                                  train=True,
                                  download=True,
                                  transform=transforms.Compose([
                                      transforms.RandomCrop(32, padding=4),
                                      transforms.RandomHorizontalFlip(),
                                      transforms.ToTensor(),
                                      transforms.Normalize(**norm_cfg)
                                  ])))

val_dataloader = DataLoader(batch_size=32,
                            shuffle=False,
                            dataset=torchvision.datasets.CIFAR10(
                                'data/cifar10',
                                train=False,
                                download=True,
                                transform=transforms.Compose([
                                    transforms.ToTensor(),
                                    transforms.Normalize(**norm_cfg)
                                ])))
```

3.3 Build a Evaluation Metrics

To validate and test the model, we need to define a **Metric** called accuracy to evaluate the model. This metric needs inherit from `BaseMetric` and implements the `process` and `compute_metrics` methods where the `process` method accepts the output of the dataset and other outputs when `mode="predict"`. The output data at this scenario is a batch of data. After processing this batch of data, we save the information to `self.results` property. `compute_metrics` accepts a `results` parameter. The input `results` of `compute_metrics` is all the information saved in `process` (In the case of a distributed environment, `results` are the information collected from all `process` in all the processes). Use these information to calculate and return a `dict` that holds the results of the evaluation metrics

```
from mmengine.evaluator import BaseMetric

class Accuracy(BaseMetric):
    def process(self, data_batch, data_samples):
        score, gt = data_samples
        # save the middle result of a batch to `self.results`
        self.results.append({
            'batch_size': len(gt),
```

(continues on next page)

(continued from previous page)

```
            'correct': (score.argmax(dim=1) == gt).sum().cpu(),
        })

    def compute_metrics(self, results):
        total_correct = sum(item['correct'] for item in results)
        total_size = sum(item['batch_size'] for item in results)
        # return the dict containing the eval results
        # the key is the name of the metric name
        return dict(accuracy=100 * total_correct / total_size)
```

3.4 Build a Runner and Run the Task

Now we can build a **Runner** with previously defined `Model`, `DataLoader`, and `Metrics`, and some other configs shown as follows:

```
from torch.optim import SGD
from mmengine.runner import Runner

runner = Runner(
    # the model used for training and validation.
    # Needs to meet specific interface requirements
    model=MMResNet50(),
    # working directory which saves training logs and weight files
    work_dir='./work_dir',
    # train dataloader needs to meet the PyTorch data loader protocol
    train_dataloader=train_dataloader,
    # optimize wrapper for optimization with additional features like
    # AMP, gradtient accumulation, etc
    optim_wrapper=dict(optimizer=dict(type=SGD, lr=0.001, momentum=0.9)),
    # trainging coinfs for specifying training epoches, verification intervals, etc
    train_cfg=dict(by_epoch=True, max_epochs=5, val_interval=1),
    # validation dataloaer also needs to meet the PyTorch data loader protocol
    val_dataloader=val_dataloader,
    # validation configs for specifying additional parameters required for validation
    val_cfg=dict(),
    # validation evaluator. The default one is used here
    val_evaluator=dict(type=Accuracy),
)

runner.train()
```

Finally, let's put all the codes above together into a complete script that uses the `MMEngine` executor for training and validation:

```
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
from torch.optim import SGD
from torch.utils.data import DataLoader
```

(continues on next page)

(continued from previous page)

```
from mmengine.evaluator import BaseMetric
from mmengine.model import BaseModel
from mmengine.runner import Runner


class MMResNet50(BaseModel):
    def __init__(self):
        super().__init__()
        self.resnet = torchvision.models.resnet50()

    def forward(self, imgs, labels, mode):
        x = self.resnet(imgs)
        if mode == 'loss':
            return {'loss': F.cross_entropy(x, labels)}
        elif mode == 'predict':
            return x, labels


class Accuracy(BaseMetric):
    def process(self, data_batch, data_samples):
        score, gt = data_samples
        self.results.append({
            'batch_size': len(gt),
            'correct': (score.argmax(dim=1) == gt).sum().cpu(),
        })

    def compute_metrics(self, results):
        total_correct = sum(item['correct'] for item in results)
        total_size = sum(item['batch_size'] for item in results)
        return dict(accuracy=100 * total_correct / total_size)


norm_cfg = dict(mean=[0.491, 0.482, 0.447], std=[0.202, 0.199, 0.201])
train_dataloader = DataLoader(batch_size=32,
                              shuffle=True,
                              dataset=torchvision.datasets.CIFAR10(
                                  'data/cifar10',
                                  train=True,
                                  download=True,
                                  transform=transforms.Compose([
                                      transforms.RandomCrop(32, padding=4),
                                      transforms.RandomHorizontalFlip(),
                                      transforms.ToTensor(),
                                      transforms.Normalize(**norm_cfg)
                                  ])))

val_dataloader = DataLoader(batch_size=32,
                            shuffle=False,
                            dataset=torchvision.datasets.CIFAR10(
                                'data/cifar10',
                                train=False,
```

(continues on next page)

(continued from previous page)

```
                                download=True,
                                transform=transforms.Compose([
                                    transforms.ToTensor(),
                                    transforms.Normalize(**norm_cfg)
                                ]))

runner = Runner(
    model=MMResNet50(),
    work_dir='./work_dir',
    train_dataloader=train_dataloader,
    optim_wrapper=dict(optimizer=dict(type=SGD, lr=0.001, momentum=0.9)),
    train_cfg=dict(by_epoch=True, max_epochs=5, val_interval=1),
    val_dataloader=val_dataloader,
    val_cfg=dict(),
    val_evaluator=dict(type=Accuracy),
)
runner.train()
```

Training log would be similar to this:

```
2022/08/22 15:51:53 - mmengine - INFO -
------------------------------------------------------------
System environment:
    sys.platform: linux
    Python: 3.8.12 (default, Oct 12 2021, 13:49:34) [GCC 7.5.0]
    CUDA available: True
    numpy_random_seed: 1513128759
    GPU 0: NVIDIA GeForce GTX 1660 SUPER
    CUDA_HOME: /usr/local/cuda
...

2022/08/22 15:51:54 - mmengine - INFO - Checkpoints will be saved to /home/mazerun/work_
↪dir by HardDiskBackend.
2022/08/22 15:51:56 - mmengine - INFO - Epoch(train) [1][10/1563]  lr: 1.0000e-03  eta:␣
↪0:18:23  time: 0.1414  data_time: 0.0077  memory: 392  loss: 5.3465
2022/08/22 15:51:56 - mmengine - INFO - Epoch(train) [1][20/1563]  lr: 1.0000e-03  eta:␣
↪0:11:29  time: 0.0354  data_time: 0.0077  memory: 392  loss: 2.7734
2022/08/22 15:51:56 - mmengine - INFO - Epoch(train) [1][30/1563]  lr: 1.0000e-03  eta:␣
↪0:09:10  time: 0.0352  data_time: 0.0076  memory: 392  loss: 2.7789
2022/08/22 15:51:57 - mmengine - INFO - Epoch(train) [1][40/1563]  lr: 1.0000e-03  eta:␣
↪0:08:00  time: 0.0353  data_time: 0.0073  memory: 392  loss: 2.5725
2022/08/22 15:51:57 - mmengine - INFO - Epoch(train) [1][50/1563]  lr: 1.0000e-03  eta:␣
↪0:07:17  time: 0.0347  data_time: 0.0073  memory: 392  loss: 2.7382
2022/08/22 15:51:57 - mmengine - INFO - Epoch(train) [1][60/1563]  lr: 1.0000e-03  eta:␣
↪0:06:49  time: 0.0347  data_time: 0.0072  memory: 392  loss: 2.5956
2022/08/22 15:51:58 - mmengine - INFO - Epoch(train) [1][70/1563]  lr: 1.0000e-03  eta:␣
↪0:06:28  time: 0.0348  data_time: 0.0072  memory: 392  loss: 2.7351
...
2022/08/22 15:52:50 - mmengine - INFO - Saving checkpoint at 1 epochs
2022/08/22 15:52:51 - mmengine - INFO - Epoch(val) [1][10/313]    eta: 0:00:03  time: 0.
↪0122  data_time: 0.0047  memory: 392
2022/08/22 15:52:51 - mmengine - INFO - Epoch(val) [1][20/313]    eta: 0:00:03  time: 0.
↪0122  data_time: 0.0047  memory: 308
```

(continues on next page)

(continued from previous page)

```
2022/08/22 15:52:51 - mmengine - INFO - Epoch(val) [1][30/313]    eta: 0:00:03  time: 0.
↪0123  data_time: 0.0047  memory: 308
...
2022/08/22 15:52:54 - mmengine - INFO - Epoch(val) [1][313/313]  accuracy: 35.7000
```

In addition to these basic components, you can also use **executor** to easily combine and configure various training techniques, such as enabling mixed-precision training and gradient accumulation (see *OptimWrapper*), configuring the learning rate decay curve (see *Metrics & Evaluator*), and etc.

CHAPTER

FOUR

REGISTRY

Coming soon. Please refer to chinese documentation.

CHAPTER

FIVE

CONFIG

Coming soon. Please refer to chinese documentation.

CHAPTER

SIX

RUNNER

Coming soon. Please refer to chinese documentation.

CHAPTER

SEVEN

HOOK

Coming soon. Please refer to chinese documentation.

CHAPTER

EIGHT

MODEL

Coming soon. Please refer to chinese documentation.

CHAPTER

NINE

EVALUATION

Coming soon. Please refer to chinese documentation.

CHAPTER

TEN

OPTIMWRAPPER

Coming soon. Please refer to chinese documentation.

CHAPTER

ELEVEN

PARAMETER SCHEDULER

Coming soon. Please refer to chinese documentation.

CHAPTER

TWELVE

DATA TRANSFORM

Coming soon. Please refer to chinese documentation.

CHAPTER

THIRTEEN

BASEDATASET

Coming soon. Please refer to chinese documentation.

CHAPTER

FOURTEEN

ABSTRACT DATA ELEMENT

Coming soon. Please refer to chinese documentation.

CHAPTER

FIFTEEN

VISUALIZATION

Coming soon. Please refer to chinese documentation.

CHAPTER

SIXTEEN

INITIALIZATION

Coming soon. Please refer to chinese documentation.

CHAPTER

SEVENTEEN

DISTRIBUTION COMMUNICATION

Coming soon. Please refer to chinese documentation.

CHAPTER

EIGHTEEN

LOGGING

Coming soon. Please refer to chinese documentation.

CHAPTER

NINETEEN

FILE IO

Coming soon. Please refer to chinese documentation.

CHAPTER

TWENTY

UTILS

Coming soon. Please refer to chinese documentation.

CHAPTER

TWENTYONE

RESUME TRAINING

Coming soon. Please refer to chinese documentation.

CHAPTER

TWENTYTWO

SPEED UP TRAINING

Coming soon. Please refer to chinese documentation.

CHAPTER

TWENTYTHREE

SAVE MEMORY ON GPU

Coming soon. Please refer to chinese documentation.

CHAPTER

TWENTYFOUR

USE MODULES FROM OTHER LIBRARIES

Coming soon. Please refer to chinese documentation.

CHAPTER

TWENTYFIVE

TRAIN A GAN

Coming soon. Please refer to chinese documentation.

CHAPTER

TWENTYSIX

HOOK

Coming soon. Please refer to chinese documentation.

CHAPTER

TWENTYSEVEN

RUNNER

Coming soon. Please refer to chinese documentation.

CHAPTER

TWENTYEIGHT

EVALUATION

Coming soon. Please refer to chinese documentation.

CHAPTER

TWENTYNINE

VISUALIZATION

Coming soon. Please refer to chinese documentation.

CHAPTER

THIRTY

LOGGING

Coming soon. Please refer to chinese documentation.

CHAPTER

THIRTYONE

MIGRATE RUNNER FROM MMCV TO MMENGINE

Coming soon. Please refer to chinese documentation.

CHAPTER

THIRTYTWO

MIGRATE HOOK FROM MMCV TO MMENGINE

Coming soon. Please refer to chinese documentation.

CHAPTER

THIRTYTHREE

MIGRATE MODEL FROM MMCV TO MMENGINE

Coming soon. Please refer to chinese documentation.

CHAPTER

THIRTYFOUR

MIGRATE PARAMETER SCHEDULER FROM MMCV TO MMENGINE

Coming soon. Please refer to chinese documentation.

CHAPTER

THIRTYFIVE

MIGRATE TRANSFORM FROM MMCV TO MMENGINE

Coming soon. Please refer to chinese documentation.

CHAPTER

THIRTYSIX

MMENGINE.REGISTRY

Registry	A registry to map strings to classes or functions.
DefaultScope	Scope of current task used to reset the current registry, which can be accessed globally.

36.1 Registry

class mmengine.registry.**Registry**(*name*, *build_func=None*, *parent=None*, *scope=None*)

A registry to map strings to classes or functions.

Registered object could be built from registry. Meanwhile, registered functions could be called from registry.

Parameters

- **name** (*str*) – Registry name.
- **build_func** (*callable, optional*) – A function to construct instance from Registry. *build_from_cfg()* is used if neither `parent` or `build_func` is specified. If `parent` is specified and `build_func` is not given, `build_func` will be inherited from `parent`. Defaults to None.
- **parent** (*Registry*, optional) – Parent registry. The class registered in children registry could be built from parent. Defaults to None.
- **scope** (*str, optional*) – The scope of registry. It is the key to search for children registry. If not specified, scope will be the name of the package where class is defined, e.g. mmdet, mmcls, mmseg. Defaults to None.

Examples

```
>>> # define a registry
>>> MODELS = Registry('models')
>>> # registry the `ResNet` to `MODELS`
>>> @MODELS.register_module()
>>> class ResNet:
>>>     pass
>>> # build model from `MODELS`
>>> resnet = MODELS.build(dict(type='ResNet'))
>>> @MODELS.register_module()
>>> def resnet50():
```

(continues on next page)

(continued from previous page)

```
>>>     pass
>>> resnet = MODELS.build(dict(type='resnet50'))
```

```
>>> # hierarchical registry
>>> DETECTORS = Registry('detectors', parent=MODELS, scope='det')
>>> @DETECTORS.register_module()
>>> class FasterRCNN:
>>>     pass
>>> fasterrcnn = DETECTORS.build(dict(type='FasterRCNN'))
```

More advanced usages can be found at https://mmengine.readthedocs.io/en/latest/tutorials/registry.html.

build(*cfg*, **args*, ***kwargs*)

Build an instance.

Build an instance by calling `build_func`.

Parameters **cfg** (*dict*) – Config dict needs to be built.

Returns The constructed object.

Return type Any

Examples

```
>>> from mmengine import Registry
>>> MODELS = Registry('models')
>>> @MODELS.register_module()
>>> class ResNet:
>>>     def __init__(self, depth, stages=4):
>>>         self.depth = depth
>>>         self.stages = stages
>>> cfg = dict(type='ResNet', depth=50)
>>> model = MODELS.build(cfg)
```

get(*key*)

Get the registry record.

The method will first parse `key` and check whether it contains a scope name. The logic to search for `key`:

- `key` does not contain a scope name, i.e., it is purely a module name like "ResNet": *get()* will search for `ResNet` from the current registry to its parent or ancestors until finding it.
- `key` contains a scope name and it is equal to the scope of the current registry (e.g., "mmcls"), e.g., "mmcls.ResNet": *get()* will only search for `ResNet` in the current registry.
- `key` contains a scope name and it is not equal to the scope of the current registry (e.g., "mmdet"), e.g., "mmcls.FCNet": If the scope exists in its children, *get()* will get "FCNet" from them. If not, *get()* will first get the root registry and root registry call its own *get()* method.

Parameters **key** (*str*) – Name of the registered item, e.g., the class name in string format.

Returns Return the corresponding class if `key` exists, otherwise return None.

Return type Type or None

Examples

```
>>> # define a registry
>>> MODELS = Registry('models')
>>> # register `ResNet` to `MODELS`
>>> @MODELS.register_module()
>>> class ResNet:
>>>     pass
>>> resnet_cls = MODELS.get('ResNet')
```

```
>>> # hierarchical registry
>>> DETECTORS = Registry('detector', parent=MODELS, scope='det')
>>> # `ResNet` does not exist in `DETECTORS` but `get` method
>>> # will try to search from its parenet or ancestors
>>> resnet_cls = DETECTORS.get('ResNet')
>>> CLASSIFIER = Registry('classifier', parent=MODELS, scope='cls')
>>> @CLASSIFIER.register_module()
>>> class MobileNet:
>>>     pass
>>> # `get` from its sibling registries
>>> mobilenet_cls = DETECTORS.get('cls.MobileNet')
```

static infer_scope()

Infer the scope of registry.

The name of the package where registry is defined will be returned.

Returns The inferred scope name.

Return type str

Examples

```
>>> # in mmdet/models/backbone/resnet.py
>>> MODELS = Registry('models')
>>> @MODELS.register_module()
>>> class ResNet:
>>>     pass
>>> # The scope of ``ResNet`` will be ``mmdet``.
```

register_module(*name=None*, *force=False*, *module=None*)

Register a module.

A record will be added to `self._module_dict`, whose key is the class name or the specified name, and value is the class itself. It can be used as a decorator or a normal function.

Parameters

- **name** (*str or list of str, optional*) – The module name to be registered. If not specified, the class name will be used.
- **force** (*bool*) – Whether to override an existing class with the same name. Default to False.
- **module** (*type, optional*) – Module class or function to be registered. Defaults to None.

Return type Union[type, collections.abc.Callable]

Examples

```
>>> backbones = Registry('backbone')
>>> # as a decorator
>>> @backbones.register_module()
>>> class ResNet:
>>>     pass
>>> backbones = Registry('backbone')
>>> @backbones.register_module(name='mnet')
>>> class MobileNet:
>>>     pass
```

```
>>> # as a normal function
>>> class ResNet:
>>>     pass
>>> backbones.register_module(module=ResNet)
```

static split_scope_key(*key*)

Split scope and key.

The first scope will be split from key.

Returns The former element is the first scope of the key, which can be None. The latter is the remaining key.

Return type tuple[str | None, str]

Parameters **key** (*str*) –

Examples

```
>>> Registry.split_scope_key('mmdet.ResNet')
'mmdet', 'ResNet'
>>> Registry.split_scope_key('ResNet')
None, 'ResNet'
```

switch_scope_and_registry(*scope*)

Temporarily switch default scope to the target scope, and get the corresponding registry.

If the registry of the corresponding scope exists, yield the registry, otherwise yield the current itself.

Parameters **scope** (*str*) – The target scope.

Return type Generator

Examples

```
>>> from mmengine.registry import Registry, DefaultScope, MODELS
>>> import time
>>> # External Registry
>>> MMDET_MODELS = Registry('mmdet_model', scope='mmdet',
>>>     parent=MODELS)
>>> MMCLS_MODELS = Registry('mmcls_model', scope='mmcls',
>>>     parent=MODELS)
```

(continues on next page)

(continued from previous page)

```
>>> # Local Registry
>>> CUSTOM_MODELS = Registry('custom_model', scope='custom',
>>>     parent=MODELS)
>>>
>>> # Initiate DefaultScope
>>> DefaultScope.get_instance(f'scope_{time.time()}',
>>>     scope_name='custom')
>>> # Check default scope
>>> DefaultScope.get_current_instance().scope_name
custom
>>> # Switch to mmcls scope and get `MMCLS_MODELS` registry.
>>> with CUSTOM_MODELS.switch_scope_and_registry(scope='mmcls') as registry:
>>>     DefaultScope.get_current_instance().scope_name
mmcls
>>>     registry.scope
mmcls
>>> # Nested switch scope
>>> with CUSTOM_MODELS.switch_scope_and_registry(scope='mmdet') as mmdet_
↪registry:
>>>     DefaultScope.get_current_instance().scope_name
mmdet
>>>     mmdet_registry.scope
mmdet
>>>     with CUSTOM_MODELS.switch_scope_and_registry(scope='mmcls') as mmcls_
↪registry:
>>>         DefaultScope.get_current_instance().scope_name
mmcls
>>>         mmcls_registry.scope
mmcls
>>>
>>> # Check switch back to original scope.
>>> DefaultScope.get_current_instance().scope_name
custom
```

36.2 DefaultScope

class mmengine.registry.**DefaultScope**(*name*, *scope_name*)

Scope of current task used to reset the current registry, which can be accessed globally.

Consider the case of resetting the current `Resgitry` by``default_scope`` in the internal module which cannot access runner directly, it is difficult to get the `default_scope` defined in `Runner`. However, if `Runner` created `DefaultScope` instance by given `default_scope`, the internal module can get `default_scope` by `DefaultScope.get_current_instance` everywhere.

Parameters

- **name** (*str*) – Name of default scope for global access.
- **scope_name** (*str*) – Scope of current task.

Examples

```
>>> from mmengine.model import MODELS
>>> # Define default scope in runner.
>>> DefaultScope.get_instance('task', scope_name='mmdet')
>>> # Get default scope globally.
>>> scope_name = DefaultScope.get_instance('task').scope_name
```

classmethod get_current_instance()

Get latest created default scope.

Since default_scope is an optional argument for `Registry.build`. `get_current_instance` should return `None` if there is no `DefaultScope` created.

Examples

```
>>> default_scope = DefaultScope.get_current_instance()
>>> # There is no `DefaultScope` created yet,
>>> # `get_current_instance` return `None`.
>>> default_scope = DefaultScope.get_instance(
>>>     'instance_name', scope_name='mmengine')
>>> default_scope.scope_name
mmengine
>>> default_scope = DefaultScope.get_current_instance()
>>> default_scope.scope_name
mmengine
```

Returns Return None If there has not been `DefaultScope` instance created yet, otherwise return the latest created DefaultScope instance.

Return type Optional[*DefaultScope*]

classmethod overwrite_default_scope(*scope_name*)

overwrite the current default scope with *scope_name*

Parameters **scope_name** (*Optional[str]*) –

Return type Generator

property scope_name: str

Returns: str: Get current scope.

build_from_cfg	Build a module from config dict when it is a class configuration, or call a function from config dict when it is a function configuration.
build_model_from_cfg	Build a PyTorch model from config dict(s).
build_runner_from_cfg	Build a Runner object.
build_scheduler_from_cfg	Builds a `ParamScheduler` instance from config.
count_registered_modules	Scan all modules in MMEngine's root and child registries and dump to json.
traverse_registry_tree	Traverse the whole registry tree from any given node, and collect information of all registered modules in this registry tree.

36.3 mmengine.registry.build_from_cfg

mmengine.registry.**build_from_cfg**(*cfg*, *registry*, *default_args=None*)

Build a module from config dict when it is a class configuration, or call a function from config dict when it is a function configuration.

If the global variable default scope (*DefaultScope*) exists, `build()` will firstly get the responding registry and then call its own `build()`.

At least one of the `cfg` and `default_args` contains the key "type", which should be either str or class. If they all contain it, the key in `cfg` will be used because `cfg` has a high priority than `default_args` that means if a key exists in both of them, the value of the key will be `cfg[key]`. They will be merged first and the key "type" will be popped up and the remaining keys will be used as initialization arguments.

Examples

```
>>> from mmengine import Registry, build_from_cfg
>>> MODELS = Registry('models')
>>> @MODELS.register_module()
>>> class ResNet:
>>>     def __init__(self, depth, stages=4):
>>>         self.depth = depth
>>>         self.stages = stages
>>> cfg = dict(type='ResNet', depth=50)
>>> model = build_from_cfg(cfg, MODELS)
>>> # Returns an instantiated object
>>> @MODELS.register_module()
>>> def resnet50():
>>>     pass
>>> resnet = build_from_cfg(dict(type='resnet50'), MODELS)
>>> # Return a result of the calling function
```

Parameters

- **cfg** (*dict or ConfigDict or Config*) – Config dict. It should at least contain the key "type".
- **registry** (*Registry*) – The registry to search the type from.
- **default_args** (*dict or ConfigDict or Config, optional*) – Default initialization arguments. Defaults to None.

Returns The constructed object.

Return type object

36.4 mmengine.registry.build_model_from_cfg

mmengine.registry.**build_model_from_cfg**(*cfg*, *registry*, *default_args=None*)

Build a PyTorch model from config dict(s). Different from `build_from_cfg`, if cfg is a list, a `nn.Sequential` will be built.

Parameters

- **cfg** (*dict, list[dict]*) – The config of modules, which is either a config dict or a list of config dicts. If cfg is a list, the built modules will be wrapped with `nn.Sequential`.
- **registry** (*Registry*) – A registry the module belongs to.
- **default_args** (*dict, optional*) – Default arguments to build the module. Defaults to None.

Returns A built nn.Module.

Return type nn.Module

36.5 mmengine.registry.build_runner_from_cfg

mmengine.registry.**build_runner_from_cfg**(*cfg*, *registry*)

Build a Runner object. .. rubric:: Examples

```
>>> from mmengine.registry import Registry, build_runner_from_cfg
>>> RUNNERS = Registry('runners', build_func=build_runner_from_cfg)
>>> @RUNNERS.register_module()
>>> class CustomRunner(Runner):
>>>     def setup_env(env_cfg):
>>>         pass
>>> cfg = dict(runner_type='CustomRunner', ...)
>>> custom_runner = RUNNERS.build(cfg)
```

Parameters

- **cfg** (*dict or ConfigDict or Config*) – Config dict. If "runner_type" key exists, it will be used to build a custom runner. Otherwise, it will be used to build a default runner.
- **registry** (*Registry*) – The registry to search the type from.

Returns The constructed runner object.

Return type object

36.6 mmengine.registry.build_scheduler_from_cfg

mmengine.registry.**build_scheduler_from_cfg**(*cfg*, *registry*, *default_args=None*)

Builds a `ParamScheduler` instance from config.

`ParamScheduler` supports building instance by its constructor or method `build_iter_from_epoch`. Therefore, its registry needs a build function to handle both cases.

Parameters

- **cfg** (*dict or ConfigDict or Config*) – Config dictionary. If it contains the key `convert_to_iter_based`, instance will be built by method `convert_to_iter_based`, otherwise instance will be built by its constructor.
- **registry** (*Registry*) – The `PARAM_SCHEDULERS` registry.
- **default_args** (*dict or ConfigDict or Config, optional*) – Default initialization arguments. It must contain key `optimizer`. If `convert_to_iter_based` is defined in `cfg`, it must additionally contain key `epoch_length`. Defaults to None.

Returns The constructed `ParamScheduler`.

Return type object

36.7 mmengine.registry.count_registered_modules

`mmengine.registry.count_registered_modules`(*save_path=None*, *verbose=True*)

Scan all modules in MMEngine's root and child registries and dump to json.

Parameters

- **save_path** (*str, optional*) – Path to save the json file.
- **verbose** (*bool*) – Whether to print log. Defaults to True.

Returns Statistic results of all registered modules.

Return type dict

36.8 mmengine.registry.traverse_registry_tree

`mmengine.registry.traverse_registry_tree`(*registry*, *verbose=True*)

Traverse the whole registry tree from any given node, and collect information of all registered modules in this registry tree.

Parameters

- **registry** (Registry) – a registry node in the registry tree.
- **verbose** (*bool*) – Whether to print log. Default: True

Returns Statistic results of all modules in each node of the registry tree.

Return type list

CHAPTER

THIRTYSEVEN

MMENGINE.CONFIG

Config	A facility for config and config files.
ConfigDict	A dictionary for config which has the same interface as python's built- in dictionary and can be used as a normal dictionary.
DictAction	argparse action to split an argument into KEY=VALUE form on the first = and append to a dictionary.

37.1 Config

class mmengine.config.**Config**(*cfg_dict=None*, *cfg_text=None*, *filename=None*)

A facility for config and config files.

It supports common file formats as configs: python/json/yaml. `Config.fromfile` can parse a dictionary from a config file, then build a `Config` instance with the dictionary. The interface is the same as a dict object and also allows access config values as attributes.

Parameters

- **cfg_dict** (*dict, optional*) – A config dictionary. Defaults to None.
- **cfg_text** (*str, optional*) – Text of config. Defaults to None.
- **filename** (*str or Path, optional*) – Name of config file. Defaults to None.

Examples

```
>>> cfg = Config(dict(a=1, b=dict(b1=[0, 1])))
>>> cfg.a
1
>>> cfg.b
{'b1': [0, 1]}
>>> cfg.b.b1
[0, 1]
>>> cfg = Config.fromfile('tests/data/config/a.py')
>>> cfg.filename
"/home/username/projects/mmengine/tests/data/config/a.py"
>>> cfg.item4
'test'
>>> cfg
```

(continues on next page)

(continued from previous page)

```
"Config [path: /home/username/projects/mmengine/tests/data/config/a.py]
:"
"{'item1': [1, 2], 'item2': {'a': 0}, 'item3': True, 'item4': 'test'}"
```

static auto_argparser(*description=None*)
Generate argparser from config file automatically (experimental)

dump(*file=None*)
Dump config to file or return config text.

Parameters

- **file** (*str or Path, optional*) – If not specified, then the object
- **dumped to a str** (*is*) –
- **to a file specified by the filename.** (*otherwise*) –
- **to None.** (*Defaults*) –

Returns Config text.

Return type str or None

property filename: str
get file name of config.

static fromfile(*filename*, *use_predefined_variables=True*, *import_custom_modules=True*)
Build a Config instance from config file.

Parameters

- **filename** (*str or Path*) – Name of config file.
- **use_predefined_variables** (*bool, optional*) – Whether to use predefined variables. Defaults to True.
- **import_custom_modules** (*bool, optional*) – Whether to support importing custom modules in config. Defaults to True.

Returns Config instance built from config file.

Return type *Config*

static fromstring(*cfg_str*, *file_format*)
Build a Config instance from config text.

Parameters

- **cfg_str** (*str*) – Config text.
- **file_format** (*str*) – Config file format corresponding to the config str. Only py/yml/yaml/json type are supported now!

Returns Config object generated from `cfg_str`.

Return type *Config*

merge_from_dict(*options*, *allow_list_keys=True*)
Merge list into cfg_dict.

Merge the dict parsed by MultipleKVAction into this cfg.

Parameters

- **options** (*dict*) – dict of configs to merge from.

- **allow_list_keys** (*bool*) – If True, int string keys (e.g. '0', '1') are allowed in `options` and will replace the element of the corresponding index in the config if the config is a list. Defaults to True.

Return type None

Examples

```
>>> from mmengine import Config
>>> #  Merge dictionary element
>>> options = {'model.backbone.depth': 50, 'model.backbone.with_cp': True}
>>> cfg = Config(dict(model=dict(backbone=dict(type='ResNet'))))
>>> cfg.merge_from_dict(options)
>>> cfg._cfg_dict
{'model': {'backbone': {'type': 'ResNet', 'depth': 50, 'with_cp': True}}}
>>> # Merge list element
>>> cfg = Config(
>>>     dict(pipeline=[dict(type='LoadImage'),
>>>                    dict(type='LoadAnnotations')]))
>>> options = dict(pipeline={'0': dict(type='SelfLoadImage')})
>>> cfg.merge_from_dict(options, allow_list_keys=True)
>>> cfg._cfg_dict
{'pipeline': [{'type': 'SelfLoadImage'}, {'type': 'LoadAnnotations'}]}
```

property pretty_text: str
get formatted python config text.

property text: str
get config text.

37.2 ConfigDict

class mmengine.config.**ConfigDict**(**args*, ***kwargs*)
A dictionary for config which has the same interface as python's built- in dictionary and can be used as a normal dictionary.

The Config class would transform the nested fields (dictionary-like fields) in config file into `ConfigDict`.

37.3 DictAction

class mmengine.config.**DictAction**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)
argparse action to split an argument into KEY=VALUE form on the first = and append to a dictionary. List options can be passed as comma separated values, i.e 'KEY=V1,V2,V3', or with explicit brackets, i.e. 'KEY=[V1,V2,V3]'. It also support nested brackets to build list/tuple values. e.g. 'KEY=[(V1,V2),(V3,V4)]'

CHAPTER

THIRTYEIGHT

MMENGINE.RUNNER

mmengine.runner

- *Runner*
- *Loop*
- *Checkpoints*
- *AMP*
- *Miscellaneous*

38.1 Runner

Runner	A training helper for PyTorch.

38.1.1 Runner

class mmengine.runner.**Runner**(*model*, *work_dir*, *train_dataloader=None*, *val_dataloader=None*, *test_dataloader=None*, *train_cfg=None*, *val_cfg=None*, *test_cfg=None*, *auto_scale_lr=None*, *optim_wrapper=None*, *param_scheduler=None*, *val_evaluator=None*, *test_evaluator=None*, *default_hooks=None*, *custom_hooks=None*, *data_preprocessor=None*, *load_from=None*, *resume=False*, *launcher='none'*, *env_cfg={'dist_cfg': {'backend': 'nccl'}}*, *log_processor=None*, *log_level='INFO'*, *visualizer=None*, *default_scope='mmengine'*, *randomness={'seed': None}*, *experiment_name=None*, *cfg=None*)

A training helper for PyTorch.

Runner object can be built from config by `runner = Runner.from_cfg(cfg)` where the `cfg` usually contains training, validation, and test-related configurations to build corresponding components. We usually use the same config to launch training, testing, and validation tasks. However, only some of these components are necessary at the same time, e.g., testing a model does not need training or validation-related components.

To avoid repeatedly modifying config, the construction of `Runner` adopts lazy initialization to only initialize components when they are going to be used. Therefore, the model is always initialized at the beginning, and training, validation, and, testing related components are only initialized when calling `runner.train()`, `runner.val()`, and `runner.test()`, respectively.

Parameters

- **model** (torch.nn.Module or dict) – The model to be run. It can be a dict used for build a model.
- **work_dir** (*str*) – The working directory to save checkpoints. The logs will be saved in the subdirectory of *work_dir* named *timestamp*.
- **train_dataloader** (*Dataloader or dict, optional*) – A dataloader object or a dict to build a dataloader. If None is given, it means skipping training steps. Defaults to None. See *build_dataloader()* for more details.
- **val_dataloader** (*Dataloader or dict, optional*) – A dataloader object or a dict to build a dataloader. If None is given, it means skipping validation steps. Defaults to None. See *build_dataloader()* for more details.
- **test_dataloader** (*Dataloader or dict, optional*) – A dataloader object or a dict to build a dataloader. If None is given, it means skipping test steps. Defaults to None. See *build_dataloader()* for more details.
- **train_cfg** (*dict, optional*) – A dict to build a training loop. If it does not provide "type" key, it should contain "by_epoch" to decide which type of training loop *EpochBasedTrainLoop* or *IterBasedTrainLoop* should be used. If train_cfg specified, *train_dataloader* should also be specified. Defaults to None. See *build_train_loop()* for more details.
- **val_cfg** (*dict, optional*) – A dict to build a validation loop. If it does not provide "type" key, *ValLoop* will be used by default. If val_cfg specified, *val_dataloader* should also be specified. If ValLoop is built with *fp16=True* `, runner.val() will be performed under fp16 precision. Defaults to None. See *build_val_loop()* for more details.
- **test_cfg** (*dict, optional*) – A dict to build a test loop. If it does not provide "type" key, *TestLoop* will be used by default. If test_cfg specified, *test_dataloader* should also be specified. If ValLoop is built with *fp16=True* `, runner.val() will be performed under fp16 precision. Defaults to None. See *build_test_loop()* for more details.
- **auto_scale_lr** (*dict, Optional*) – Config to scale the learning rate automatically. It includes base_batch_size and enable. base_batch_size is the batch size that the optimizer lr is based on. enable is the switch to turn on and off the feature.
- **optim_wrapper** (OptimWrapper *or dict, optional*) – Computing gradient of model parameters. If specified, *train_dataloader* should also be specified. If automatic mixed precision or gradient accmulation training is required. The type of optim_wrapper should be AmpOptimizerWrapper. See *build_optim_wrapper()* for examples. Defaults to None.
- **param_scheduler** (*_ParamScheduler or dict or list, optional*) – Parameter scheduler for updating optimizer parameters. If specified, optimizer should also be specified. Defaults to None. See *build_param_scheduler()* for examples.
- **val_evaluator** (Evaluator *or dict or list, optional*) – A evaluator object used for computing metrics for validation. It can be a dict or a list of dict to build a evaluator. If specified, *val_dataloader* should also be specified. Defaults to None.
- **test_evaluator** (Evaluator *or dict or list, optional*) – A evaluator object used for computing metrics for test steps. It can be a dict or a list of dict to build a evaluator. If specified, *test_dataloader* should also be specified. Defaults to None.
- **default_hooks** (*dict[str, dict] or dict[str,* Hook*], optional*) – Hooks to execute default actions like updating model parameters and saving checkpoints. Default hooks are OptimizerHook, IterTimerHook, LoggerHook, ParamSchedulerHook and CheckpointHook. Defaults to None. See *register_default_hooks()* for more details.

- **custom_hooks** (*list[dict] or list[Hook], optional*) – Hooks to execute custom actions like visualizing images processed by pipeline. Defaults to None.
- **data_preprocessor** (*dict, optional*) – The pre-process config of `BaseDataPreprocessor`. If the `model` argument is a dict and doesn't contain the key `data_preprocessor`, set the argument as the `data_preprocessor` of the `model` dict. Defaults to None.
- **load_from** (*str, optional*) – The checkpoint file to load from. Defaults to None.
- **resume** (*bool*) – Whether to resume training. Defaults to False. If `resume` is True and `load_from` is None, automatically to find latest checkpoint from `work_dir`. If not found, resuming does nothing.
- **launcher** (*str*) – Way to launcher multi-process. Supported launchers are 'pytorch', 'mpi', 'slurm' and 'none'. If 'none' is provided, non-distributed environment will be launched.
- **env_cfg** (*dict*) – A dict used for setting environment. Defaults to dict(dist_cfg=dict(backend='nccl')).
- **log_processor** (*dict, optional*) – A processor to format logs. Defaults to None.
- **log_level** (*int or str*) – The log level of MMLogger handlers. Defaults to 'INFO'.
- **visualizer** (Visualizer *or dict, optional*) – A Visualizer object or a dict build Visualizer object. Defaults to None. If not specified, default config will be used.
- **default_scope** (*str*) – Used to reset registries location. Defaults to "mmengine".
- **randomness** (*dict*) – Some settings to make the experiment as reproducible as possible like seed and deterministic. Defaults to `dict(seed=None)`. If seed is None, a random number will be generated and it will be broadcasted to all other processes if in distributed environment. If `cudnn_benchmarch` is `True` in `env_cfg` but `deterministic` is `True` in `randomness`, the value of `torch.backends.cudnn.benchmark` will be `False` finally.
- **experiment_name** (*str, optional*) – Name of current experiment. If not specified, timestamp will be used as `experiment_name`. Defaults to None.
- **cfg** (dict or Configdict or `Config`, optional) – Full config. Defaults to None.

Examples

```
>>> from mmengine.runner import Runner
>>> cfg = dict(
>>>     model=dict(type='ToyModel'),
>>>     work_dir='path/of/work_dir',
>>>     train_dataloader=dict(
>>>     dataset=dict(type='ToyDataset'),
>>>     sampler=dict(type='DefaultSampler', shuffle=True),
>>>     batch_size=1,
>>>     num_workers=0),
>>>     val_dataloader=dict(
>>>         dataset=dict(type='ToyDataset'),
>>>         sampler=dict(type='DefaultSampler', shuffle=False),
>>>        batch_size=1,
>>>        num_workers=0),
>>>     test_dataloader=dict(
>>>         dataset=dict(type='ToyDataset'),
```

(continues on next page)

(continued from previous page)

```
>>>         sampler=dict(type='DefaultSampler', shuffle=False),
>>>         batch_size=1,
>>>         num_workers=0),
>>>     auto_scale_lr=dict(base_batch_size=16, enable=False),
>>>     optim_wrapper=dict(type='OptimizerWrapper', optimizer=dict(
>>>         type='SGD', lr=0.01)),
>>>     param_scheduler=dict(type='MultiStepLR', milestones=[1, 2]),
>>>     val_evaluator=dict(type='ToyEvaluator'),
>>>     test_evaluator=dict(type='ToyEvaluator'),
>>>     train_cfg=dict(by_epoch=True, max_epochs=3, val_interval=1),
>>>     val_cfg=dict(),
>>>     test_cfg=dict(),
>>>     custom_hooks=[],
>>>     default_hooks=dict(
>>>         timer=dict(type='IterTimerHook'),
>>>         checkpoint=dict(type='CheckpointHook', interval=1),
>>>         logger=dict(type='LoggerHook'),
>>>         optimizer=dict(type='OptimizerHook', grad_clip=False),
>>>         param_scheduler=dict(type='ParamSchedulerHook')),
>>>     launcher='none',
>>>     env_cfg=dict(dist_cfg=dict(backend='nccl')),
>>>     log_processor=dict(window_size=20),
>>>     visualizer=dict(type='Visualizer',
>>>     vis_backends=[dict(type='LocalVisBackend',
>>>                        save_dir='temp_dir')])
>>>    )
>>> runner = Runner.from_cfg(cfg)
>>> runner.train()
>>> runner.test()
```

static build_dataloader(*dataloader*, *seed=None*, *diff_rank_seed=False*)

Build dataloader.

The method builds three components:

- Dataset
- Sampler
- Dataloader

An example of `dataloader`:

```
dataloader = dict(
    dataset=dict(type='ToyDataset'),
    sampler=dict(type='DefaultSampler', shuffle=True),
    batch_size=1,
    num_workers=9
)
```

Parameters

- **dataloader** (*DataLoader or dict*) – A Dataloader object or a dict to build Dataloader object. If `dataloader` is a Dataloader object, just returns itself.
- **seed** (*int, optional*) – Random seed. Defaults to None.

- **diff_rank_seed** (*bool*) – Whether or not set different seeds to different ranks. If True, the seed passed to sampler is set to None, in order to synchronize the seeds used in samplers across different ranks.

Returns DataLoader build from `dataloader_cfg`.

Return type Dataloader

build_evaluator(*evaluator*)

Build evaluator.

Examples of `evaluator`:

```
# evaluator could be a built Evaluator instance
evaluator = Evaluator(metrics=[ToyMetric()])

# evaluator can also be a list of dict
evaluator = [
    dict(type='ToyMetric1'),
    dict(type='ToyEvaluator2')
]

# evaluator can also be a list of built metric
evaluator = [ToyMetric1(), ToyMetric2()]

# evaluator can also be a dict with key metrics
evaluator = dict(metrics=ToyMetric())
# metric is a list
evaluator = dict(metrics=[ToyMetric()])
```

Parameters **evaluator** (*Evaluator or dict or list*) – An Evaluator object or a config dict or list of config dict used to build an Evaluator.

Returns Evaluator build from `evaluator`.

Return type *Evaluator*

build_log_processor(*log_processor*)

Build test log_processor.

Examples of `log_processor`:

LogProcessor will be used log_processor = dict()

custom log_processor log_processor = dict(type='CustomLogProcessor')

Parameters

- **log_processor** (*LogProcessor or dict*) – A log processor or a dict
- **build log processor. If log_processor is a log processor** (*to*) –
- **object** –
- **returns itself.** (*just*) –

Returns Log processor object build from `log_processor_cfg`.

Return type *LogProcessor*

build_logger(*log_level='INFO'*, *log_file=None*, ***kwargs*)
Build a global asscessable MMLogger.

Parameters

- **log_level** (*int or str*) – The log level of MMLogger handlers. Defaults to 'INFO'.
- **log_file** (*str, optional*) – Path of filename to save log. Defaults to None.
- ****kwargs** – Remaining parameters passed to `MMLogger`.

Returns A MMLogger object build from `logger`.

Return type *MMLogger*

build_message_hub(*message_hub=None*)
Build a global asscessable MessageHub.

Parameters **message_hub** (*dict, optional*) – A dict to build MessageHub object. If not specified, default config will be used to build MessageHub object. Defaults to None.

Returns A MessageHub object build from `message_hub`.

Return type *MessageHub*

build_model(*model*)
Build model.

If `model` is a dict, it will be used to build a nn.Module object. Else, if `model` is a nn.Module object it will be returned directly.

An example of `model`:

```
model = dict(type='ResNet')
```

Parameters **model** (*nn.Module or dict*) – A `nn.Module` object or a dict to build nn.Module object. If `model` is a nn.Module object, just returns itself.

Return type torch.nn.modules.module.Module

Note: The returned model must implement `train_step`, `test_step` if `runner.train` or `runner.test` will be called. If `runner.val` will be called or `val_cfg` is configured, model must implement *val_step*.

Returns Model build from `model`.

Return type nn.Module

Parameters **model** (*Union[torch.nn.modules.module.Module, Dict]*) –

build_optim_wrapper(*optim_wrapper*)
Build optimizer wrapper.

If `optim_wrapper` is a config dict for only one optimizer, the keys must contain `optimizer`, and `type` is optional. It will build a `OptimWrapper` by default.

If `optim_wrapper` is a config dict for multiple optimizers, i.e., it has multiple keys and each key is for an optimizer wrapper. The constructor must be specified since `DefaultOptimizerConstructor` cannot handle the building of training with multiple optimizers.

If `optim_wrapper` is a dict of pre-built optimizer wrappers, i.e., each value of `optim_wrapper` represents an `OptimWrapper` instance. `build_optim_wrapper` will directly build the `OptimWrapperDict` instance from `optim_wrapper`.

Parameters **optim_wrapper** (*OptimWrapper or dict*) – An OptimWrapper object or a dict to build OptimWrapper objects. If `optim_wrapper` is an OptimWrapper, just return an `OptimizeWrapper` instance.

Return type Union[*mmengine.optim.optimizer.optimizer_wrapper.OptimWrapper*, *mmengine.optim.optimizer.optimizer_wrapper_dict.OptimWrapperDict*]

Note: For single optimizer training, if *optim_wrapper* is a config dict, *type* is optional(defaults to `OptimWrapper`) and it must contain *optimizer* to build the corresponding optimizer.

Examples

```
>>> # build an optimizer
>>> optim_wrapper_cfg = dict(type='OptimWrapper', optimizer=dict(
...     type='SGD', lr=0.01))
>>> # optim_wrapper_cfg = dict(optimizer=dict(type='SGD', lr=0.01))
>>> # is also valid.
>>> optim_wrapper = runner.build_optim_wrapper(optim_wrapper_cfg)
>>> optim_wrapper
Type: OptimWrapper
accumulative_counts: 1
optimizer:
SGD (
Parameter Group 0
    dampening: 0
    lr: 0.01
    momentum: 0
    nesterov: False
    weight_decay: 0
)
>>> # build optimizer without `type`
>>> optim_wrapper_cfg = dict(optimizer=dict(type='SGD', lr=0.01))
>>> optim_wrapper = runner.build_optim_wrapper(optim_wrapper_cfg)
>>> optim_wrapper
Type: OptimWrapper
accumulative_counts: 1
optimizer:
SGD (
Parameter Group 0
    dampening: 0
    lr: 0.01
    maximize: False
    momentum: 0
    nesterov: False
    weight_decay: 0
)
>>> # build multiple optimizers
>>> optim_wrapper_cfg = dict(
```

(continues on next page)

(continued from previous page)

```
...     generator=dict(type='OptimWrapper', optimizer=dict(
...         type='SGD', lr=0.01)),
...     discriminator=dict(type='OptimWrapper', optimizer=dict(
...         type='Adam', lr=0.001))
...     # need to customize a multiple optimizer constructor
...     constructor='CustomMultiOptimizerConstructor',
...)
>>> optim_wrapper = runner.optim_wrapper(optim_wrapper_cfg)
>>> optim_wrapper
name: generator
Type: OptimWrapper
accumulative_counts: 1
optimizer:
SGD (
Parameter Group 0
    dampening: 0
    lr: 0.1
    momentum: 0
    nesterov: False
    weight_decay: 0
)
name: discriminator
Type: OptimWrapper
accumulative_counts: 1
optimizer:
'discriminator': Adam (
Parameter Group 0
    dampening: 0
    lr: 0.02
    momentum: 0
    nesterov: False
    weight_decay: 0
)
```

Important: If you need to build multiple optimizers, you should implement a MultiOptimWrapperConstructor which gets parameters passed to corresponding optimizers and compose the `OptimWrapperDict`. More details about how to customize OptimizerConstructor can be found at optimizer-docs.

Returns Optimizer wrapper build from `optimizer_cfg`.

Return type *OptimWrapper*

Parameters **optim_wrapper** (*Union[torch.optim.optimizer.Optimizer, mmengine.optim.optimizer.optimizer_wrapper.OptimWrapper, Dict]*) –

build_param_scheduler(*scheduler*)

Build parameter schedulers.

`build_param_scheduler` should be called after `build_optim_wrapper` because the building logic will change according to the number of optimizers built by the runner. The cases are as below:

- Single optimizer: When only one optimizer is built and used in the runner, `build_param_scheduler` will return a list of parameter schedulers.

- Multiple optimizers: When two or more optimizers are built and used in runner, `build_param_scheduler` will return a dict containing the same keys with multiple optimizers and each value is a list of parameter schedulers. Note that, if you want different optimizers to use different parameter shedulers to update optimizer's hyper-parameters, the input parameter `scheduler` also needs to be a dict and its key are consistent with multiple optimizers. Otherwise, the same parameter schedulers will be used to update optimizer's hyper-parameters.

Parameters **scheduler** (*_ParamScheduler or dict or list*) – A Param Scheduler object or a dict or list of dict to build parameter schedulers.

Return type Union[List[*mmengine.optim.scheduler.param_scheduler._ParamScheduler*], Dict[str, List[*mmengine.optim.scheduler.param_scheduler._ParamScheduler*]]]

Examples

```
>>> # build one scheduler
>>> optim_cfg = dict(dict(type='SGD', lr=0.01))
>>> runner.optim_wrapper = runner.build_optim_wrapper(
>>>     optim_cfg)
>>> scheduler_cfg = dict(type='MultiStepLR', milestones=[1, 2])
>>> schedulers = runner.build_param_scheduler(scheduler_cfg)
>>> schedulers
[<mmengine.optim.scheduler.lr_scheduler.MultiStepLR at 0x7f70f6966290>]  #␣
↪noqa: E501
```

```
>>> # build multiple schedulers
>>> scheduler_cfg = [
...     dict(type='MultiStepLR', milestones=[1, 2]),
...     dict(type='StepLR', step_size=1)
... ]
>>> schedulers = runner.build_param_scheduler(scheduler_cfg)
>>> schedulers
[<mmengine.optim.scheduler.lr_scheduler.MultiStepLR at 0x7f70f60dd3d0>,  #␣
↪noqa: E501
<mmengine.optim.scheduler.lr_scheduler.StepLR at 0x7f70f6eb6150>]
```

Above examples only provide the case of one optimizer and one scheduler or multiple shedulers. If you want to know how to set parameter scheduler when using multiple optimizers, you can find more examples optimizer-docs.

Returns List of parameter schedulers or a dictionary contains list of parameter schedulers build from `scheduler`.

Return type list[*_ParamScheduler*] or dict[str, list[*_ParamScheduler*]]

Parameters **scheduler** (*Union[mmengine.optim.scheduler.param_scheduler._ParamScheduler, Dict, List]*) –

build_test_loop(*loop*)

Build test loop.

Examples of `loop`:

```
# `TestLoop` will be used
loop = dict()
```

(continues on next page)

(continued from previous page)

```
# custom test loop
loop = dict(type='CustomTestLoop')
```

Parameters **loop** (*BaseLoop or dict*) – A test loop or a dict to build test loop. If `loop` is a test loop object, just returns itself.

Returns Test loop object build from `loop_cfg`.

Return type *BaseLoop*

build_train_loop(*loop*)

Build training loop.

Examples of `loop`:

```
# `EpochBasedTrainLoop` will be used
loop = dict(by_epoch=True, max_epochs=3)

# `IterBasedTrainLoop` will be used
loop = dict(by_epoch=False, max_epochs=3)

# custom training loop
loop = dict(type='CustomTrainLoop', max_epochs=3)
```

Parameters **loop** (*BaseLoop or dict*) – A training loop or a dict to build training loop. If `loop` is a training loop object, just returns itself.

Returns Training loop object build from `loop`.

Return type *BaseLoop*

build_val_loop(*loop*)

Build validation loop.

Examples of `loop`:

ValLoop will be used loop = dict()

custom validation loop loop = dict(type='CustomValLoop')

Parameters **loop** (*BaseLoop or dict*) – A validation loop or a dict to build validation loop. If `loop` is a validation loop object, just returns itself.

Returns Validation loop object build from `loop`.

Return type *BaseLoop*

build_visualizer(*visualizer=None*)

Build a global asscessable Visualizer.

Parameters **visualizer** (*Visualizer or dict, optional*) – A Visualizer object or a dict to build Visualizer object. If `visualizer` is a Visualizer object, just returns itself. If not specified, default config will be used to build Visualizer object. Defaults to None.

Returns A Visualizer object build from `visualizer`.

Return type *Visualizer*

call_hook(*fn_name*, ***kwargs*)
Call all hooks.

Parameters

- **fn_name** (*str*) – The function name in each hook to be called, such as "before_train_epoch".
- ****kwargs** – Keyword arguments passed to hook.

Return type None

property deterministic
Whether cudnn to select deterministic algorithms.

Type int

property distributed
Whether current environment is distributed.

Type bool

dump_config()
Dump config to *work_dir*.

Return type None

property epoch
Current epoch.

Type int

property experiment_name
Name of experiment.

Type str

classmethod from_cfg(*cfg*)
Build a runner from config.

Parameters **cfg** (*ConfigType*) – A config used for building runner. Keys of `cfg` can see `__init__()`.

Returns A runner build from `cfg`.

Return type *Runner*

property hooks
A list of registered hooks.

Type list[Hook]

property iter
Current iteration.

Type int

property launcher
Way to launcher multi processes.

Type str

load_checkpoint(*filename*, *map_location='cpu'*, *strict=False*, *revise_keys=[('^module.', '')]*)
Load checkpoint from given `filename`.

Parameters

- **filename** (*str*) – Accept local filepath, URL, `torchvision://xxx`, `open-mmlab://xxx`.
- **map_location** (*str or callable*) – A string or a callable function to specifying how to remap storage locations. Defaults to 'cpu'.
- **strict** (*bool*) – strict (bool): Whether to allow different params for the model and checkpoint.
- **revise_keys** (*list*) – A list of customized keywords to modify the state_dict in checkpoint. Each item is a (pattern, replacement) pair of the regular expression operations. Default: strip the prefix 'module.' by [(r'^module.', '')].

load_or_resume()
load or resume checkpoint.

Return type None

property max_epochs
Total epochs to train model.

Type int

property max_iters
Total iterations to train model.

Type int

property model_name
Name of the model, usually the module class name.

Type str

property rank
Rank of current process.

Type int

register_custom_hooks(*hooks*)
Register custom hooks into hook list.

Parameters **hooks** (*list[Hook | dict]*) – List of hooks or configs to be registered.

Return type None

register_default_hooks(*hooks=None*)
Register default hooks into hook list.

`hooks` will be registered into runner to execute some default actions like updating model parameters or saving checkpoints.

Default hooks and their priorities:

Hooks	Priority
RuntimeInfoHook	VERY_HIGH (10)
IterTimerHook	NORMAL (50)
DistSamplerSeedHook	NORMAL (50)
LoggerHook	BELOW_NORMAL (60)
ParamSchedulerHook	LOW (70)
CheckpointHook	VERY_LOW (90)

If `hooks` is None, above hooks will be registered by default:

```
default_hooks = dict(
    runtime_info=dict(type='RuntimeInfoHook'),
    timer=dict(type='IterTimerHook'),
    sampler_seed=dict(type='DistSamplerSeedHook'),
    logger=dict(type='LoggerHook'),
    param_scheduler=dict(type='ParamSchedulerHook'),
    checkpoint=dict(type='CheckpointHook', interval=1),
)
```

If not None, `hooks` will be merged into `default_hooks`. If there are None value in default_hooks, the corresponding item will be popped from `default_hooks`:

```
hooks = dict(timer=None)
```

The final registered default hooks will be `RuntimeInfoHook`, `DistSamplerSeedHook`, `LoggerHook`, `ParamSchedulerHook` and `CheckpointHook`.

Parameters **hooks** (*dict[str, Hook or dict], optional*) – Default hooks or configs to be registered.

Return type None

register_hook(*hook*, *priority=None*)

Register a hook into the hook list.

The hook will be inserted into a priority queue, with the specified priority (See *Priority* for details of priorities). For hooks with the same priority, they will be triggered in the same order as they are registered.

Priority of hook will be decided with the following priority:

- `priority` argument. If `priority` is given, it will be priority of hook.
- If `hook` argument is a dict and `priority` in it, the priority will be the value of `hook['priority']`.
- If `hook` argument is a dict but `priority` not in it or `hook` is an instance of `hook`, the priority will be `hook.priority`.

Parameters

- **hook** (Hook or dict) – The hook to be registered.
- **priority** (int or str or *Priority*, optional) – Hook priority. Lower value means higher priority.

Return type None

register_hooks(*default_hooks=None*, *custom_hooks=None*)

Register default hooks and custom hooks into hook list.

Parameters

- **default_hooks** (*dict[str, dict] or dict[str, Hook], optional*) – Hooks to execute default actions like updating model parameters and saving checkpoints. Defaults to None.
- **custom_hooks** (*list[dict] or list[Hook], optional*) – Hooks to execute custom actions like visualizing images processed by pipeline. Defaults to None.

Return type None

resume(*filename*, *resume_optimizer=True*, *resume_param_scheduler=True*, *map_location='default'*)

Resume model from checkpoint.

Parameters

- **filename** (*str*) – Accept local filepath, URL, `torchvision://xxx`, `open-mmlab://xxx`.
- **resume_optimizer** (*bool*) – Whether to resume optimizer state. Defaults to True.
- **resume_param_scheduler** (*bool*) – Whether to resume param scheduler state. Defaults to True.
- **map_location** (*str or callable*) – A string or a callable function to specifying how to remap storage locations. Defaults to 'default'.

Return type None

save_checkpoint(*out_dir*, *filename*, *file_client_args=None*, *save_optimizer=True*, *save_param_scheduler=True*, *meta=None*, *by_epoch=True*, *backend_args=None*)

Save checkpoints.

`CheckpointHook` invokes this method to save checkpoints periodically.

Parameters

- **out_dir** (*str*) – The directory that checkpoints are saved.
- **filename** (*str*) – The checkpoint filename.
- **file_client_args** (*dict, optional*) – Arguments to instantiate a FileClient. See *mmengine.fileio.FileClient* for details. Defaults to None. It will be deprecated in future. Please use *backend_args* instead.
- **save_optimizer** (*bool*) – Whether to save the optimizer to the checkpoint. Defaults to True.
- **save_param_scheduler** (*bool*) – Whether to save the param_scheduler to the checkpoint. Defaults to True.
- **meta** (*dict, optional*) – The meta information to be saved in the checkpoint. Defaults to None.
- **by_epoch** (*bool*) – Whether the scheduled momentum is updated by epochs. Defaults to True.
- **backend_args** (*dict, optional*) – Arguments to instantiate the preifx of uri corresponding backend. Defaults to None. New in v0.2.0.

scale_lr(*optim_wrapper*, *auto_scale_lr=None*)

Automatically scaling learning rate in training according to the ratio of `base_batch_size` in `autoscalelr_cfg` and real batch size.

It scales the learning rate linearly according to the paper.

Note: `scale_lr` must be called after building optimizer wrappers and before building parameter schedulers.

Parameters

- **optim_wrapper** (`OptimWrapper`) – An OptimWrapper object whose parameter groups' learning rate need to be scaled.

- **auto_scale_lr** (*Dict, Optional*) – Config to scale the learning rate automatically. It includes `base_batch_size` and `enable`. `base_batch_size` is the batch size that the optimizer lr is based on. `enable` is the switch to turn on and off the feature.

Return type None

property seed
A number to set random modules.

Type int

set_randomness(*seed*, *diff_rank_seed=False*, *deterministic=False*)
Set random seed to guarantee reproducible results.

Parameters

- **seed** (*int*) – A number to set random modules.
- **diff_rank_seed** (*bool*) – Whether or not set different seeds according to global rank. Defaults to False.
- **deterministic** (*bool*) – Whether to set the deterministic option for CUDNN backend, i.e., set *torch.backends.cudnn.deterministic* to True and *torch.backends.cudnn.benchmark* to False. Defaults to False. See https://pytorch.org/docs/stable/notes/randomness.html for more details.

Return type None

setup_env(*env_cfg*)
Setup environment.

An example of `env_cfg`:

```
env_cfg = dict(
    cudnn_benchmark=True,
    mp_cfg=dict(
        mp_start_method='fork',
        opencv_num_threads=0
    ),
    dist_cfg=dict(backend='nccl'),
    resource_limit=4096
)
```

Parameters **env_cfg** (*dict*) – Config for setting environment.

Return type None

test()
Launch test.

Returns A dict of metrics on testing set.

Return type dict

property test_dataloader
The data loader for testing.

property test_evaluator
An evaluator for testing.

Type `Evaluator`

property test_loop
A loop to run testing.

Type *BaseLoop*

property timestamp
Timestamp when creating experiment.

Type str

train()
Launch training.

Returns The model after training.

Return type nn.Module

property train_dataloader
The data loader for training.

property train_loop
A loop to run training.

Type *BaseLoop*

val()
Launch validation.

Returns A dict of metrics on validation set.

Return type dict

property val_begin
The epoch/iteration to start running validation during training.

Type int

property val_dataloader
The data loader for validation.

property val_evaluator
An evaluator for validation.

Type Evaluator

property val_interval
Interval to run validation during training.

Type int

property val_loop
A loop to run validation.

Type *BaseLoop*

property work_dir
The working directory to save checkpoints and logs.

Type str

property world_size
Number of processes participating in the job.

Type int

wrap_model(*model_wrapper_cfg*, *model*)

Wrap the model to :obj:`MMDistributedDataParallel` or other custom distributed data-parallel module wrappers.

An example of `model_wrapper_cfg`:

```
model_wrapper_cfg = dict(
    broadcast_buffers=False,
    find_unused_parameters=False
)
```

Parameters

- **model_wrapper_cfg** (*dict, optional*) – Config to wrap model. If not specified, `DistributedDataParallel` will be used in distributed environment. Defaults to None.
- **model** (*nn.Module*) – Model to be wrapped.

Returns nn.Module or subclass of `DistributedDataParallel`.

Return type nn.Module or DistributedDataParallel

38.2 Loop

BaseLoop	Base loop class.
EpochBasedTrainLoop	Loop for epoch-based training.
IterBasedTrainLoop	Loop for iter-based training.
ValLoop	Loop for validation.
TestLoop	Loop for test.

38.2.1 BaseLoop

class mmengine.runner.**BaseLoop**(*runner*, *dataloader*)

Base loop class.

All subclasses inherited from `BaseLoop` should overwrite the *run()* method.

Parameters

- **runner** (Runner) – A reference of runner.
- **dataloader** (*Dataloader or dict*) – An iterator to generate one batch of dataset each iteration.

Return type None

abstract run()

Execute loop.

Return type Any

38.2.2 EpochBasedTrainLoop

class mmengine.runner.**EpochBasedTrainLoop**(*runner*, *dataloader*, *max_epochs*, *val_begin=1*, *val_interval=1*, *dynamic_intervals=None*)

Loop for epoch-based training.

Parameters

- **runner** (Runner) – A reference of runner.
- **dataloader** (*Dataloader or dict*) – A dataloader object or a dict to build a dataloader.
- **max_epochs** (*int*) – Total training epochs.
- **val_begin** (*int*) – The epoch that begins validating. Defaults to 1.
- **val_interval** (*int*) – Validation interval. Defaults to 1.
- **dynamic_intervals** (*List[Tuple[int, int]], optional*) – The first element in the tuple is a milestone and the second element is a interval. The interval is used after the corresponding milestone. Defaults to None.

Return type None

property **epoch**

Current epoch.

Type int

property **iter**

Current iteration.

Type int

property **max_epochs**

Total epochs to train model.

Type int

property **max_iters**

Total iterations to train model.

Type int

run()

Launch training.

Return type torch.nn.modules.module.Module

run_epoch()

Iterate one epoch.

Return type None

run_iter(*idx*, *data_batch*)

Iterate one min-batch.

Parameters **data_batch** (*Sequence[dict]*) – Batch of data from dataloader.

Return type None

38.2.3 IterBasedTrainLoop

class mmengine.runner.**IterBasedTrainLoop**(*runner*, *dataloader*, *max_iters*, *val_begin=1*, *val_interval=1000*, *dynamic_intervals=None*)

Loop for iter-based training.

Parameters

- **runner** (Runner) – A reference of runner.
- **dataloader** (*Dataloader or dict*) – A dataloader object or a dict to build a dataloader.
- **max_iters** (*int*) – Total training iterations.
- **val_begin** (*int*) – The iteration that begins validating. Defaults to 1.
- **val_interval** (*int*) – Validation interval. Defaults to 1000.
- **dynamic_intervals** (*List[Tuple[int, int]], optional*) – The first element in the tuple is a milestone and the second element is a interval. The interval is used after the corresponding milestone. Defaults to None.

Return type None

property **epoch**

Current epoch.

Type int

property **iter**

Current iteration.

Type int

property **max_epochs**

Total epochs to train model.

Type int

property **max_iters**

Total iterations to train model.

Type int

run()

Launch training.

Return type None

run_iter(*data_batch*)

Iterate one mini-batch.

Parameters **data_batch** (*Sequence[dict]*) – Batch of data from dataloader.

Return type None

38.2.4 ValLoop

class mmengine.runner.**ValLoop**(*runner*, *dataloader*, *evaluator*, *fp16=False*)
Loop for validation.

Parameters

- **runner** (Runner) – A reference of runner.
- **dataloader** (*Dataloader or dict*) – A dataloader object or a dict to build a dataloader.
- **evaluator** (Evaluator *or dict or list*) – Used for computing metrics.
- **fp16** (*bool*) – Whether to enable fp16 validation. Defaults to False.

Return type None

run()
Launch validation.

Return type dict

run_iter(*idx*, *data_batch*)
Iterate one mini-batch.

Parameters **data_batch** (*Sequence[dict]*) – Batch of data from dataloader.

38.2.5 TestLoop

class mmengine.runner.**TestLoop**(*runner*, *dataloader*, *evaluator*, *fp16=False*)
Loop for test.

Parameters

- **runner** (Runner) – A reference of runner.
- **dataloader** (*Dataloader or dict*) – A dataloader object or a dict to build a dataloader.
- **evaluator** (Evaluator *or dict or list*) – Used for computing metrics.
- **fp16** (*bool*) – Whether to enable fp16 testing. Defaults to False.

run()
Launch test.

Return type dict

run_iter(*idx*, *data_batch*)
Iterate one mini-batch.

Parameters **data_batch** (*Sequence[dict]*) – Batch of data from dataloader.

Return type None

38.3 Checkpoints

CheckpointLoader	A general checkpoint loader to manage all schemes.

38.3.1 CheckpointLoader

class mmengine.runner.**CheckpointLoader**

A general checkpoint loader to manage all schemes.

classmethod load_checkpoint(*filename*, *map_location=None*, *logger=None*)

load checkpoint through URL scheme path.

Parameters

- **filename** (*str*) – checkpoint file name with given prefix
- **map_location** (*str, optional*) – Same as `torch.load()`. Default: None
- **logger** (`logging.Logger`, optional) – The logger for message. Default: None

Returns The loaded checkpoint.

Return type dict or OrderedDict

classmethod register_scheme(*prefixes*, *loader=None*, *force=False*)

Register a loader to CheckpointLoader.

This method can be used as a normal class method or a decorator.

Parameters

- **prefixes** (*str or list[str] or tuple[str]*) –
- **prefix of the registered loader.** (*The*) –
- **loader** (*function, optional*) – The loader function to be registered. When this method is used as a decorator, loader is None. Defaults to None.
- **force** (*bool, optional*) – Whether to override the loader if the prefix has already been registered. Defaults to False.

find_latest_checkpoint	Find the latest checkpoint from the given path.
get_deprecated_model_names	
get_external_models	
get_mmcls_models	
get_state_dict	Returns a dictionary containing a whole state of the module.
get_torchvision_models	
load_checkpoint	Load checkpoint from a file or URI.
load_state_dict	Load state_dict to a module.
save_checkpoint	Save checkpoint to file.
weights_to_cpu	Copy a model state_dict to cpu.

38.3.2 mmengine.runner.find_latest_checkpoint

mmengine.runner.**find_latest_checkpoint**(*path*)

Find the latest checkpoint from the given path.

Refer to https://github.com/facebookresearch/fvcore/blob/main/fvcore/common/checkpoint.py # noqa: E501

Parameters **path** (*str*) – The path to find checkpoints.

Returns File path of the latest checkpoint.

Return type str or None

38.3.3 mmengine.runner.get_deprecated_model_names

mmengine.runner.**get_deprecated_model_names**()

38.3.4 mmengine.runner.get_external_models

mmengine.runner.**get_external_models**()

38.3.5 mmengine.runner.get_mmcls_models

mmengine.runner.**get_mmcls_models**()

38.3.6 mmengine.runner.get_state_dict

mmengine.runner.**get_state_dict**(*module*, *destination=None*, *prefix=''*, *keep_vars=False*)

Returns a dictionary containing a whole state of the module.

Both parameters and persistent buffers (e.g. running averages) are included. Keys are corresponding parameter and buffer names. This method is modified from `torch.nn.Module.state_dict()` to recursively check parallel module in case that the model has a complicated structure, e.g., nn.Module(nn.Module(DDP)).

Parameters

- **module** (*nn.Module*) – The module to generate state_dict.
- **destination** (*OrderedDict*) – Returned dict for the state of the module.
- **prefix** (*str*) – Prefix of the key.
- **keep_vars** (*bool*) – Whether to keep the variable property of the parameters. Default: False.

Returns A dictionary containing a whole state of the module.

Return type dict

38.3.7 mmengine.runner.get_torchvision_models

mmengine.runner.**get_torchvision_models**()

38.3.8 mmengine.runner.load_checkpoint

mmengine.runner.**load_checkpoint**(*model*, *filename*, *map_location=None*, *strict=False*, *logger=None*, *revise_keys=[('^module\\.', '')]*)

Load checkpoint from a file or URI.

Parameters

- **model** (*Module*) – Module to load checkpoint.
- **filename** (*str*) – Accept local filepath, URL, `torchvision://xxx`, `open-mmlab://xxx`. Please refer to `docs/model_zoo.md` for details.
- **map_location** (*str*) – Same as `torch.load()`.
- **strict** (*bool*) – Whether to allow different params for the model and checkpoint.
- **logger** (`logging.Logger` or None) – The logger for error message.
- **revise_keys** (*list*) – A list of customized keywords to modify the state_dict in checkpoint. Each item is a (pattern, replacement) pair of the regular expression operations. Default: strip the prefix 'module.' by [(r'^module.', '')].

Returns The loaded checkpoint.

Return type dict or OrderedDict

38.3.9 mmengine.runner.load_state_dict

mmengine.runner.**load_state_dict**(*module*, *state_dict*, *strict=False*, *logger=None*)

Load state_dict to a module.

This method is modified from `torch.nn.Module.load_state_dict()`. Default value for `strict` is set to `False` and the message for param mismatch will be shown even if strict is False.

Parameters

- **module** (*Module*) – Module that receives the state_dict.
- **state_dict** (*OrderedDict*) – Weights.
- **strict** (*bool*) – whether to strictly enforce that the keys in `state_dict` match the keys returned by this module's `state_dict()` function. Default: `False`.
- **logger** (`logging.Logger`, optional) – Logger to log the error message. If not specified, print function will be used.

38.3.10 mmengine.runner.save_checkpoint

mmengine.runner.**save_checkpoint**(*checkpoint*, *filename*, *file_client_args=None*, *backend_args=None*)
Save checkpoint to file.

Parameters

- **checkpoint** (*dict*) – Module whose params are to be saved.
- **filename** (*str*) – Checkpoint filename.
- **file_client_args** (*dict, optional*) – Arguments to instantiate a FileClient. See *mmengine.fileio.FileClient* for details. Defaults to None. It will be deprecated in future. Please use *backend_args* instead.
- **backend_args** (*dict, optional*) – Arguments to instantiate the preifx of uri corresponding backend. Defaults to None. New in v0.2.0.

38.3.11 mmengine.runner.weights_to_cpu

mmengine.runner.**weights_to_cpu**(*state_dict*)
Copy a model state_dict to cpu.

Parameters **state_dict** (*OrderedDict*) – Model weights on GPU.

Returns Model weights on GPU.

Return type OrderedDict

38.4 AMP

autocast	A wrapper of `torch.autocast` and `toch.cuda.amp.autocast`.

38.4.1 mmengine.runner.autocast

mmengine.runner.**autocast**(*device_type=None*, *dtype=None*, *enabled=True*, *cache_enabled=None*)
A wrapper of `torch.autocast` and `toch.cuda.amp.autocast`.

Pytorch 1.5.0 provide `torch.cuda.amp.autocast` for running in mixed precision , and update it to `torch.autocast` in 1.10.0. Both interfaces have different arguments, and `torch.autocast` support running with cpu additionally.

This function provides a unified interface by wrapping `torch.autocast` and `torch.cuda.amp.autocast`, which resolves the compatibility issues that `torch.cuda.amp.autocast` does not support running mixed precision with cpu, and both contexts have different arguments. We suggest users using this function in the code to achieve maximized compatibility of different PyTorch versions.

Note: `autocast` requires pytorch version >= 1.5.0. If pytorch version <= 1.10.0 and cuda is not available, it will raise an error with `enabled=True`, since `torch.cuda.amp.autocast` only support cuda mode.

Examples

```
>>> # case1: 1.10 > Pytorch version >= 1.5.0
>>> with autocast():
>>>    # run in mixed precision context
>>>    pass
>>> with autocast(device_type='cpu')::
>>>    # raise error, torch.cuda.amp.autocast only support cuda mode.
>>>    pass
>>> # case2: Pytorch version >= 1.10.0
>>> with autocast():
>>>    # default cuda mixed precision context
>>>    pass
>>> with autocast(device_type='cpu'):
>>>    # cpu mixed precision context
>>>    pass
>>> with autocast(
>>>     device_type='cuda', enabled=True, cache_enabled=True):
>>>    # enable precision context with more specific arguments.
>>>    pass
```

Parameters

- **device_type** (*str, required*) – Whether to use 'cuda' or 'cpu' device.
- **enabled** (*bool*) – Whether autocasting should be enabled in the region. Defaults to True
- **dtype** (*torch_dtype, optional*) – Whether to use `torch.float16` or `torch.bfloat16`.
- **cache_enabled** (*bool, optional*) – Whether the weight cache inside autocast should be enabled.

38.5 Miscellaneous

LogProcessor	A log processor used to format log information collected from `runner.message_hub.log_scalars`.
Priority	Hook priority levels.

38.5.1 LogProcessor

class mmengine.runner.**LogProcessor**(*window_size=10*, *by_epoch=True*, *custom_cfg=None*, *num_digits=4*)

A log processor used to format log information collected from `runner.message_hub.log_scalars`.

`LogProcessor` instance is built by runner and will format `runner.message_hub.log_scalars` to `tag` and `log_str`, which can directly used by `LoggerHook` and `MMLogger`. Besides, the argument `custom_cfg` of constructor can control the statistics method of logs.

Parameters

- **window_size** (*int*) – default smooth interval Defaults to 10.
- **by_epoch** (*bool*) – Whether to format logs with epoch stype. Defaults to True.

- **custom_cfg** (*list[dict], optional*) – Contains multiple log config dict, in which key means the data source name of log and value means the statistic method and corresponding arguments used to count the data source. Defaults to None.
  - If custom_cfg is None, all logs will be formatted via default methods, such as smoothing loss by default window_size. If custom_cfg is defined as a list of config dict, for example: [dict(data_src=loss, method='mean', log_name='global_loss', window_size='global')]. It means the log item `loss` will be counted as global mean and additionally logged as `global_loss` (defined by `log_name`). If `log_name` is not defined in config dict, the original logged key will be overwritten.
  - The original log item cannot be overwritten twice. Here is an error example: [dict(data_src=loss, method='mean', window_size='global'), dict(data_src=loss, method='mean', window_size='epoch')]. Both log config dict in custom_cfg do not have `log_name` key, which means the loss item will be overwritten twice.
  - For those statistic methods with the `window_size` argument, if `by_epoch` is set to False, `windows_size` should not be *epoch* to statistics log value by epoch.
- **num_digits** (*int*) – The number of significant digit shown in the logging message.

Examples

```
>>> # `log_name` is defined, `loss_large_window` will be an additional
>>> # record.
>>> log_processor = dict(
>>>     window_size=10,
>>>     by_epoch=True,
>>>     custom_cfg=[dict(data_src='loss',
>>>                      log_name='loss_large_window',
>>>                      method_name='mean',
>>>                      window_size=100)])
>>> # `log_name` is not defined. `loss` will be overwritten.
>>> log_processor = dict(
>>>     window_size=10,
>>>     by_epoch=True,
>>>     custom_cfg=[dict(data_src='loss',
>>>                      method_name='mean',
>>>                      window_size=100)])
>>> # Record loss with different statistics methods.
>>> log_processor = dict(
>>>     window_size=10,
>>>     by_epoch=True,
>>>     custom_cfg=[dict(data_src='loss',
>>>                      log_name='loss_large_window',
>>>                      method_name='mean',
>>>                      window_size=100),
>>>                 dict(data_src='loss',
>>>                      method_name='mean',
>>>                      window_size=100)])
>>> # Overwrite loss item twice will raise an error.
>>> log_processor = dict(
>>>     window_size=10,
>>>     by_epoch=True,
```

(continues on next page)

(continued from previous page)

```
>>>     custom_cfg=[dict(data_src='loss',
>>>                      method_name='mean',
>>>                      window_size=100),
>>>                 dict(data_src='loss',
>>>                      method_name='max',
>>>                      window_size=100)])
AssertionError
```

get_log_after_epoch(*runner*, *batch_idx*, *mode*)
Format log string after validation or testing epoch.

Parameters

- **runner** (Runner) – The runner of validation/testing phase.
- **batch_idx** (*int*) – The index of the current batch in the current loop.
- **mode** (*str*) – Current mode of runner.

Returns Formatted log dict/string which will be recorded by `runner.message_hub` and `runner.visualizer`.

Return type Tuple(dict, str)

get_log_after_iter(*runner*, *batch_idx*, *mode*)
Format log string after training, validation or testing epoch.

Parameters

- **runner** (Runner) – The runner of training phase.
- **batch_idx** (*int*) – The index of the current batch in the current loop.
- **mode** (*str*) – Current mode of runner, train, test or val.

Returns Formatted log dict/string which will be recorded by `runner.message_hub` and `runner.visualizer`.

Return type Tuple(dict, str)

38.5.2 Priority

class mmengine.runner.**Priority**(*value*)
Hook priority levels.

Level	Value
HIGHEST	0
VERY_HIGH	10
HIGH	30
ABOVE_NORMAL	40
NORMAL	50
BELOW_NORMAL	60
LOW	70
VERY_LOW	90
LOWEST	100

get_priority	Get priority value.

38.5.3 mmengine.runner.get_priority

mmengine.runner.**get_priority**(*priority*)

Get priority value.

Parameters **priority** (int or str or *Priority*) – Priority.

Returns The priority value.

Return type int

CHAPTER

THIRTYNINE

MMENGINE.HOOKS

Hook	Base hook class.
CheckpointHook	Save checkpoints periodically.
EMAHook	A Hook to apply Exponential Moving Average (EMA) on the model during training.
LoggerHook	Collect logs from different components of `Runner` and write them to terminal, JSON file, tensorboard and wandb .etc.
NaiveVisualizationHook	Show or Write the predicted results during the process of testing.
ParamSchedulerHook	A hook to update some hyper-parameters in optimizer, e.g., learning rate and momentum.
RuntimeInfoHook	A hook that updates runtime information into message hub.
DistSamplerSeedHook	Data-loading sampler for distributed training.
IterTimerHook	A hook that logs the time spent during iteration.
SyncBuffersHook	Synchronize model buffers such as running_mean and running_var in BN at the end of each epoch.
EmptyCacheHook	Releases all unoccupied cached GPU memory during the process of training.

39.1 Hook

class mmengine.hooks.**Hook**

Base hook class.

All hooks should inherit from this class.

after_load_checkpoint(*runner*, *checkpoint*)

All subclasses should override this method, if they need any operations after loading the checkpoint.

Parameters

- **runner** (Runner) – The runner of the training, validation or testing process.
- **checkpoint** (*dict*) – Model's checkpoint.

Return type None

after_run(*runner*)

All subclasses should override this method, if they need any operations before the training validation or testing process.

Parameters **runner** (Runner) – The runner of the training, validation or testing process.

Return type None

after_test(*runner*)
All subclasses should override this method, if they need any operations after testing.

Parameters **runner** (Runner) – The runner of the testing process.

Return type None

after_test_epoch(*runner*, *metrics=None*)
All subclasses should override this method, if they need any operations after each test epoch.

Parameters

- **runner** (Runner) – The runner of the testing process.
- **metrics** (*Dict[str, float], optional*) – Evaluation results of all metrics on test dataset. The keys are the names of the metrics, and the values are corresponding results.

Return type None

after_test_iter(*runner*, *batch_idx*, *data_batch=None*, *outputs=None*)
All subclasses should override this method, if they need any operations after each test iteration.

Parameters

- **runner** (Runner) – The runner of the training process.
- **batch_idx** (*int*) – The index of the current batch in the test loop.
- **data_batch** (*dict or tuple or list, optional*) – Data from dataloader.
- **outputs** (*Sequence, optional*) – Outputs from model.

Return type None

after_train(*runner*)
All subclasses should override this method, if they need any operations after train.

Parameters **runner** (Runner) – The runner of the training process.

Return type None

after_train_epoch(*runner*)
All subclasses should override this method, if they need any operations after each training epoch.

Parameters **runner** (Runner) – The runner of the training process.

Return type None

after_train_iter(*runner*, *batch_idx*, *data_batch=None*, *outputs=None*)
All subclasses should override this method, if they need any operations after each training iteration.

Parameters

- **runner** (Runner) – The runner of the training process.
- **batch_idx** (*int*) – The index of the current batch in the train loop.
- **data_batch** (*dict tuple or list, optional*) – Data from dataloader.
- **outputs** (*dict, optional*) – Outputs from model.

Return type None

after_val(*runner*)
All subclasses should override this method, if they need any operations after validation.

Parameters **runner** (Runner) – The runner of the validation process.

Return type None

after_val_epoch(*runner*, *metrics=None*)
All subclasses should override this method, if they need any operations after each validation epoch.

Parameters

- **runner** (Runner) – The runner of the validation process.
- **metrics** (*Dict[str, float], optional*) – Evaluation results of all metrics on validation dataset. The keys are the names of the metrics, and the values are corresponding results.

Return type None

after_val_iter(*runner*, *batch_idx*, *data_batch=None*, *outputs=None*)
All subclasses should override this method, if they need any operations after each validation iteration.

Parameters

- **runner** (Runner) – The runner of the validation process.
- **batch_idx** (*int*) – The index of the current batch in the val loop.
- **data_batch** (*dict or tuple or list, optional*) – Data from dataloader.
- **outputs** (*Sequence, optional*) – Outputs from model.

Return type None

before_run(*runner*)
All subclasses should override this method, if they need any operations before the training validation or testing process.

Parameters **runner** (Runner) – The runner of the training, validation or testing process.

Return type None

before_save_checkpoint(*runner*, *checkpoint*)
All subclasses should override this method, if they need any operations before saving the checkpoint.

Parameters

- **runner** (Runner) – The runner of the training, validation or testing process.
- **checkpoint** (*dict*) – Model's checkpoint.

Return type None

before_test(*runner*)
All subclasses should override this method, if they need any operations before testing.

Parameters **runner** (Runner) – The runner of the testing process.

Return type None

before_test_epoch(*runner*)
All subclasses should override this method, if they need any operations before each test epoch.

Parameters **runner** (Runner) – The runner of the testing process.

Return type None

before_test_iter(*runner*, *batch_idx*, *data_batch=None*)
All subclasses should override this method, if they need any operations before each test iteration.

Parameters

- **runner** (Runner) – The runner of the testing process.
- **batch_idx** (*int*) – The index of the current batch in the test loop.
- **data_batch** (*dict or tuple or list, optional*) – Data from dataloader. Defaults to None.

Return type None

before_train(*runner*)
All subclasses should override this method, if they need any operations before train.

Parameters **runner** (Runner) – The runner of the training process.

Return type None

before_train_epoch(*runner*)
All subclasses should override this method, if they need any operations before each training epoch.

Parameters **runner** (Runner) – The runner of the training process.

Return type None

before_train_iter(*runner*, *batch_idx*, *data_batch=None*)
All subclasses should override this method, if they need any operations before each training iteration.

Parameters

- **runner** (Runner) – The runner of the training process.
- **batch_idx** (*int*) – The index of the current batch in the train loop.
- **data_batch** (*dict or tuple or list, optional*) – Data from dataloader.

Return type None

before_val(*runner*)
All subclasses should override this method, if they need any operations before validation.

Parameters **runner** (Runner) – The runner of the validation process.

Return type None

before_val_epoch(*runner*)
All subclasses should override this method, if they need any operations before each validation epoch.

Parameters **runner** (Runner) – The runner of the validation process.

Return type None

before_val_iter(*runner*, *batch_idx*, *data_batch=None*)
All subclasses should override this method, if they need any operations before each validation iteration.

Parameters

- **runner** (Runner) – The runner of the validation process.
- **batch_idx** (*int*) – The index of the current batch in the val loop.
- **data_batch** (*dict, optional*) – Data from dataloader. Defaults to None.

Return type None

end_of_epoch(*dataloader*, *batch_idx*)
Check whether the current iteration reaches the last iteration of the dataloader.

Parameters

- **dataloader** (*Dataloader*) – The dataloader of the training, validation or testing process.
- **batch_idx** (*int*) – The index of the current batch in the loop.

Returns Whether reaches the end of current epoch or not.

Return type bool

every_n_epochs(*runner*, *n*)
Test whether current epoch can be evenly divided by n.

Parameters

- **runner** (Runner) – The runner of the training, validation or testing process.
- **n** (*int*) – Whether current epoch can be evenly divided by n.

Returns Whether current epoch can be evenly divided by n.

Return type bool

every_n_inner_iters(*batch_idx*, *n*)
Test whether current inner iteration can be evenly divided by n.

Parameters

- **batch_idx** (*int*) – Current batch index of the training, validation or testing loop.
- **n** (*int*) – Whether current inner iteration can be evenly divided by n.

Returns Whether current inner iteration can be evenly divided by n.

Return type bool

every_n_train_iters(*runner*, *n*)
Test whether current training iteration can be evenly divided by n.

Parameters

- **runner** (Runner) – The runner of the training, validation or testing process.
- **n** (*int*) – Whether current iteration can be evenly divided by n.

Returns Return True if the current iteration can be evenly divided by n, otherwise False.

Return type bool

is_last_train_epoch(*runner*)
Test whether current epoch is the last train epoch.

Parameters **runner** (Runner) – The runner of the training process.

Returns Whether reaches the end of training epoch.

Return type bool

is_last_train_iter(*runner*)
Test whether current iteration is the last train iteration.

Parameters **runner** (Runner) – The runner of the training process.

Returns Whether current iteration is the last train iteration.

Return type bool

39.2 CheckpointHook

class mmengine.hooks.**CheckpointHook**(*interval=- 1*, *by_epoch=True*, *save_optimizer=True*, *save_param_scheduler=True*, *out_dir=None*, *max_keep_ckpts=- 1*, *save_last=True*, *save_best=None*, *rule=None*, *greater_keys=None*, *less_keys=None*, *file_client_args=None*, *filename_tmpl=None*, *backend_args=None*, ***kwargs*)

Save checkpoints periodically.

Parameters

- **interval** (*int*) – The saving period. If `by_epoch=True`, interval indicates epochs, otherwise it indicates iterations. Defaults to -1, which means "never".
- **by_epoch** (*bool*) – Saving checkpoints by epoch or by iteration. Defaults to True.
- **save_optimizer** (*bool*) – Whether to save optimizer state_dict in the checkpoint. It is usually used for resuming experiments. Defaults to True.
- **save_param_scheduler** (*bool*) – Whether to save param_scheduler state_dict in the checkpoint. It is usually used for resuming experiments. Defaults to True.
- **out_dir** (*str, Path, Optional*) – The root directory to save checkpoints. If not specified, `runner.work_dir` will be used by default. If specified, the `out_dir` will be the concatenation of `out_dir` and the last level directory of `runner.work_dir`. For example, if the input `our_dir` is `./tmp` and `runner.work_dir` is `./work_dir/cur_exp`, then the ckpt will be saved in `./tmp/cur_exp`. Defaults to None.
- **max_keep_ckpts** (*int*) – The maximum checkpoints to keep. In some cases we want only the latest few checkpoints and would like to delete old ones to save the disk space. Defaults to -1, which means unlimited.
- **save_last** (*bool*) – Whether to force the last checkpoint to be saved regardless of interval. Defaults to True.
- **save_best** (*str, List[str], optional*) – If a metric is specified, it would measure the best checkpoint during evaluation. If a list of metrics is passed, it would measure a group of best checkpoints corresponding to the passed metrics. The information about best checkpoint(s) would be saved in `runner.message_hub` to keep best score value and best checkpoint path, which will be also loaded when resuming checkpoint. Options are the evaluation metrics on the test dataset. e.g., `bbox_mAP`, `segm_mAP` for bbox detection and instance segmentation. `AR@100` for proposal recall. If `save_best` is `auto`, the first key of the returned `OrderedDict` result will be used. Defaults to None.
- **rule** (*str, List[str], optional*) – Comparison rule for best score. If set to None, it will infer a reasonable rule. Keys such as 'acc', 'top' .etc will be inferred by 'greater' rule. Keys contain 'loss' will be inferred by 'less' rule. If `save_best` is a list of metrics and `rule` is a str, all metrics in `save_best` will share the comparison rule. If `save_best` and `rule` are both lists, their length must be the same, and metrics in `save_best` will use the corresponding comparison rule in `rule`. Options are 'greater', 'less', None and list which contains 'greater' and 'less'. Defaults to None.
- **greater_keys** (*List[str], optional*) – Metric keys that will be inferred by 'greater' comparison rule. If `None`, _default_greater_keys will be used. Defaults to None.
- **less_keys** (*List[str], optional*) – Metric keys that will be inferred by 'less' comparison rule. If `None`, _default_less_keys will be used. Defaults to None.

- **file_client_args** (*dict, optional*) – Arguments to instantiate a FileClient. See *mmengine.fileio.FileClient* for details. Defaults to None. It will be deprecated in future. Please use `backend_args` instead.
- **filename_tmpl** (*str, optional*) – String template to indicate checkpoint name. If specified, must contain one and only one "{}", which will be replaced with `epoch + 1` if `by_epoch=True` else `iteration + 1`. Defaults to None, which means "epoch_{}.pth" or "iter_{}.pth" accordingly.
- **backend_args** (*dict, optional*) – Arguments to instantiate the preifx of uri corresponding backend. Defaults to None. New in v0.2.0.

Return type None

Examples

```
>>> # Save best based on single metric
>>> CheckpointHook(interval=2, by_epoch=True, save_best='acc',
>>>                rule='less')
>>> # Save best based on multi metrics with the same comparison rule
>>> CheckpointHook(interval=2, by_epoch=True,
>>>                save_best=['acc', 'mIoU'], rule='greater')
>>> # Save best based on multi metrics with different comparison rule
>>> CheckpointHook(interval=2, by_epoch=True,
>>>                save_best=['FID', 'IS'], rule=['less', 'greater'])
```

after_train_epoch(*runner*)

Save the checkpoint and synchronize buffers after each epoch.

Parameters **runner** (Runner) – The runner of the training process.

Return type None

after_train_iter(*runner*, *batch_idx*, *data_batch=None*, *outputs=typing.Union[dict, NoneType]*)

Save the checkpoint and synchronize buffers after each iteration.

Parameters

- **runner** (Runner) – The runner of the training process.
- **batch_idx** (*int*) – The index of the current batch in the train loop.
- **data_batch** (*dict or tuple or list, optional*) – Data from dataloader.
- **outputs** (*dict, optional*) – Outputs from model.

Return type None

after_val_epoch(*runner*, *metrics*)

Save the checkpoint and synchronize buffers after each evaluation epoch.

Parameters

- **runner** (Runner) – The runner of the training process.
- **metrics** (*dict*) – Evaluation results of all metrics

before_train(*runner*)

Finish all operations, related to checkpoint.

This function will get the appropriate file client, and the directory to save these checkpoints of the model.

Parameters **runner** (Runner) – The runner of the training process.

Return type None

39.3 EMAHook

class mmengine.hooks.**EMAHook**(*ema_type='ExponentialMovingAverage'*, *strict_load=False*, *begin_iter=0*, *begin_epoch=0*, ***kwargs*)

A Hook to apply Exponential Moving Average (EMA) on the model during training.

Note:

- EMAHook takes priority over CheckpointHook.
- The original model parameters are actually saved in ema field after train.
- `begin_iter` and `begin_epoch` cannot be set at the same time.

Parameters

- **ema_type** (*str*) – The type of EMA strategy to use. You can find the supported strategies in `mmengine.model.averaged_model`. Defaults to 'ExponentialMovingAverage'.
- **strict_load** (*bool*) – Whether to strictly enforce that the keys of `state_dict` in checkpoint match the keys returned by `self.module.state_dict`. Defaults to False. Changed in v0.3.0.
- **begin_iter** (*int*) – The number of iteration to enable `EMAHook`. Defaults to 0.
- **begin_epoch** (*int*) – The number of epoch to enable `EMAHook`. Defaults to 0.
- ****kwargs** – Keyword arguments passed to subclasses of `BaseAveragedModel`

after_load_checkpoint(*runner*, *checkpoint*)

Resume ema parameters from checkpoint.

Parameters

- **runner** (Runner) – The runner of the testing process.
- **checkpoint** (*dict*) –

Return type None

after_test_epoch(*runner*, *metrics=None*)

We recover source model's parameter from ema model after test.

Parameters

- **runner** (Runner) – The runner of the testing process.
- **metrics** (*Dict[str, float], optional*) – Evaluation results of all metrics on test dataset. The keys are the names of the metrics, and the values are corresponding results.

Return type None

after_train_iter(*runner*, *batch_idx*, *data_batch=None*, *outputs=None*)

Update ema parameter.

Parameters

- **runner** (Runner) – The runner of the training process.
- **batch_idx** (*int*) – The index of the current batch in the train loop.
- **data_batch** (*Sequence[dict], optional*) – Data from dataloader. Defaults to None.
- **outputs** (*dict, optional*) – Outputs from model. Defaults to None.

Return type None

after_val_epoch(*runner*, *metrics=None*)
We recover source model's parameter from ema model after validation.

Parameters

- **runner** (Runner) – The runner of the validation process.
- **metrics** (*Dict[str, float], optional*) – Evaluation results of all metrics on validation dataset. The keys are the names of the metrics, and the values are corresponding results.

Return type None

before_run(*runner*)
Create an ema copy of the model.

Parameters **runner** (Runner) – The runner of the training process.

Return type None

before_save_checkpoint(*runner*, *checkpoint*)
Save ema parameters to checkpoint.

Parameters

- **runner** (Runner) – The runner of the testing process.
- **checkpoint** (*dict*) –

Return type None

before_test_epoch(*runner*)
We load parameter values from ema model to source model before test.

Parameters **runner** (Runner) – The runner of the training process.

Return type None

before_train(*runner*)
Check the begin_epoch/iter is smaller than max_epochs/iters.

Parameters **runner** (Runner) – The runner of the training process.

Return type None

before_val_epoch(*runner*)
We load parameter values from ema model to source model before validation.

Parameters **runner** (Runner) – The runner of the training process.

Return type None

39.4 LoggerHook

class mmengine.hooks.**LoggerHook**(*interval=10*, *ignore_last=True*, *interval_exp_name=1000*, *out_dir=None*, *out_suffix=('.json', '.log', '.py', 'yaml')*, *keep_local=True*, *file_client_args=None*, *log_metric_by_epoch=True*, *backend_args=None*)

Collect logs from different components of `Runner` and write them to terminal, JSON file, tensorboard and wandb .etc.

`LoggerHook` is used to record logs formatted by `LogProcessor` during training/validation/testing phase. It is used to control following behaviors:

- The frequency of logs update in terminal, local, tensorboad wandb.etc.
- The frequency of show experiment information in terminal.
- The work directory to save logs.

Parameters

- **interval** (*int*) – Logging interval (every k iterations). Defaults to 10.
- **ignore_last** (*bool*) – Ignore the log of last iterations in each epoch if the number of remaining iterations is less than `interval`. Defaults to True.
- **interval_exp_name** (*int*) – Logging interval for experiment name. This feature is to help users conveniently get the experiment information from screen or log file. Defaults to 1000.
- **out_dir** (*str or Path, optional*) – The root directory to save checkpoints. If not specified, `runner.work_dir` will be used by default. If specified, the `out_dir` will be the concatenation of `out_dir` and the last level directory of `runner.work_dir`. For example, if the input `our_dir` is `./tmp` and `runner.work_dir` is `./work_dir/cur_exp`, then the log will be saved in `./tmp/cur_exp`. Defaults to None.
- **out_suffix** (*Tuple[str] or str*) – Those files in `runner._log_dir` ending with `out_suffix` will be copied to `out_dir`. Defaults to ('json', '.log', '.py').
- **keep_local** (*bool*) – Whether to keep local logs in the local machine when `out_dir` is specified. If False, the local log will be removed. Defaults to True.
- **file_client_args** (*dict, optional*) – Arguments to instantiate a FileClient. See *mmengine.fileio.FileClient* for details. Defaults to None. It will be deprecated in future. Please use *backend_args* instead.
- **log_metric_by_epoch** (*bool*) – Whether to output metric in validation step by epoch. It can be true when running in epoch based runner. If set to True, *after_val_epoch* will set *step* to self.epoch in *runner.visualizer.add_scalars*. Otherwise *step* will be self.iter. Default to True.
- **backend_args** (*dict, optional*) – Arguments to instantiate the preifx of uri corresponding backend. Defaults to None. New in v0.2.0.

Examples

```
>>> # The simplest LoggerHook config.
>>> logger_hook_cfg = dict(interval=20)
```

after_run(*runner*)
Copy logs to `self.out_dir` if `self.out_dir is not None`

Parameters **runner** (Runner) – The runner of the training/testing/validation process.

Return type None

after_test_epoch(*runner*, *metrics=None*)
All subclasses should override this method, if they need any operations after each test epoch.

Parameters

- **runner** (Runner) – The runner of the testing process.
- **metrics** (*Dict[str, float], optional*) – Evaluation results of all metrics on test dataset. The keys are the names of the metrics, and the values are corresponding results.

Return type None

after_test_iter(*runner*, *batch_idx*, *data_batch=None*, *outputs=None*)
Record logs after testing iteration.

Parameters

- **runner** (Runner) – The runner of the testing process.
- **batch_idx** (*int*) – The index of the current batch in the test loop.
- **data_batch** (*dict or tuple or list, optional*) – Data from dataloader.
- **outputs** (*sequence, optional*) – Outputs from model.

Return type None

after_train_iter(*runner*, *batch_idx*, *data_batch=None*, *outputs=None*)
Record logs after training iteration.

Parameters

- **runner** (Runner) – The runner of the training process.
- **batch_idx** (*int*) – The index of the current batch in the train loop.
- **data_batch** (*dict tuple or list, optional*) – Data from dataloader.
- **outputs** (*dict, optional*) – Outputs from model.

Return type None

after_val_epoch(*runner*, *metrics=None*)
All subclasses should override this method, if they need any operations after each validation epoch.

Parameters

- **runner** (Runner) – The runner of the validation process.
- **metrics** (*Dict[str, float], optional*) – Evaluation results of all metrics on validation dataset. The keys are the names of the metrics, and the values are corresponding results.

Return type None

after_val_iter(*runner*, *batch_idx*, *data_batch=None*, *outputs=None*)
Record logs after validation iteration.

Parameters

- **runner** (Runner) – The runner of the validation process.
- **batch_idx** (*int*) – The index of the current batch in the validation loop.
- **data_batch** (*dict or tuple or list, optional*) – Data from dataloader. Defaults to None.
- **outputs** (*sequence, optional*) – Outputs from model.

Return type None

before_run(*runner*)
Infer `self.file_client` from `self.out_dir`. Initialize the `self.start_iter` and record the meta information.

Parameters **runner** (Runner) – The runner of the training process.

Return type None

39.5 NaiveVisualizationHook

class mmengine.hooks.**NaiveVisualizationHook**(*interval=1*, *draw_gt=True*, *draw_pred=True*)
Show or Write the predicted results during the process of testing.

Parameters

- **interval** (*int*) – Visualization interval. Defaults to 1.
- **draw_gt** (*bool*) – Whether to draw the ground truth. Default to True.
- **draw_pred** (*bool*) – Whether to draw the predicted result. Default to True.

after_test_iter(*runner*, *batch_idx*, *data_batch=None*, *outputs=None*)
Show or Write the predicted results.

Parameters

- **runner** (Runner) – The runner of the training process.
- **batch_idx** (*int*) – The index of the current batch in the test loop.
- **data_batch** (*dict or tuple or list, optional*) – Data from dataloader.
- **outputs** (*Sequence, optional*) – Outputs from model.

Return type None

39.6 ParamSchedulerHook

class mmengine.hooks.**ParamSchedulerHook**

A hook to update some hyper-parameters in optimizer, e.g., learning rate and momentum.

after_train_epoch(*runner*)

Call step function for each scheduler after each epoch.

Parameters **runner** (Runner) – The runner of the training process.

Return type None

after_train_iter(*runner*, *batch_idx*, *data_batch=None*, *outputs=None*)

Call step function for each scheduler after each iteration.

Parameters

- **runner** (Runner) – The runner of the training process.
- **batch_idx** (*int*) – The index of the current batch in the train loop.
- **data_batch** (*dict or tuple or list, optional*) – Data from dataloader. In order to keep this interface consistent with other hooks, we keep `data_batch` here.
- **outputs** (*dict, optional*) – Outputs from model. In order to keep this interface consistent with other hooks, we keep `data_batch` here.

Return type None

39.7 RuntimeInfoHook

class mmengine.hooks.**RuntimeInfoHook**

A hook that updates runtime information into message hub.

E.g. `epoch`, `iter`, `max_epochs`, and `max_iters` for the training state. Components that cannot access the runner can get runtime information through the message hub.

after_test_epoch(*runner*, *metrics=None*)

All subclasses should override this method, if they need any operations after each test epoch.

Parameters

- **runner** (Runner) – The runner of the testing process.
- **metrics** (*Dict[str, float], optional*) – Evaluation results of all metrics on test dataset. The keys are the names of the metrics, and the values are corresponding results.

Return type None

after_train_iter(*runner*, *batch_idx*, *data_batch=None*, *outputs=None*)

Update `log_vars` in model outputs every iteration.

Parameters

- **runner** (Runner) – The runner of the training process.
- **batch_idx** (*int*) – The index of the current batch in the train loop.
- **data_batch** (*Sequence[dict], optional*) – Data from dataloader. Defaults to None.
- **outputs** (*dict, optional*) – Outputs from model. Defaults to None.

Return type None

after_val_epoch(*runner*, *metrics=None*)
All subclasses should override this method, if they need any operations after each validation epoch.

Parameters

- **runner** (Runner) – The runner of the validation process.
- **metrics** (*Dict[str, float], optional*) – Evaluation results of all metrics on validation dataset. The keys are the names of the metrics, and the values are corresponding results.

Return type None

before_run(*runner*)
Update metainfo.

Parameters **runner** (Runner) – The runner of the training process.

Return type None

before_train(*runner*)
Update resumed training state.

Parameters **runner** (Runner) – The runner of the training process.

Return type None

before_train_epoch(*runner*)
Update current epoch information before every epoch.

Parameters **runner** (Runner) – The runner of the training process.

Return type None

before_train_iter(*runner*, *batch_idx*, *data_batch=None*)
Update current iter and learning rate information before every iteration.

Parameters

- **runner** (Runner) – The runner of the training process.
- **batch_idx** (*int*) – The index of the current batch in the train loop.
- **data_batch** (*Sequence[dict], optional*) – Data from dataloader. Defaults to None.

Return type None

39.8 DistSamplerSeedHook

class mmengine.hooks.DistSamplerSeedHook
Data-loading sampler for distributed training.

When distributed training, it is only useful in conjunction with EpochBasedRunner, while IterBasedRunner achieves the same purpose with IterLoader.

before_train_epoch(*runner*)
Set the seed for sampler and batch_sampler.

Parameters **runner** (Runner) – The runner of the training process.

Return type None

39.9 IterTimerHook

class mmengine.hooks.**IterTimerHook**
A hook that logs the time spent during iteration.

E.g. `data_time` for loading data and `time` for a model train step.

before_train(*runner*)
Synchronize the number of iterations with the runner after resuming from checkpoints.

Parameters **runner** – The runner of the training, validation or testing process.

Return type None

39.10 SyncBuffersHook

class mmengine.hooks.**SyncBuffersHook**
Synchronize model buffers such as running_mean and running_var in BN at the end of each epoch.

Return type None

after_train_epoch(*runner*)
All-reduce model buffers at the end of each epoch.

Parameters **runner** (Runner) – The runner of the training process.

Return type None

39.11 EmptyCacheHook

class mmengine.hooks.**EmptyCacheHook**(*before_epoch=False*, *after_epoch=True*, *after_iter=False*)
Releases all unoccupied cached GPU memory during the process of training.

Parameters

- **before_epoch** (*bool*) – Whether to release cache before an epoch. Defaults to False.
- **after_epoch** (*bool*) – Whether to release cache after an epoch. Defaults to True.
- **after_iter** (*bool*) – Whether to release cache after an iteration. Defaults to False.

Return type None

CHAPTER

FORTY

MMENGINE.MODEL

mmengine.model

- *Module*
- *Model*
- *EMA*
- *Model Wrapper*
- *Weight Initialization*
- *Utils*

40.1 Module

BaseModule	Base module for all modules in openmmlab.
ModuleDict	ModuleDict in openmmlab.
ModuleList	ModuleList in openmmlab.
Sequential	Sequential module in openmmlab.

40.1.1 BaseModule

class mmengine.model.**BaseModule**(*init_cfg=None*)

Base module for all modules in openmmlab. `BaseModule` is a wrapper of `torch.nn.Module` with additional functionality of parameter initialization. Compared with `torch.nn.Module`, `BaseModule` mainly adds three attributes.

- `init_cfg`: the config to control the initialization.
- `init_weights`: The function of parameter initialization and recording initialization information.
- `_params_init_info`: Used to track the parameter initialization information. This attribute only exists during executing the `init_weights`.

Parameters **init_cfg** (*dict, optional*) – Initialization config dict.

init_weights()

Initialize the weights.

40.1.2 ModuleDict

class mmengine.model.**ModuleDict**(*modules=None*, *init_cfg=None*)
ModuleDict in openmmlab.

Ensures that all modules in `ModuleDict` have a different initialization strategy than the outer model

Parameters

- **modules** (*dict, optional*) – A mapping (dictionary) of (string: module) or an iterable of key-value pairs of type (string, module).
- **init_cfg** (*dict, optional*) – Initialization config dict.

40.1.3 ModuleList

class mmengine.model.**ModuleList**(*modules=None*, *init_cfg=None*)
ModuleList in openmmlab.

Ensures that all modules in `ModuleList` have a different initialization strategy than the outer model

Parameters

- **modules** (*iterable, optional*) – An iterable of modules to add.
- **init_cfg** (*dict, optional*) – Initialization config dict.

40.1.4 Sequential

class mmengine.model.**Sequential**(**args*, *init_cfg=None*)
Sequential module in openmmlab.

Ensures that all modules in `Sequential` have a different initialization strategy than the outer model

Parameters **init_cfg** (*dict, optional*) – Initialization config dict.

40.2 Model

BaseModel	Base class for all algorithmic models.
BaseDataPreprocessor	Base data pre-processor used for copying data to the target device.
ImgDataPreprocessor	Image pre-processor for normalization and bgr to rgb conversion.
BaseTTAModel	Base model for inference with test-time augmentation.

40.2.1 BaseModel

class mmengine.model.**BaseModel**(*data_preprocessor=None*, *init_cfg=None*)

Base class for all algorithmic models.

BaseModel implements the basic functions of the algorithmic model, such as weights initialize, batch inputs preprocess(see more information in *BaseDataPreprocessor*), parse losses, and update model parameters.

Subclasses inherit from BaseModel only need to implement the forward method, which implements the logic to calculate loss and predictions, then can be trained in the runner.

Examples

```
>>> @MODELS.register_module()
>>> class ToyModel(BaseModel):
>>>
>>>     def __init__(self):
>>>         super().__init__()
>>>         self.backbone = nn.Sequential()
>>>         self.backbone.add_module('conv1', nn.Conv2d(3, 6, 5))
>>>         self.backbone.add_module('pool', nn.MaxPool2d(2, 2))
>>>         self.backbone.add_module('conv2', nn.Conv2d(6, 16, 5))
>>>         self.backbone.add_module('fc1', nn.Linear(16 * 5 * 5, 120))
>>>         self.backbone.add_module('fc2', nn.Linear(120, 84))
>>>         self.backbone.add_module('fc3', nn.Linear(84, 10))
>>>
>>>         self.criterion = nn.CrossEntropyLoss()
>>>
>>>     def forward(self, batch_inputs, data_samples, mode='tensor'):
>>>         data_samples = torch.stack(data_samples)
>>>         if mode == 'tensor':
>>>             return self.backbone(batch_inputs)
>>>         elif mode == 'predict':
>>>             feats = self.backbone(batch_inputs)
>>>             predictions = torch.argmax(feats, 1)
>>>             return predictions
>>>         elif mode == 'loss':
>>>             feats = self.backbone(batch_inputs)
>>>             loss = self.criterion(feats, data_samples)
>>>             return dict(loss=loss)
```

Parameters

- **data_preprocessor** (*dict, optional*) – The pre-process config of *BaseDataPreprocessor*.
- **init_cfg** (*dict, optional*) – The weight initialized config for *BaseModule*.

data_preprocessor

Used for pre-processing data sampled by dataloader to the format accepted by *forward()*.

Type *BaseDataPreprocessor*

init_cfg

Initialization config dict.

Type dict, optional

cpu(**args*, ***kwargs*)
Overrides this method to call *BaseDataPreprocessor.cpu()* additionally.

Returns The model itself.

Return type nn.Module

cuda(*device=None*)
Overrides this method to call *BaseDataPreprocessor.cuda()* additionally.

Returns The model itself.

Return type nn.Module

Parameters **device** (*Optional[Union[int, str, torch.device]]*) –

abstract forward(*inputs*, *data_samples=None*, *mode='tensor'*)
Returns losses or predictions of training, validation, testing, and simple inference process.

`forward` method of BaseModel is an abstract method, its subclasses must implement this method.

Accepts `batch_inputs` and `data_sample` processed by *data_preprocessor*, and returns results according to mode arguments.

During non-distributed training, validation, and testing process, `forward` will be called by `BaseModel.train_step`, `BaseModel.val_step` and `BaseModel.val_step` directly.

During distributed data parallel training process, `MMSeparateDistributedDataParallel.train_step` will first call `DistributedDataParallel.forward` to enable automatic gradient synchronization, and then call `forward` to get training loss.

Parameters

- **inputs** (*torch.Tensor*) – batch input tensor collated by *data_preprocessor*.
- **data_samples** (*list, optional*) – data samples collated by *data_preprocessor*.
- **mode** (*str*) – mode should be one of `loss`, `predict` and `tensor`
  - `loss`: Called by `train_step` and return loss `dict` used for logging
  - `predict`: Called by `val_step` and `test_step` and return list of `results used for computing metric.
  - `tensor`: Called by custom use to get `Tensor` type results.

Returns

- If `mode == loss`, return a `dict` of loss tensor used for backward and logging.
- If `mode == predict`, return a `list` of inference results.
- If `mode == tensor`, return a tensor or `tuple` of tensor or ``dict of tensor for custom use.

Return type dict or list

npu(*device=None*)
Overrides this method to call `BaseDataPreprocessor.npu()` additionally.

Returns The model itself.

Return type nn.Module

Parameters **device** (*Optional[Union[int, str, torch.device]]*) –

Note: This generation of NPU(Ascend910) does not support the use of multiple cards in a single process, so the index here needs to be consistent with the default device

parse_losses(*losses*)
Parses the raw outputs (losses) of the network.

Parameters **losses** (*dict*) – Raw output of the network, which usually contain losses and other necessary information.

Returns There are two elements. The first is the loss tensor passed to optim_wrapper which may be a weighted sum of all losses, and the second is log_vars which will be sent to the logger.

Return type tuple[Tensor, dict]

test_step(*data*)
`BaseModel` implements `test_step` the same as `val_step`.

Parameters **data** (*dict or tuple or list*) – Data sampled from dataset.

Returns The predictions of given data.

Return type list

to(*device=None*, **args*, ***kwargs*)
Overrides this method to call *BaseDataPreprocessor.to()* additionally.

Parameters **device** (*int, str or torch.device, optional*) – the desired device of the parameters and buffers in this module.

Returns The model itself.

Return type nn.Module

train_step(*data*, *optim_wrapper*)
Implements the default model training process including preprocessing, model forward propagation, loss calculation, optimization, and back-propagation.

During non-distributed training. If subclasses do not override the *train_step()*, `EpochBasedTrainLoop` or `IterBasedTrainLoop` will call this method to update model parameters. The default parameter update process is as follows:

1. Calls `self.data_processor(data, training=False)` to collect batch_inputs and corresponding data_samples(labels).
2. Calls `self(batch_inputs, data_samples, mode='loss')` to get raw loss
3. Calls `self.parse_losses` to get `parsed_losses` tensor used to backward and dict of loss tensor used to log messages.
4. Calls `optim_wrapper.update_params(loss)` to update model.

Parameters

- **data** (*dict or tuple or list*) – Data sampled from dataset.
- **optim_wrapper** (`OptimWrapper`) – OptimWrapper instance used to update model parameters.

Returns A `dict` of tensor for logging.

Return type Dict[str, torch.Tensor]

val_step(*data*)
Gets the predictions of given data.

Calls `self.data_preprocessor(data, False)` and `self(inputs, data_sample, mode='predict')` in order. Return the predictions which will be passed to evaluator.

Parameters **data** (*dict or tuple or list*) – Data sampled from dataset.

Returns The predictions of given data.

Return type list

40.2.2 BaseDataPreprocessor

class mmengine.model.**BaseDataPreprocessor**(*non_blocking=False*)
Base data pre-processor used for copying data to the target device.

Subclasses inherit from `BaseDataPreprocessor` could override the forward method to implement custom data pre-processing, such as batch-resize, MixUp, or CutMix.

Parameters **non_blocking** (*bool*) – Whether block current process when transferring data to device. New in version 0.3.0.

Note: Data dictionary returned by dataloader must be a dict and at least contain the `inputs` key.

---

cast_data(*data*)
Copying data to the target device.

Parameters **data** (*dict*) – Data returned by `DataLoader`.

Returns Inputs and data sample at target device.

Return type CollatedResult

cpu(**args*, ***kwargs*)
Overrides this method to set the `device`

Returns The model itself.

Return type nn.Module

cuda(**args*, ***kwargs*)
Overrides this method to set the `device`

Returns The model itself.

Return type nn.Module

forward(*data*, *training=False*)
Preprocesses the data into the model input format.

After the data pre-processing of *cast_data()*, `forward` will stack the input tensor list to a batch tensor at the first dimension.

Parameters

- **data** (*dict*) – Data returned by dataloader
- **training** (*bool*) – Whether to enable training time augmentation.

Returns Data in the same format as the model input.

Return type dict or list

to(*device*, **args*, ***kwargs*)
Overrides this method to set the `device`

Parameters **device** (*int or torch.device, optional*) – The desired device of the parameters and buffers in this module.

Returns The model itself.

Return type nn.Module

40.2.3 ImgDataPreprocessor

class mmengine.model.**ImgDataPreprocessor**(*mean=None*, *std=None*, *pad_size_divisor=1*, *pad_value=0*, *bgr_to_rgb=False*, *rgb_to_bgr=False*, *non_blocking=False*)
Image pre-processor for normalization and bgr to rgb conversion.

Accepts the data sampled by the dataloader, and preprocesses it into the format of the model input. `ImgDataPreprocessor` provides the basic data pre-processing as follows

- Collates and moves data to the target device.
- Converts inputs from bgr to rgb if the shape of input is (3, H, W).
- Normalizes image with defined std and mean.
- Pads inputs to the maximum size of current batch with defined `pad_value`. The padding size can be divisible by a defined `pad_size_divisor`
- Stack inputs to batch_inputs.

For `ImgDataPreprocessor`, the dimension of the single inputs must be (3, H, W).

Note: `ImgDataPreprocessor` and its subclass is built in the constructor of `BaseDataset`.

Parameters

- **mean** (*Sequence[float or int], optional*) – The pixel mean of image channels. If `bgr_to_rgb=True` it means the mean value of R, G, B channels. If the length of *mean* is 1, it means all channels have the same mean value, or the input is a gray image. If it is not specified, images will not be normalized. Defaults None.
- **std** (*Sequence[float or int], optional*) – The pixel standard deviation of image channels. If `bgr_to_rgb=True` it means the standard deviation of R, G, B channels. If the length of *std* is 1, it means all channels have the same standard deviation, or the input is a gray image. If it is not specified, images will not be normalized. Defaults None.
- **pad_size_divisor** (*int*) – The size of padded image should be divisible by `pad_size_divisor`. Defaults to 1.
- **pad_value** (*float or int*) – The padded pixel value. Defaults to 0.
- **bgr_to_rgb** (*bool*) – whether to convert image from BGR to RGB. Defaults to False.
- **rgb_to_bgr** (*bool*) – whether to convert image from RGB to RGB. Defaults to False.
- **non_blocking** (*bool*) – Whether block current process when transferring data to device. New in version v0.3.0.

Note: if images do not need to be normalized, *std* and *mean* should be both set to None, otherwise both of them should be set to a tuple of corresponding values.

forward(*data*, *training=False*)

Performs normalizationpadding and bgr2rgb conversion based on `BaseDataPreprocessor`.

Parameters

- **data** (*dict*) – Data sampled from dataset. If the collate function of DataLoader is `pseudo_collate`, data will be a list of dict. If collate function is `default_collate`, data will be a tuple with batch input tensor and list of data samples.
- **training** (*bool*) – Whether to enable training time augmentation. If subclasses override this method, they can perform different preprocessing strategies for training and testing based on the value of `training`.

Returns Data in the same format as the model input.

Return type dict or list

40.2.4 BaseTTAModel

class mmengine.model.**BaseTTAModel**(*module*)

Base model for inference with test-time augmentation.

`BaseTTAModel` is a wrapper for inference given multi-batch data. It implements the *test_step()* for multi-batch data inference. `multi-batch` data means data processed by different augmentation from the same batch.

During test time augmentation, the data processed by `mmcv.transforms.TestTimeAug`, and then collated by `pseudo_collate` will have the following format:

```
result = dict(
    inputs=[
        [image1_aug1, image2_aug1],
        [image1_aug2, image2_aug2]
    ],
    data_samples=[
        [data_sample1_aug1, data_sample2_aug1],
        [data_sample1_aug2, data_sample2_aug2],
    ]
)
```

`image{i}_aug{j}` means the i-th image of the batch, which is augmented by the j-th augmentation.

`BaseTTAModel` will collate the data to:

```
data1 = dict(
    inputs=[image1_aug1, image2_aug1],
    data_samples=[data_sample1_aug1, data_sample2_aug1]
)

data2 = dict(
    inputs=[image1_aug2, image2_aug2],
    data_samples=[data_sample1_aug2, data_sample2_aug2]
)
```

data1 and data2 will be passed to model, and the results will be merged by *merge_preds()*.

Note: *merge_preds()* is an abstract method, all subclasses should implement it.

Parameters **module** (*dict or nn.Module*) – Tested model.

abstract merge_preds(*data_samples_list*)
Merge predictions of enhanced data to one prediction.

Parameters **data_samples_list** (*EnhancedBatchDataSamples*) – List of predictions of all enhanced data.

Returns Merged prediction.

Return type List[*BaseDataElement*]

test_step(*data*)
Get predictions of each enhanced data, a multiple predictions.

Parameters **data** (*DataBatch*) – Enhanced data batch sampled from dataloader.

Returns Merged prediction.

Return type MergedDataSamples

40.3 EMA

BaseAveragedModel	A base class for averaging model weights.
ExponentialMovingAverage	Implements the exponential moving average (EMA) of the model.
MomentumAnnealingEMA	Exponential moving average (EMA) with momentum annealing strategy.
StochasticWeightAverage	Implements the stochastic weight averaging (SWA) of the model.

40.3.1 BaseAveragedModel

class mmengine.model.**BaseAveragedModel**(*model*, *interval=1*, *device=None*, *update_buffers=False*)
A base class for averaging model weights.

Weight averaging, such as SWA and EMA, is a widely used technique for training neural networks. This class implements the averaging process for a model. All subclasses must implement the *avg_func* method. This class creates a copy of the provided module `model` on the `device` and allows computing running averages of the parameters of the `model`.

The code is referenced from: https://github.com/pytorch/pytorch/blob/master/torch/optim/swa_utils.py.

Different from the *AveragedModel* in PyTorch, we use in-place operation to improve the parameter updating speed, which is about 5 times faster than the non-in-place version.

In mmengine, we provide two ways to use the model averaging:

1. Use the model averaging module in hook: We provide an *mmengine.hooks.EMAHook* to apply the model averaging during training. Add `custom_hooks=[dict(type='EMAHook')]` to the config or the runner.

2. Use the model averaging module directly in the algorithm. Take the ema teacher in semi-supervise as an example:

```
>>> from mmengine.model import ExponentialMovingAverage
>>> student = ResNet(depth=50)
>>> # use ema model as teacher
>>> ema_teacher = ExponentialMovingAverage(student)
```

Parameters

- **model** (*nn.Module*) – The model to be averaged.
- **interval** (*int*) – Interval between two updates. Defaults to 1.
- **device** (*torch.device, optional*) – If provided, the averaged model will be stored on the `device`. Defaults to None.
- **update_buffers** (*bool*) – if True, it will compute running averages for both the parameters and the buffers of the model. Defaults to False.

Return type None

abstract avg_func(*averaged_param*, *source_param*, *steps*)

Use in-place operation to compute the average of the parameters. All subclasses must implement this method.

Parameters

- **averaged_param** (*Tensor*) – The averaged parameters.
- **source_param** (*Tensor*) – The source parameters.
- **steps** (*int*) – The number of times the parameters have been updated.

Return type None

forward(**args*, ***kwargs*)

Forward method of the averaged model.

update_parameters(*model*)

Update the parameters of the model. This method will execute the `avg_func` to compute the new parameters and update the model's parameters.

Parameters **model** (*nn.Module*) – The model whose parameters will be averaged.

Return type None

40.3.2 ExponentialMovingAverage

class mmengine.model.**ExponentialMovingAverage**(*model*, *momentum=0.0002*, *interval=1*, *device=None*, *update_buffers=False*)

Implements the exponential moving average (EMA) of the model.

All parameters are updated by the formula as below:

$$Xema_{t+1} = (1 - momentum) * Xema_t + momentum * X_t$$

Parameters

- **model** (*nn.Module*) – The model to be averaged.
- **momentum** (*float*) – The momentum used for updating ema parameter. Defaults to 0.0002. Ema's parameter are updated with the formula $averaged\_param = (1 - momentum) * averaged\_param + momentum * source\_param$.
- **interval** (*int*) – Interval between two updates. Defaults to 1.
- **device** (*torch.device, optional*) – If provided, the averaged model will be stored on the `device`. Defaults to None.
- **update_buffers** (*bool*) – if True, it will compute running averages for both the parameters and the buffers of the model. Defaults to False.

Return type None

avg_func(*averaged_param*, *source_param*, *steps*)
Compute the moving average of the parameters using exponential moving average.

Parameters

- **averaged_param** (*Tensor*) – The averaged parameters.
- **source_param** (*Tensor*) – The source parameters.
- **steps** (*int*) – The number of times the parameters have been updated.

Return type None

40.3.3 MomentumAnnealingEMA

class mmengine.model.**MomentumAnnealingEMA**(*model*, *momentum=0.0002*, *gamma=100*, *interval=1*, *device=None*, *update_buffers=False*)
Exponential moving average (EMA) with momentum annealing strategy.

Parameters

- **model** (*nn.Module*) – The model to be averaged.
- **momentum** (*float*) – The momentum used for updating ema parameter. Defaults to 0.0002. Ema's parameter are updated with the formula $averaged\_param = (1 - momentum) * averaged\_param + momentum * source\_param$.
- **gamma** (*int*) – Use a larger momentum early in training and gradually annealing to a smaller value to update the ema model smoothly. The momentum is calculated as max(momentum, gamma / (gamma + steps)) Defaults to 100.
- **interval** (*int*) – Interval between two updates. Defaults to 1.
- **device** (*torch.device, optional*) – If provided, the averaged model will be stored on the `device`. Defaults to None.
- **update_buffers** (*bool*) – if True, it will compute running averages for both the parameters and the buffers of the model. Defaults to False.

Return type None

avg_func(*averaged_param*, *source_param*, *steps*)
Compute the moving average of the parameters using the linear momentum strategy.

Parameters

- **averaged_param** (*Tensor*) – The averaged parameters.

- **source_param** (*Tensor*) – The source parameters.
- **steps** (*int*) – The number of times the parameters have been updated.

Return type None

40.3.4 StochasticWeightAverage

class mmengine.model.**StochasticWeightAverage**(*model*, *interval=1*, *device=None*, *update_buffers=False*)

Implements the stochastic weight averaging (SWA) of the model.

Stochastic Weight Averaging was proposed in Averaging Weights Leads to Wider Optima and Better Generalization, UAI 2018. by Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov and Andrew Gordon Wilson.

Parameters

- **model** (*torch.nn.modules.module.Module*) –
- **interval** (*int*) –
- **device** (*Optional[torch.device]*) –
- **update_buffers** (*bool*) –

Return type None

avg_func(*averaged_param*, *source_param*, *steps*)

Compute the average of the parameters using stochastic weight average.

Parameters

- **averaged_param** (*Tensor*) – The averaged parameters.
- **source_param** (*Tensor*) – The source parameters.
- **steps** (*int*) – The number of times the parameters have been updated.

Return type None

40.4 Model Wrapper

MMDistributedDataParallel	A distributed model wrapper used for training,testing and validation in loop.
MMSeparateDistributedDataParallel	A DistributedDataParallel wrapper for models in MMGeneration.
MMFullyShardedDataParallel	A wrapper for sharding Module parameters across data parallel workers.

40.4.1 MMDistributedDataParallel

class mmengine.model.**MMDistributedDataParallel**(*module*, *detect_anomalous_params=False*, ***kwargs*)

A distributed model wrapper used for training,testing and validation in loop.

Different from DistributedDataParallel, MMDistributedDataParallel implements three methods *train_step()*, *val_step()* and *test_step()*, which will be called by `train_loop`, `val_loop` and `test_loop`.

- `train_step`: Called by `runner.train_loop`, and implement default model forward, gradient back propagation, parameter updating logic. To take advantage of DistributedDataParallel's automatic gradient synchronization, `train_step` calls `DistributedDataParallel.forward` to calculate the losses, and call other methods of *BaseModel* to pre-process data and parse losses. Finally, update model parameters by `OptimWrapper` and return the loss dictionary used for logging.
- `val_step`: Called by `runner.val_loop` and get the inference results. Since there is no gradient synchronization requirement, this procedure is equivalent to `BaseModel.val_step`
- `test_step`: Called by `runner.test_loop`, equivalent `val_step`.

Parameters

- **detect_anomalous_params** (*bool*) – This option is only used for debugging which will slow down the training speed. Detect anomalous parameters that are not included in the computational graph with *loss* as the root. There are two cases
  - Parameters were not used during forward pass.
  - Parameters were not used to produce loss.

  Defaults to False.
- ****kwargs** – keyword arguments passed to `DistributedDataParallel`.
  - device_ids (List[int] or torch.device, optional): CUDA devices for module.
  - output_device (int or torch.device, optional): Device location of output for single-device CUDA modules.
  - dim (int): Defaults to 0.
  - broadcast_buffers (bool): Flag that enables syncing (broadcasting) buffers of the module at beginning of the `forward` function. Defaults to True
  - find_unused_parameters (bool): Whether to find parameters of module, which are not in the forward graph. Defaults to False.
  - process_group (ProcessGroup, optional): The process group to be used for distributed data all-reduction.
  - bucket_cap_mb (int): bucket size in MegaBytes (MB). Defaults to 25.
  - check_reduction (bool): This argument is deprecated. Defaults to False.
  - gradient_as_bucket_view (bool): Defaults to False.
  - static_graph (bool): Defaults to False.

See more information about arguments in `torch.nn.parallel.DistributedDataParallel`.

Note: If model has multiple submodules and each module has separate optimization strategies, *MMSeparateDistributedDataParallel* should be used to wrap the model.

Note: If model itself has custom optimization strategy, rather than simply forward model and update model. A custom model wrapper inherit from `MMDistributedDataParallel` should be defined and override the `train_step` method.

`test_step`(*data*)
Gets the predictions of module during testing process.

Parameters **data** (*dict or tuple or list*) – Data sampled from dataset.

Returns The predictions of given data.

Return type list

`train_step`(*data*, *optim_wrapper*)
Interface for model forward, backward and parameters updating during training process.

train_step() will perform the following steps in order:

- If `module` defines the preprocess method, call `module.preprocess` to pre-processing data.
- Call `module.forward(**data)` and get losses.
- Parse losses.
- Call `optim_wrapper.optimizer_step` to update parameters.
- Return log messages of losses.

Parameters

- **data** (*dict or tuple or list*) – Data sampled from dataset.
- **optim_wrapper** (`OptimWrapper`) – A wrapper of optimizer to update parameters.

Returns A `dict` of tensor for logging.

Return type Dict[str, torch.Tensor]

`val_step`(*data*)
Gets the prediction of module during validation process.

Parameters **data** (*dict or tuple or list*) – Data sampled from dataset.

Returns The predictions of given data.

Return type list

40.4.2 MMSeparateDistributedDataParallel

`class mmengine.model.MMSeparateDistributedDataParallel`(*module*, *broadcast_buffers=False*, *find_unused_parameters=False*, ***kwargs*)
A DistributedDataParallel wrapper for models in MMGeneration.

In MMedting and MMGeneration there is a need to wrap different modules in the models with separate DistributedDataParallel. Otherwise, it will cause errors for GAN training. For example, the GAN model, usually has two submodules: generator and discriminator. If we wrap both of them in one standard DistributedDataParallel, it will cause errors during training, because when we update the parameters of the generator (or discriminator), the parameters of the discriminator (or generator) is not updated, which is not allowed for DistributedDataParallel. So we design this wrapper to separately wrap DistributedDataParallel for generator and discriminator. In this wrapper, we perform two operations:

1. Wraps each module in the models with separate MMDistributedDataParallel. Note that only modules with parameters will be wrapped.
2. Calls `train_step`, `val_step` and `test_step` of submodules to get losses and predictions.

Parameters

- **module** (*nn.Module*) – model contain multiple submodules which have separately updating strategy.
- **broadcast_buffers** (*bool*) – Same as that in `torch.nn.parallel.distributed.DistributedDataParallel`. Defaults to False.
- **find_unused_parameters** (*bool*) – Same as that in `torch.nn.parallel.distributed.DistributedDataParallel`. Traverse the autograd graph of all tensors contained in returned value of the wrapped module's forward function. Defaults to False.
- ****kwargs** – Keyword arguments passed to `MMDistributedDataParallel`.
  - device_ids (List[int] or torch.device, optional): CUDA devices for module.
  - output_device (int or torch.device, optional): Device location of output for single-device CUDA modules.
  - dim (int): Defaults to 0.
  - process_group (ProcessGroup, optional): The process group to be used for distributed data all-reduction.
  - bucket_cap_mb (int): bucket size in MegaBytes (MB). Defaults to 25.
  - check_reduction (bool): This argument is deprecated. Defaults to False.
  - gradient_as_bucket_view (bool): Defaults to False.
  - static_graph (bool): Defaults to False.

See more information about arguments in `torch.nn.parallel.DistributedDataParallel`.

no_sync()
Enables `no_sync` context of all sub `MMDistributedDataParallel` modules.

test_step(*data*)
Gets the predictions of module during testing process.

Parameters **data** (*dict or tuple or list*) – Data sampled from dataset.

Returns The predictions of given data.

Return type list

train(*mode=True*)
Sets the module in training mode.

In order to make the ddp wrapper inheritance hierarchy more uniform, `MMSeparateDistributedDataParallel` inherits from `DistributedDataParallel`, but will not call its constructor. Since the attributes of `DistributedDataParallel` have not been initialized, call the `train` method of `DistributedDataParallel` will raise an error if pytorch version <= 1.9. Therefore, override this method to call the `train` method of submodules.

Parameters **mode** (*bool*) – whether to set training mode (`True`) or evaluation mode (`False`). Defaults to `True`.

Returns self.

Return type Module

train_step(*data*, *optim_wrapper*)
Interface for model forward, backward and parameters updating during training process.

Parameters

- **data** (*dict or tuple or list*) – Data sampled from dataset.
- **optim_wrapper** (OptimWrapperDict) – A wrapper of optimizer to update parameters.

Returns A dict of tensor for logging.

Return type Dict[str, torch.Tensor]

val_step(*data*)
Gets the prediction of module during validation process.

Parameters **data** (*dict or tuple or list*) – Data sampled from dataset.

Returns The predictions of given data.

Return type list

40.4.3 MMFullyShardedDataParallel

class mmengine.model.**MMFullyShardedDataParallel**(*module*, *process_group=None*, *cpu_offload=None*, *fsdp_auto_wrap_policy=None*, *backward_prefetch=None*, ***kwargs*)
A wrapper for sharding Module parameters across data parallel workers.

Different from FullyShardedDataParallel, MMFullyShardedDataParallel implements three methods *train_step()*, *val_step()* and *test_step()*, which will be called by `train_loop`, `val_loop` and `test_loop`.

- `train_step`: Called by `runner.train_loop`, and implement default model forward, gradient back propagation, parameter updating logic.
- `val_step`: Called by `runner.val_loop` and get the inference results. Specially, since MMFullyShardedDataParallel will wrap model recursively, it may cause some problem if one just use `BaseModel.val_step` to implement `val_step` here. To avoid that, `val_step` will call methods of *BaseModel* to pre-process data first, and use `FullyShardedDataParallel.forward` to get result.
- `test_step`: Called by `runner.test_loop` and get the inference results. Its logic is equivalent to `val_loop`.

Parameters

- **module** (*nn.Module*) – module to be wrapped with FSDP.
- **process_group** (*Optional[ProcessGroup]*) – process group for sharding.
- **cpu_offload** (*Optional[Union[bool,CPUOffload]]*) – CPU offloading config. Different from FullyShardedDataParallel,Since it can be set by users' pre-defined config in MMEngine,its type is expected to be *None*, *bool* or *CPUOffload*.

  Currently, only parameter and gradient CPU offload is supported. It can be enabled via passing in `cpu_offload=CPUOffload(offload_params=True)`. Note that this currently implicitly enables gradient offloading to CPU in order for params and grads to be on same device to work with optimizer. This API is subject to change. Default is `None` in which case there will be no offloading.

- **fsdp_auto_wrap_policy** (*Optional[Union[str, Callable]]*) – (Optional[Union[str,Callable]]): Specifying a policy to recursively wrap layers with FSDP. Different from FullyShardedDataParallel, Since it can be set by users' pre-defined config in MMEngine, its type is expected to be *None*, *str* or *Callable*. If it's *str*, then MMFullyShardedDataParallel will try to get specified method in `FSDP_WRAP_POLICIES` registry,and this method will be passed to FullyShardedDataParallel to finally initialize model.

  Note that this policy currently will only apply to child modules of the passed in module. The remainder modules are always wrapped in the returned FSDP root instance. `default_auto_wrap_policy` written in `torch.distributed.fsdp.wrap` is an example of `fsdp_auto_wrap_policy` callable, this policy wraps layers with parameter sizes larger than 100M. Users can supply the customized `fsdp_auto_wrap_policy` callable that should accept following arguments: `module: nn.Module`, `recurse: bool`, `unwrapped_params: int`, extra customized arguments could be added to the customized `fsdp_auto_wrap_policy` callable as well.

  Example:

```
>>> def custom_auto_wrap_policy(
>>>     module: nn.Module,
>>>     recurse: bool,
>>>     unwrapped_params: int,
>>>     # These are customizable for this policy function.
>>>     min_num_params: int = int(1e8),
>>> ) -> bool:
>>>     return unwrapped_params >= min_num_params
```

- **backward_prefetch** (*Optional[Union[str, torch.distributed.fsdp.fully_sharded_data_parallel.BackwardPrefetch]]*) – (Optional[Union[str,BackwardPrefetch]]): Different from FullyShardedDataParallel, Since it will be set by users' pre-defined config in MMEngine,its type is expected to be *None*, *str* or *BackwardPrefetch*.

  This is an experimental feature that is subject to change in the the near future. It allows users to enable two different backward_prefetch algorithms to help backward communication and computation overlapping. Pros and cons of each algorithm is explained in class `BackwardPrefetch`.

- ****kwargs** – Keyword arguments passed to `FullyShardedDataParallel`.

test_step(*data*)

Gets the predictions of module during testing process.

Parameters **data** (*dict*) – Data sampled by dataloader.

Returns The predictions of given data.

Return type List[*BaseDataElement*]

train_step(*data*, *optim_wrapper*)

Interface for model forward, backward and parameters updating during training process.

train_step() will perform the following steps in order:

- **If `module` defines the preprocess method,** call `module.preprocess` to pre-processing data.
- Call `module.forward(**data)` and get losses.
- Parse losses.

- Call `optim_wrapper.optimizer_step` to update parameters.
- Return log messages of losses.

Parameters

- **data** (*dict*) – Data sampled by dataloader.
- **optim_wrapper** (OptimWrapper) – A wrapper of optimizer to update parameters.

Returns A `dict` of tensor for logging.

Return type Dict[str, torch.Tensor]

val_step(*data*)

Gets the prediction of module during validation process.

Parameters **data** (*dict*) – Data sampled by dataloader.

Returns The predictions of given data.

Return type List[*BaseDataElement*] or dict

is_model_wrapper	Check if a module is a model wrapper.

40.4.4 is_model_wrapper

class mmengine.model.**is_model_wrapper**(*model*, *registry=Registry(name=model_wrapper, items={'DistributedDataParallel': <class 'torch.nn.parallel.distributed.DistributedDataParallel'>, 'DataParallel': <class 'torch.nn.parallel.data_parallel.DataParallel'>, 'MMDistributedDataParallel': <class 'mmengine.model.wrappers.distributed.MMDistributedDataParallel'>, 'MMSeparateDistributedDataParallel': <class 'mmengine.model.wrappers.seperate_distributed.MMSeparateDistributedDataParalle 'MMFullyShardedDataParallel': <class 'mmengine.model.wrappers.fully_sharded_distributed.MMFullyShardedDataParallel'*

Check if a module is a model wrapper.

The following 4 model in MMEngine (and their subclasses) are regarded as model wrappers: DataParallel, DistributedDataParallel, MMDataParallel, MMDistributedDataParallel. You may add you own model wrapper by registering it to `mmengine.registry.MODEL_WRAPPERS`.

Parameters

- **model** (*nn.Module*) – The model to be checked.
- **registry** (Registry) – The parent registry to search for model wrappers.

Returns True if the input model is a model wrapper.

Return type bool

40.5 Weight Initialization

BaseInit	
Caffe2XavierInit	
ConstantInit	Initialize module parameters with constant values.
KaimingInit	Initialize module parameters with the values according to the method described in `Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification - He, K.
NormalInit	Initialize module parameters with the values drawn from the normal distribution $\mathcal{N}(\text{mean}, \text{std}^2)$.
PretrainedInit	Initialize module by loading a pretrained model.
TruncNormalInit	Initialize module parameters with the values drawn from the normal distribution $\mathcal{N}(\text{mean}, \text{std}^2)$ with values outside $[a, b]$.
UniformInit	Initialize module parameters with values drawn from the uniform distribution $\mathcal{U}(a, b)$.
XavierInit	Initialize module parameters with values according to the method described in `Understanding the difficulty of training deep feedforward neural networks - Glorot, X.

40.5.1 BaseInit

class mmengine.model.**BaseInit**(*, *bias=0*, *bias_prob=None*, *layer=None*)

40.5.2 Caffe2XavierInit

class mmengine.model.**Caffe2XavierInit**(***kwargs*)

40.5.3 ConstantInit

class mmengine.model.**ConstantInit**(*val*, ***kwargs*)

Initialize module parameters with constant values.

Parameters

- **val** (*int | float*) – the value to fill the weights in the module with
- **bias** (*int | float*) – the value to fill the bias. Defaults to 0.
- **bias_prob** (*float, optional*) – the probability for bias initialization. Defaults to None.
- **layer** (*str | list[str], optional*) – the layer will be initialized. Defaults to None.

40.5.4 KaimingInit

class mmengine.model.**KaimingInit**(*a=0*, *mode='fan_out'*, *nonlinearity='relu'*, *distribution='normal'*, ***kwargs*)

Initialize module parameters with the values according to the method described in Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification - He, K. et al. (2015).

Parameters

- **a** (*int | float*) – the negative slope of the rectifier used after this layer (only used with `'leaky_relu'`). Defaults to 0.
- **mode** (*str*) – either `'fan_in'` or `'fan_out'`. Choosing `'fan_in'` preserves the magnitude of the variance of the weights in the forward pass. Choosing `'fan_out'` preserves the magnitudes in the backwards pass. Defaults to `'fan_out'`.
- **nonlinearity** (*str*) – the non-linear function (*nn.functional* name), recommended to use only with `'relu'` or `'leaky_relu'` . Defaults to 'relu'.
- **bias** (*int | float*) – the value to fill the bias. Defaults to 0.
- **bias_prob** (*float, optional*) – the probability for bias initialization. Defaults to None.
- **distribution** (*str*) – distribution either be `'normal'` or `'uniform'`. Defaults to `'normal'`.
- **layer** (*str | list[str], optional*) – the layer will be initialized. Defaults to None.

40.5.5 NormalInit

class mmengine.model.**NormalInit**(*mean=0*, *std=1*, ***kwargs*)

Initialize module parameters with the values drawn from the normal distribution $\mathcal{N}(\text{mean}, \text{std}^2)$.

Parameters

- **mean** (*int | float*) – the mean of the normal distribution. Defaults to 0.
- **std** (*int | float*) – the standard deviation of the normal distribution. Defaults to 1.
- **bias** (*int | float*) – the value to fill the bias. Defaults to 0.
- **bias_prob** (*float, optional*) – the probability for bias initialization. Defaults to None.
- **layer** (*str | list[str], optional*) – the layer will be initialized. Defaults to None.

40.5.6 PretrainedInit

class mmengine.model.**PretrainedInit**(*checkpoint*, *prefix=None*, *map_location='cpu'*)

Initialize module by loading a pretrained model.

Parameters

- **checkpoint** (*str*) – the checkpoint file of the pretrained model should be load.
- **prefix** (*str, optional*) – the prefix of a sub-module in the pretrained model. it is for loading a part of the pretrained model to initialize. For example, if we would like to only load the backbone of a detector model, we can set `prefix='backbone.'`. Defaults to None.
- **map_location** (*str*) – map tensors into proper locations. Defaults to cpu.

40.5.7 TruncNormalInit

class mmengine.model.**TruncNormalInit**(*mean=0*, *std=1*, *a=- 2*, *b=2*, ***kwargs*)

Initialize module parameters with the values drawn from the normal distribution $\mathcal{N}(\text{mean}, \text{std}^2)$ with values outside $[a, b]$.

Parameters

- **mean** (*float*) – the mean of the normal distribution. Defaults to 0.
- **std** (*float*) – the standard deviation of the normal distribution. Defaults to 1.
- **a** (*float*) – The minimum cutoff value.
- **b** (*float*) – The maximum cutoff value.
- **bias** (*float*) – the value to fill the bias. Defaults to 0.
- **bias_prob** (*float, optional*) – the probability for bias initialization. Defaults to None.
- **layer** (*str | list[str], optional*) – the layer will be initialized. Defaults to None.

Return type None

40.5.8 UniformInit

class mmengine.model.**UniformInit**(*a=0*, *b=1*, ***kwargs*)

Initialize module parameters with values drawn from the uniform distribution $\mathcal{U}(a, b)$.

Parameters

- **a** (*int | float*) – the lower bound of the uniform distribution. Defaults to 0.
- **b** (*int | float*) – the upper bound of the uniform distribution. Defaults to 1.
- **bias** (*int | float*) – the value to fill the bias. Defaults to 0.
- **bias_prob** (*float, optional*) – the probability for bias initialization. Defaults to None.
- **layer** (*str | list[str], optional*) – the layer will be initialized. Defaults to None.

40.5.9 XavierInit

class mmengine.model.**XavierInit**(*gain=1*, *distribution='normal'*, ***kwargs*)

Initialize module parameters with values according to the method described in Understanding the difficulty of training deep feedforward neural networks - Glorot, X. & Bengio, Y. (2010).

Parameters

- **gain** (*int | float*) – an optional scaling factor. Defaults to 1.
- **bias** (*int | float*) – the value to fill the bias. Defaults to 0.
- **bias_prob** (*float, optional*) – the probability for bias initialization. Defaults to None.
- **distribution** (*str*) – distribution either be `'normal'` or `'uniform'`. Defaults to `'normal'`.
- **layer** (*str | list[str], optional*) – the layer will be initialized. Defaults to None.

bias_init_with_prob	initialize conv/fc bias value according to a given probability value.
caffe2_xavier_init	
constant_init	
initialize	Initialize a module.
kaiming_init	
normal_init	
trunc_normal_init	
uniform_init	
update_init_info	Update the *_params_init_info* in the module if the value of parameters are changed.
xavier_init	

40.5.10 mmengine.model.bias_init_with_prob

mmengine.model.**bias_init_with_prob**(*prior_prob*)
initialize conv/fc bias value according to a given probability value.

40.5.11 mmengine.model.caffe2_xavier_init

mmengine.model.**caffe2_xavier_init**(*module*, *bias=0*)

40.5.12 mmengine.model.constant_init

mmengine.model.**constant_init**(*module*, *val*, *bias=0*)

40.5.13 mmengine.model.initialize

mmengine.model.**initialize**(*module*, *init_cfg*)
Initialize a module.

Parameters

- **module** (torch.nn.Module) – the module will be initialized.
- **init_cfg** (*dict | list[dict]*) – initialization configuration dict to define initializer. OpenMMLab has implemented 6 initializers including Constant, Xavier, Normal, Uniform, Kaiming, and Pretrained.

Example

```
>>> module = nn.Linear(2, 3, bias=True)
>>> init_cfg = dict(type='Constant', layer='Linear', val =1 , bias =2)
>>> initialize(module, init_cfg)
>>> module = nn.Sequential(nn.Conv1d(3, 1, 3), nn.Linear(1,2))
>>> # define key ``'layer'`` for initializing layer with different
>>> # configuration
>>> init_cfg = [dict(type='Constant', layer='Conv1d', val=1),
        dict(type='Constant', layer='Linear', val=2)]
>>> initialize(module, init_cfg)
>>> # define key``'override'`` to initialize some specific part in
>>> # module
>>> class FooNet(nn.Module):
>>>     def __init__(self):
>>>         super().__init__()
>>>         self.feat = nn.Conv2d(3, 16, 3)
>>>         self.reg = nn.Conv2d(16, 10, 3)
>>>         self.cls = nn.Conv2d(16, 5, 3)
>>> model = FooNet()
>>> init_cfg = dict(type='Constant', val=1, bias=2, layer='Conv2d',
>>>     override=dict(type='Constant', name='reg', val=3, bias=4))
>>> initialize(model, init_cfg)
>>> model = ResNet(depth=50)
>>> # Initialize weights with the pretrained model.
>>> init_cfg = dict(type='Pretrained',
        checkpoint='torchvision://resnet50')
>>> initialize(model, init_cfg)
>>> # Initialize weights of a sub-module with the specific part of
>>> # a pretrained model by using "prefix".
>>> url = 'http://download.openmmlab.com/mmdetection/v2.0/retinanet/'\
>>>     'retinanet_r50_fpn_1x_coco/'\
>>>     'retinanet_r50_fpn_1x_coco_20200130-c2398f9e.pth'
>>> init_cfg = dict(type='Pretrained',
        checkpoint=url, prefix='backbone.')
```

40.5.14 mmengine.model.kaiming_init

mmengine.model.**kaiming_init**(*module*, *a=0*, *mode='fan_out'*, *nonlinearity='relu'*, *bias=0*, *distribution='normal'*)

40.5.15 mmengine.model.normal_init

mmengine.model.**normal_init**(*module*, *mean=0*, *std=1*, *bias=0*)

40.5.16 mmengine.model.trunc_normal_init

mmengine.model.**trunc_normal_init**(*module*, *mean=0*, *std=1*, *a=- 2*, *b=2*, *bias=0*)

Parameters

- **module** (*torch.nn.modules.module.Module*) –
- **mean** (*float*) –
- **std** (*float*) –
- **a** (*float*) –
- **b** (*float*) –
- **bias** (*float*) –

Return type None

40.5.17 mmengine.model.uniform_init

mmengine.model.**uniform_init**(*module*, *a=0*, *b=1*, *bias=0*)

40.5.18 mmengine.model.update_init_info

mmengine.model.**update_init_info**(*module*, *init_info*)

Update the *_params_init_info* in the module if the value of parameters are changed.

Parameters

- **(obj** (*module*) – *nn.Module*): The module of PyTorch with a user-defined attribute *_params_init_info* which records the initialization information.
- **init_info** (*str*) – The string that describes the initialization.

40.5.19 mmengine.model.xavier_init

mmengine.model.**xavier_init**(*module*, *gain=1*, *bias=0*, *distribution='normal'*)

40.6 Utils

detect_anomalous_params	

merge_dict	Merge all dictionaries into one dictionary.
stack_batch	Stack multiple tensors to form a batch and pad the tensor to the max shape use the right bottom padding mode in these images.
revert_sync_batchnorm	Helper function to convert all *SyncBatchNorm* (SyncBN) and *mmcv.ops.sync_bn.SyncBatchNorm`(MMSyncBN) layers in the model to `BatchNormXd* layers.
convert_sync_batchnorm	Helper function to convert all *BatchNorm* layers in the model to *SyncBatchNorm* (SyncBN) or `mmcv.ops.sync_bn.SyncBatchNorm`(MMSyncBN) layers. Adapted from <https://pytorch.org/docs/stable/generated/torch.nn.SyncBatchNorm.html#torch.nn.SyncBatchNorm.convert_sync_batchnorm>_.

40.6.1 mmengine.model.detect_anomalous_params

mmengine.model.**detect_anomalous_params**(*loss*, *model*)

Parameters **loss** (*torch.Tensor*) –

Return type None

40.6.2 mmengine.model.merge_dict

mmengine.model.**merge_dict**(**args*)

Merge all dictionaries into one dictionary.

If pytorch version >= 1.8, `merge_dict` will be wrapped by `torch.fx.wrap`, which will make `torch.fx.symbolic_trace` skip trace `merge_dict`.

Note: If a function needs to be traced by `torch.fx.symbolic_trace`, but inevitably needs to use `update` method of `dict``(``)``update` is not traceable). It should use `merge_dict` to replace `xxx.update`.

Parameters ***args** – dictionary needs to be merged.

Returns Merged dict from args

Return type dict

40.6.3 mmengine.model.stack_batch

mmengine.model.**stack_batch**(*tensor_list*, *pad_size_divisor=1*, *pad_value=0*)

Stack multiple tensors to form a batch and pad the tensor to the max shape use the right bottom padding mode in these images. If `pad_size_divisor > 0`, add padding to ensure the shape of each dim is divisible by `pad_size_divisor`.

Parameters

- **tensor_list** (*List[Tensor]*) – A list of tensors with the same dim.
- **pad_size_divisor** (*int*) – If `pad_size_divisor > 0`, add padding to ensure the shape of each dim is divisible by `pad_size_divisor`. This depends on the model, and many models need to be divisible by 32. Defaults to 1
- **pad_value** (*int, float*) – The padding value. Defaults to 0.

Returns The n dim tensor.

Return type Tensor

40.6.4 mmengine.model.revert_sync_batchnorm

mmengine.model.**revert_sync_batchnorm**(*module*)

Helper function to convert all *SyncBatchNorm* (SyncBN) and *mmcv.ops.sync_bn.SyncBatchNorm`(MMSyncBN) layers in the model to `BatchNormXd* layers.

Adapted from @kapily's work: (https://github.com/pytorch/pytorch/issues/41081#issuecomment-783961547)

Parameters **module** (*nn.Module*) – The module containing *SyncBatchNorm* layers.

Returns The converted module with *BatchNormXd* layers.

Return type module_output

40.6.5 mmengine.model.convert_sync_batchnorm

mmengine.model.**convert_sync_batchnorm**(*module*, *implementation='torch'*)

Helper function to convert all *BatchNorm* layers in the model to *SyncBatchNorm* (SyncBN) or `mmcv.ops.sync_bn.SyncBatchNorm`(MMSyncBN) layers. Adapted from <https://pytorch.org/docs/stable/generated/torch.nn.Sy ncBatchNorm.html#torch.nn.SyncBatchNorm.convert_sync_batchnorm>_.

Parameters

- **module** (*nn.Module*) – The module containing *SyncBatchNorm* layers.
- **implementation** (*str*) – The type of *SyncBatchNorm* to convert to.
  - 'torch': convert to *torch.nn.modules.batchnorm.SyncBatchNorm*.
  - 'mmcv': convert to *mmcv.ops.sync_bn.SyncBatchNorm*.

Returns The converted module with *SyncBatchNorm* layers.

Return type nn.Module

CHAPTER

FORTYONE

MMENGINE.OPTIM

mmengine.optim

- *Optimizer*
- *Scheduler*

41.1 Optimizer

AmpOptimWrapper	A subclass of *OptimWrapper* that supports automatic mixed precision training based on torch.cuda.amp.
OptimWrapper	Optimizer wrapper provides a common interface for updating parameters.
OptimWrapperDict	A dictionary container of *OptimWrapper*.
DefaultOptimWrapperConstructor	Default constructor for optimizers.

41.1.1 AmpOptimWrapper

class mmengine.optim.**AmpOptimWrapper**(*loss_scale='dynamic'*, ***kwargs*)

A subclass of *OptimWrapper* that supports automatic mixed precision training based on torch.cuda.amp.

`AmpOptimWrapper` provides a unified interface with `OptimWrapper`, so `AmpOptimWrapper` can be used in the same way as `OptimWrapper`.

> **Warning:** `AmpOptimWrapper` requires PyTorch >= 1.6.

Parameters

- **loss_scale** (*float or str or dict*) – The initial configuration of *torch.cuda.amp.GradScaler*. See more specific arguments introduction at PyTorch AMP # noqa: E501 Defaults to `dynamic`.
  - "dynamic": Initialize GradScale without any arguments.
  - float: Initialize GradScaler with `init_scale`.
  - dict: Initialize GradScaler with more detail configuration.

- ****kwargs** – Keyword arguments passed to OptimWrapper.

Note: If you use `IterBasedRunner` and enable gradient accumulation, the original *max_iters* should be multiplied by `accumulative_counts`.

---

backward(*loss*, ***kwargs*)
Perform gradient back propagation with `loss_scaler`.

Parameters

- **loss** (*torch.Tensor*) – The loss of current iteration.
- **kwargs** – Keyword arguments passed to `torch.Tensor.backward()`

load_state_dict(*state_dict*)
Load and parse the state dictionary of `optimizer` and `loss_scaler`.

If state_dict contains "loss_scaler.", the `loss_scaler` will load the corresponding keys. Otherwise, only the `optimizer` will load the state dictionary.

Parameters **state_dict** (*dict*) – The state dict of `optimizer` and `loss_scaler`

optim_context(*model*)
Enables the context for mixed precision training, and enables the context for disabling gradient synchronization during gradient accumulation context.

Parameters **model** (*nn.Module*) – The training model.

state_dict()
Get the state dictionary of `optimizer` and `loss_scaler`.

Based on the state dictionary of the optimizer, the returned state dictionary will add a key named "loss_scaler".

Returns The merged state dict of `loss_scaler` and `optimizer`.

Return type dict

step(***kwargs*)
Update parameters with `loss_scaler`.

Parameters **kwargs** – Keyword arguments passed to `torch.optim.Optimizer.step()`.

41.1.2 OptimWrapper

class mmengine.optim.**OptimWrapper**(*optimizer*, *accumulative_counts=1*, *clip_grad=None*)
Optimizer wrapper provides a common interface for updating parameters.

Optimizer wrapper provides a unified interface for single precision training and automatic mixed precision training with different hardware. OptimWrapper encapsulates optimizer to provide simplified interfaces for commonly used training techniques such as gradient accumulative and grad clips. `OptimWrapper` implements the basic logic of gradient accumulation and gradient clipping based on `torch.optim.Optimizer`. The subclasses only need to override some methods to implement the mixed precision training. See more information in *AmpOptimWrapper*.

Parameters

- **optimizer** (*Optimizer*) – Optimizer used to update model parameters.

- **accumulative_counts** (*int*) – The number of iterations to accumulate gradients. The parameters will be updated per `accumulative_counts`.
- **clip_grad** (*dict, optional*) – If `clip_grad` is not None, it will be the arguments of `torch.nn.utils.clip_grad_norm_()` or `torch.nn.utils.clip_grad_value_()`. `clip_grad` should be a dict, and the keys could be set as follows:

  If the key `type` is not set, or `type` is "norm", the accepted keys are as follows:

  - max_norm (float or int): Max norm of the gradients.
  - norm_type (float or int): Type of the used p-norm. Can be `'inf'` for infinity norm.
  - error_if_nonfinite (bool): If True, an error is thrown if the total norm of the gradients from `parameters` is `nan`, `inf`, or `-inf`. Default: False (will switch to True in the future)

  If the key `type` is set to "value", the accepted keys are as follows:

  - clip_value (float or int): maximum allowed value of the gradients. The gradients are clipped in the range `(-clip_value, +clip_value)`.

Note: If `accumulative_counts` is larger than 1, perform *update_params()* under the context of `optim_context` could avoid unnecessary gradient synchronization.

Note: If you use `IterBasedRunner` and enable gradient accumulation, the original *max_iters* should be multiplied by `accumulative_counts`.

Note: The subclass should ensure that once *update_params()* is called, `_inner_count += 1` is automatically performed.

Examples

```
>>> # Config sample of OptimWrapper and enable clipping gradient by
>>> # norm.
>>> optim_wrapper_cfg = dict(
>>>     type='OptimWrapper',
>>>     _accumulative_counts=1,
>>>     clip_grad=dict(max_norm=0.2))
>>> # Config sample of OptimWrapper and enable clipping gradient by
>>> # value.
>>> optim_wrapper_cfg = dict(
>>>     type='OptimWrapper',
>>>     _accumulative_counts=1,
>>>     clip_grad=dict(type='value', clip_value=0.2))
>>> # Use OptimWrapper to update model.
>>> import torch.nn as nn
>>> import torch
>>> from torch.optim import SGD
>>> from torch.utils.data import DataLoader
>>> from mmengine.optim import OptimWrapper
>>>
```

(continues on next page)

(continued from previous page)

```
>>> model = nn.Linear(1, 1)
>>> dataset = torch.randn(10, 1, 1)
>>> dataloader = DataLoader(dataset)
>>> optimizer = SGD(model.parameters(), lr=0.1)
>>> optim_wrapper = OptimWrapper(optimizer)
>>>
>>> for data in dataloader:
>>>     loss = model(data)
>>>     optim_wrapper.update_params(loss)
>>> # Enable gradient accumulation
>>> optim_wrapper_cfg = dict(
>>>     type='OptimWrapper',
>>>     _accumulative_counts=3,
>>>     clip_grad=dict(max_norm=0.2))
>>> ddp_model = DistributedDataParallel(model)
>>> optimizer = SGD(ddp_model.parameters(), lr=0.1)
>>> optim_wrapper = OptimWrapper(optimizer)
>>> optim_wrapper.initialize_count_status(0, len(dataloader))
>>> # If model is a subclass instance of DistributedDataParallel,
>>> # `optim_context` context manager can avoid unnecessary gradient
>>> #  synchronize.
>>> for iter, data in enumerate(dataloader):
>>>     with optim_wrapper.optim_context(ddp_model):
>>>         loss = model(data)
>>>     optim_wrapper.update_params(loss)
```

backward(*loss*, ***kwargs*)

Perform gradient back propagation.

Provide unified `backward` interface compatible with automatic mixed precision training. Subclass can overload this method to implement the required logic. For example, `torch.cuda.amp` require some extra operation on GradScaler during backward process.

Note: If subclasses inherit from `OptimWrapper` override `backward`, `_inner_count +=1` must be implemented.

Parameters

- **loss** (*torch.Tensor*) – The loss of current iteration.
- **kwargs** – Keyword arguments passed to `torch.Tensor.backward()`.

Return type None

property defaults: dict

A wrapper of `Optimizer.defaults`.

Make OptimizeWrapper compatible with *_ParamScheduler*.

Returns the `param_groups` of `optimizer`.

Return type dict

get_lr()

Get the learning rate of the optimizer.

Provide unified interface to get learning rate of optimizer.

Returns Learning rate of the optimizer.

Return type Dict[str, List[float]]

`get_momentum()`
Get the momentum of the optimizer.

Provide unified interface to get momentum of optimizer.

Returns Momentum of the optimizer.

Return type Dict[str, List[float]]

`initialize_count_status(model, init_counts, max_counts)`
Initialize gradient accumulation related attributes.

`OptimWrapper` can be used without calling `initialize_iter_status`. However, Consider the case of `len( dataloader) == 10`, and the `accumulative_iter == 3`. Since 10 is not divisible by 3, the last iteration will not trigger `optimizer.step()`, resulting in one less parameter updating.

Parameters

- **model** (*nn.Module*) – Training model
- **init_counts** (*int*) – The initial value of the inner count.
- **max_counts** (*int*) – The maximum value of the inner count.

Return type None

`property inner_count`
Get the number of updating parameters of optimizer wrapper.

`load_state_dict(state_dict)`
A wrapper of `Optimizer.load_state_dict`. load the state dict of `optimizer`.

Provide unified `load_state_dict` interface compatible with automatic mixed precision training. Subclass can overload this method to implement the required logic. For example, the state dictionary of GradScaler should be loaded when training with `torch.cuda.amp`.

Parameters **state_dict** (*dict*) – The state dictionary of `optimizer`.

Return type None

`optim_context(model)`
A Context for gradient accumulation and automatic mix precision training.

If subclasses need to enable the context for mix precision training, e.g., `:class:`AmpOptimWrapper`, the corresponding context should be enabled in *optim_context*. Since `OptimWrapper` uses default fp32 training, `optim_context` will only enable the context for blocking the unnecessary gradient synchronization during gradient accumulation

If model is an instance with `no_sync` method (which means blocking the gradient synchronization) and `self._accumulative_counts != 1`. The model will not automatically synchronize gradients if `cur_iter` is divisible by `self._accumulative_counts`. Otherwise, this method will enable an empty context.

Parameters **model** (*nn.Module*) – The training model.

`property param_groups: List[dict]`
A wrapper of `Optimizer.param_groups`.

Make OptimizeWrapper compatible with *_ParamScheduler*.

Returns the `param_groups` of `optimizer`.

Return type dict

`scale_loss`(*loss*)
Get scaled loss according to `_accumulative_counts`, `_inner_count` and max_counts.

Parameters **loss** (*torch.Tensor*) – Original loss calculated by model.

Returns Scaled loss.

Return type loss (torch.Tensor)

`should_sync`()
Decide whether the automatic gradient synchronization should be allowed at the current iteration.

It takes effect when gradient accumulation is used to skip synchronization at the iterations where the parameter is not updated.

Since `should_sync` is called by *optim_context()*, and it is called before *backward()* which means `self._inner_count += 1` has not happened yet. Therefore, `self._inner_count += 1` should be performed manually here.

Returns Whether to block the automatic gradient synchronization.

Return type bool

`should_update`()
Decide whether the parameters should be updated at the current iteration.

Called by *update_params()* and check whether the optimizer wrapper should update parameters at current iteration.

Returns Whether to update parameters.

Return type bool

`state_dict`()
A wrapper of `Optimizer.state_dict`.

Provide unified `state_dict` interface compatible with automatic mixed precision training. Subclass can overload this method to implement the required logic. For example, the state dictionary of GradScaler should be saved when training with `torch.cuda.amp`.

Returns The state dictionary of `optimizer`.

Return type dict

`step`(***kwargs*)
A wrapper of `Optimizer.step`.

Provide unified `step` interface compatible with automatic mixed precision training. Subclass can overload this method to implement the required logic. For example, `torch.cuda.amp` require some extra operation on `GradScaler` during step process.

Clip grad if `clip_grad_kwargs` is not None, and then update parameters.

Parameters **kwargs** – Keyword arguments passed to `torch.optim.Optimizer.step()`.

Return type None

`update_params`(*loss*)
Update parameters in `optimizer`.

Parameters **loss** (*torch.Tensor*) – A tensor for back propagation.

Return type None

zero_grad(***kwargs*)
A wrapper of `Optimizer.zero_grad`.

Provide unified `zero_grad` interface compatible with automatic mixed precision training. Subclass can overload this method to implement the required logic.

Parameters **kwargs** – Keyword arguments passed to `torch.optim.Optimizer.zero_grad()`.

Return type None

41.1.3 OptimWrapperDict

class mmengine.optim.**OptimWrapperDict**(***optim_wrapper_dict*)
A dictionary container of *OptimWrapper*.

If runner is training with multiple optimizers, all optimizer wrappers should be managed by *OptimWrapperDict* which is built by `CustomOptimWrapperConstructor`. `OptimWrapperDict` will load and save the state dictionary of all optimizer wrappers.

Consider the semantic ambiguity of calling :meth:`update_params`, *backward()* of all optimizer wrappers, `OptimWrapperDict` will not implement these methods.

Examples

```
>>> import torch.nn as nn
>>> from torch.optim import SGD
>>> from mmengine.optim import OptimWrapperDict, OptimWrapper
>>> model1 = nn.Linear(1, 1)
>>> model2 = nn.Linear(1, 1)
>>> optim_wrapper1 = OptimWrapper(SGD(model1.parameters(), lr=0.1))
>>> optim_wrapper2 = OptimWrapper(SGD(model2.parameters(), lr=0.1))
>>> optim_wrapper_dict = OptimWrapperDict(model1=optim_wrapper1,
>>>                                       model2=optim_wrapper2)
```

Note: The optimizer wrapper contained in `OptimWrapperDict` can be accessed in the same way as *dict*.

Parameters

- ****optim_wrappers** – A dictionary of `OptimWrapper` instance.
- **optim_wrapper_dict** (`mmengine.optim.optimizer.optimizer_wrapper.OptimWrapper`) –

backward(*loss*, ***kwargs*)
Since OptimWrapperDict doesn't know which optimizer wrapper's backward method should be called (`loss_scaler` maybe different in different :obj:AmpOptimWrapper), this method is not implemented.

The optimizer wrapper of OptimWrapperDict should be accessed and call its `backward.

Parameters **loss** (*torch.Tensor*) –

Return type None

get_lr()
Get the learning rate of all optimizers.

Returns Learning rate of all optimizers.

Return type Dict[str, List[float]]

get_momentum()
Get the momentum of all optimizers.

Returns momentum of all optimizers.

Return type Dict[str, List[float]]

initialize_count_status(*model*, *cur_iter*, *max_iters*)
Do nothing but provide unified interface for *OptimWrapper*

Since `OptimWrapperDict` does not know the correspondence between model and optimizer wrapper. `initialize_iter_status` will do nothing and each optimizer wrapper should call `initialize_iter_status` separately.

Parameters **model** (*torch.nn.modules.module.Module*) –

Return type None

items()
A generator to get the name and corresponding *OptimWrapper*

Return type Iterator[Tuple[str, *mmengine.optim.optimizer.optimizer_wrapper.OptimWrapper*]]

keys()
A generator to get the name of *OptimWrapper*

Return type Iterator[str]

load_state_dict(*state_dict*)
Load the state dictionary from the `state_dict`.

Parameters **state_dict** (*dict*) – Each key-value pair in *state_dict* represents the name and the state dictionary of corresponding *OptimWrapper*.

Return type None

optim_context(*model*)
`optim_context` should be called by each optimizer separately.

Parameters **model** (*torch.nn.modules.module.Module*) –

property param_groups
Returns the parameter groups of each OptimWrapper.

state_dict()
Get the state dictionary of all optimizer wrappers.

Returns Each key-value pair in the dictionary represents the name and state dictionary of corresponding *OptimWrapper*.

Return type dict

step(***kwargs*)
Since the backward method is not implemented, the step should not be implemented either.

Return type None

update_params(*loss*)
Update all optimizer wrappers would lead to a duplicate backward errors, and OptimWrapperDict does not know which optimizer wrapper should be updated.

Therefore, this method is not implemented. The optimizer wrapper of OptimWrapperDict should be accessed and call its `update_params.

Parameters **loss** (*torch.Tensor*) –

Return type None

values()
A generator to get *OptimWrapper*

Return type Iterator[*mmengine.optim.optimizer.optimizer_wrapper.OptimWrapper*]

zero_grad(***kwargs*)
Set the gradients of all optimizer wrappers to zero.

Return type None

41.1.4 DefaultOptimWrapperConstructor

class mmengine.optim.**DefaultOptimWrapperConstructor**(*optim_wrapper_cfg*, *paramwise_cfg=None*)
Default constructor for optimizers.

By default, each parameter share the same optimizer settings, and we provide an argument `paramwise_cfg` to specify parameter-wise settings. It is a dict and may contain the following fields:

- `custom_keys` (dict): Specified parameters-wise settings by keys. If one of the keys in `custom_keys` is a substring of the name of one parameter, then the setting of the parameter will be specified by `custom_keys[key]` and other setting like `bias_lr_mult` etc. will be ignored. It should be noted that the aforementioned `key` is the longest key that is a substring of the name of the parameter. If there are multiple matched keys with the same length, then the key with lower alphabet order will be chosen. `custom_keys[key]` should be a dict and may contain fields `lr_mult` and `decay_mult`. See Example 2 below.
- `bias_lr_mult` (float): It will be multiplied to the learning rate for all bias parameters (except for those in normalization layers and offset layers of DCN).
- `bias_decay_mult` (float): It will be multiplied to the weight decay for all bias parameters (except for those in normalization layers, depthwise conv layers, offset layers of DCN).
- `norm_decay_mult` (float): It will be multiplied to the weight decay for all weight and bias parameters of normalization layers.
- `dwconv_decay_mult` (float): It will be multiplied to the weight decay for all weight and bias parameters of depthwise conv layers.
- `dcn_offset_lr_mult` (float): It will be multiplied to the learning rate for parameters of offset layer in the deformable convs of a model.
- `bypass_duplicate` (bool): If true, the duplicate parameters would not be added into optimizer. Default: False.

Note: 1. If the option `dcn_offset_lr_mult` is used, the constructor will override the effect of `bias_lr_mult` in the bias of offset layer. So be careful when using both `bias_lr_mult` and `dcn_offset_lr_mult`. If you wish to apply both of them to the offset layer in deformable convs, set `dcn_offset_lr_mult` to the original `dcn_offset_lr_mult` * `bias_lr_mult`.

2. If the option `dcn_offset_lr_mult` is used, the constructor will apply it to all the DCN layers in the model. So be careful when the model contains multiple DCN layers in places other than backbone.

Parameters

- **optim_wrapper_cfg** (*dict*) – The config dict of the optimizer wrapper. Positional fields are
  - `type`: class name of the OptimizerWrapper
  - `optimizer`: The configuration of optimizer.

  Optional fields are
  - any arguments of the corresponding optimizer wrapper type, e.g., accumulative_counts, clip_grad, etc.
- **positional fields of optimizer are** (*The*) –
  - *type*: class name of the optimizer.
- **fields are** (*Optional*) –
  - any arguments of the corresponding optimizer type, e.g., lr, weight_decay, momentum, etc.
- **paramwise_cfg** (*dict, optional*) – Parameter-wise options.

Example 1:

```
>>> model = torch.nn.modules.Conv1d(1, 1, 1)
>>> optim_wrapper_cfg = dict(
>>>     dict(type='OptimWrapper', optimizer=dict(type='SGD', lr=0.01,
>>>         momentum=0.9, weight_decay=0.0001))
>>> paramwise_cfg = dict(norm_decay_mult=0.)
>>> optim_wrapper_builder = DefaultOptimWrapperConstructor(
>>>     optim_wrapper_cfg, paramwise_cfg)
>>> optim_wrapper = optim_wrapper_builder(model)
```

Example 2:

```
>>> # assume model have attribute model.backbone and model.cls_head
>>> optim_wrapper_cfg = dict(type='OptimWrapper', optimizer=dict(
>>>     type='SGD', lr=0.01, weight_decay=0.95))
>>> paramwise_cfg = dict(custom_keys={
>>>     'backbone': dict(lr_mult=0.1, decay_mult=0.9)})
>>> optim_wrapper_builder = DefaultOptimWrapperConstructor(
>>>     optim_wrapper_cfg, paramwise_cfg)
>>> optim_wrapper = optim_wrapper_builder(model)
>>> # Then the `lr` and `weight_decay` for model.backbone is
>>> # (0.01 * 0.1, 0.95 * 0.9). `lr` and `weight_decay` for
>>> # model.cls_head is (0.01, 0.95).
```

add_params(*params*, *module*, *prefix=''*, *is_dcn_module=None*)

Add all parameters of module to the params list.

The parameters of the given module will be added to the list of param groups, with specific rules defined by paramwise_cfg.

Parameters

- **params** (*list[dict]*) – A list of param groups, it will be modified in place.
- **module** (*nn.Module*) – The module to be added.
- **prefix** (*str*) – The prefix of the module
- **is_dcn_module** (*int|float|None*) – If the current module is a submodule of DCN, *is_dcn_module* will be passed to control conv_offset layer's learning rate. Defaults to None.

Return type None

build_optim_wrapper	Build function of OptimWrapper.

41.1.5 mmengine.optim.build_optim_wrapper

mmengine.optim.**build_optim_wrapper**(*model*, *cfg*)

Build function of OptimWrapper.

If `constructor` is set in the `cfg`, this method will build an optimizer wrapper constructor, and use optimizer wrapper constructor to build the optimizer wrapper. If `constructor` is not set, the `DefaultOptimWrapperConstructor` will be used by default.

Parameters

- **model** (*nn.Module*) – Model to be optimized.
- **cfg** (*dict*) – Config of optimizer wrapper, optimizer constructor and optimizer.

Returns The built optimizer wrapper.

Return type *OptimWrapper*

41.2 Scheduler

_ParamScheduler	Base class for parameter schedulers.
ConstantLR	Decays the learning rate value of each parameter group by a small constant factor until the number of epoch reaches a pre-defined milestone: end.
ConstantMomentum	Decays the momentum value of each parameter group by a small constant factor until the number of epoch reaches a pre-defined milestone: end.
ConstantParamScheduler	Decays the parameter value of each parameter group by a small constant factor until the number of epoch reaches a pre-defined milestone: end.
CosineAnnealingLR	Set the learning rate of each parameter group using a cosine annealing schedule, where η_{max} is set to the initial value and T_{cur} is the number of epochs since the last restart in SGDR:
CosineAnnealingMomentum	Set the momentum of each parameter group using a cosine annealing schedule, where η_{max} is set to the initial value and T_{cur} is the number of epochs since the last restart in SGDR:

continues on next page

Table 3 – continued from previous page

CosineAnnealingParamScheduler	Set the parameter value of each parameter group using a cosine annealing schedule, where η_{max} is set to the initial value and T_{cur} is the number of epochs since the last restart in SGDR:
ExponentialLR	Decays the learning rate of each parameter group by gamma every epoch.
ExponentialMomentum	Decays the momentum of each parameter group by gamma every epoch.
ExponentialParamScheduler	Decays the parameter value of each parameter group by gamma every epoch.
LinearLR	Decays the learning rate of each parameter group by linearly changing small multiplicative factor until the number of epoch reaches a pre-defined milestone: end.
LinearMomentum	Decays the momentum of each parameter group by linearly changing small multiplicative factor until the number of epoch reaches a pre-defined milestone: end.
LinearParamScheduler	Decays the parameter value of each parameter group by linearly changing small multiplicative factor until the number of epoch reaches a pre-defined milestone: end.
MultiStepLR	Decays the specified learning rate in each parameter group by gamma once the number of epoch reaches one of the milestones.
MultiStepMomentum	Decays the specified momentum in each parameter group by gamma once the number of epoch reaches one of the milestones.
MultiStepParamScheduler	Decays the specified parameter in each parameter group by gamma once the number of epoch reaches one of the milestones.
OneCycleLR	Sets the learning rate of each parameter group according to the 1cycle learning rate policy.
OneCycleParamScheduler	Sets the parameters of each parameter group according to the 1cycle learning rate policy.
PolyLR	Decays the learning rate of each parameter group in a polynomial decay scheme.
PolyMomentum	Decays the momentum of each parameter group in a polynomial decay scheme.
PolyParamScheduler	Decays the parameter value of each parameter group in a polynomial decay scheme.
StepLR	Decays the learning rate of each parameter group by gamma every step_size epochs.
StepMomentum	Decays the momentum of each parameter group by gamma every step_size epochs.
StepParamScheduler	Decays the parameter value of each parameter group by gamma every step_size epochs.

41.2.1 _ParamScheduler

class mmengine.optim._ParamScheduler(*optimizer*, *param_name*, *begin=0*, *end=1000000000*, *last_step=- 1*, *by_epoch=True*, *verbose=False*)

Base class for parameter schedulers.

It should be inherited by all schedulers that schedule parameters in the optimizer's `param_groups`. All subclasses should overwrite the `_get_value()` according to their own schedule strategy. The implementation is motivated by https://github.com/pytorch/pytorch/blob/master/torch/optim/lr_scheduler.py.

Parameters

- **optimizer** (*OptimWrapper or Optimizer*) – Wrapped optimizer.
- **param_name** (*str*) – Name of the parameter to be adjusted, such as `lr`, `momentum`.
- **begin** (*int*) – Step at which to start updating the parameters. Defaults to 0.
- **end** (*int*) – Step at which to stop updating the parameters. Defaults to INF.
- **last_step** (*int*) – The index of last step. Used for resuming without state dict. Default value `-1` means the `step` function is never be called before. Defaults to -1.
- **by_epoch** (*bool*) – Whether the scheduled parameters are updated by epochs. Defaults to True.
- **verbose** (*bool*) – Whether to print the value for each update. Defaults to False.

get_last_value()

Return the last computed value by current scheduler.

Returns A list of the last computed value of the optimizer's `param_group`.

Return type list

load_state_dict(*state_dict*)

Loads the schedulers state.

Parameters **state_dict** (*dict*) – scheduler state. Should be an object returned from a call to *state_dict()*.

print_value(*is_verbose*, *group*, *value*)

Display the current parameter value.

Parameters

- **is_verbose** (*bool*) – Whether to print the value.
- **group** (*int*) – The index of the current `param_group`.
- **value** (*float*) – The parameter value.

state_dict()

Returns the state of the scheduler as a `dict`.

It contains an entry for every variable in self.__dict__ which is not the optimizer.

Returns scheduler state.

Return type dict

step()

Adjusts the parameter value of each parameter group based on the specified schedule.

41.2.2 ConstantLR

class mmengine.optim.ConstantLR(*optimizer*, **args*, ***kwargs*)

Decays the learning rate value of each parameter group by a small constant factor until the number of epoch reaches a pre-defined milestone: end. Notice that such decay can happen simultaneously with other changes to the learning rate value from outside this scheduler.

Parameters

- **optimizer** (*Optimizer or* OptimWrapper) – Wrapped optimizer.
- **factor** (*float*) – The number we multiply learning rate until the milestone. Defaults to 1./3.
- **begin** (*int*) – Step at which to start updating the learning rate. Defaults to 0.
- **end** (*int*) – Step at which to stop updating the learning rate. Defaults to INF.
- **last_step** (*int*) – The index of last step. Used for resume without state dict. Defaults to -1.
- **by_epoch** (*bool*) – Whether the scheduled learning rate is updated by epochs. Defaults to True.
- **verbose** (*bool*) – Whether to print the learning rate for each update. Defaults to False.

41.2.3 ConstantMomentum

class mmengine.optim.ConstantMomentum(*optimizer*, **args*, ***kwargs*)

Decays the momentum value of each parameter group by a small constant factor until the number of epoch reaches a pre-defined milestone: end. Notice that such decay can happen simultaneously with other changes to the momentum value from outside this scheduler.

Parameters

- **optimizer** (*Optimizer or* OptimWrapper) – optimizer or Wrapped optimizer.
- **factor** (*float*) – The number we multiply momentum until the milestone. Defaults to 1./3.
- **begin** (*int*) – Step at which to start updating the momentum. Defaults to 0.
- **end** (*int*) – Step at which to stop updating the momentum. Defaults to INF.
- **last_step** (*int*) – The index of last step. Used for resume without state dict. Defaults to -1.
- **by_epoch** (*bool*) – Whether the scheduled momentum is updated by epochs. Defaults to True.
- **verbose** (*bool*) – Whether to print the momentum for each update. Defaults to False.

41.2.4 ConstantParamScheduler

class mmengine.optim.**ConstantParamScheduler**(*optimizer*, *param_name*, *factor=0.3333333333333333*, *begin=0*, *end=1000000000*, *last_step=- 1*, *by_epoch=True*, *verbose=False*)

Decays the parameter value of each parameter group by a small constant factor until the number of epoch reaches a pre-defined milestone: `end`. Notice that such decay can happen simultaneously with other changes to the parameter value from outside this scheduler.

Parameters

- **optimizer** (*Optimizer or* `OptimWrapper`) – optimizer or Wrapped optimizer.
- **param_name** (*str*) – Name of the parameter to be adjusted, such as `lr`, `momentum`.
- **factor** (*float*) – The number we multiply parameter value until the milestone. Defaults to 1./3.
- **begin** (*int*) – Step at which to start updating the parameters. Defaults to 0.
- **end** (*int*) – Step at which to stop updating the parameters. Defaults to INF.
- **last_step** (*int*) – The index of last step. Used for resume without state dict. Defaults to -1.
- **by_epoch** (*bool*) – Whether the scheduled parameters are updated by epochs. Defaults to True.
- **verbose** (*bool*) – Whether to print the value for each update. Defaults to False.

classmethod **build_iter_from_epoch**(**args*, *begin=0*, *end=1000000000*, *by_epoch=True*, *epoch_length=None*, ***kwargs*)

Build an iter-based instance of this scheduler from an epoch-based config.

41.2.5 CosineAnnealingLR

class mmengine.optim.**CosineAnnealingLR**(*optimizer*, **args*, ***kwargs*)

Set the learning rate of each parameter group using a cosine annealing schedule, where η_{max} is set to the initial value and T_{cur} is the number of epochs since the last restart in SGDR:

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 + \cos\left(\frac{T_{cur}}{T_{max}}\pi\right)\right), \quad T_{cur} \neq (2k+1)T_{max};$$
$$\eta_{t+1} = \eta_t + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 - \cos\left(\frac{1}{T_{max}}\pi\right)\right), \quad T_{cur} = (2k+1)T_{max}.$$

Notice that because the schedule is defined recursively, the learning rate can be simultaneously modified outside this scheduler by other operators. If the learning rate is set solely by this scheduler, the learning rate at each step becomes:

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 + \cos\left(\frac{T_{cur}}{T_{max}}\pi\right)\right)$$

It has been proposed in SGDR: Stochastic Gradient Descent with Warm Restarts. Note that this only implements the cosine annealing part of SGDR, and not the restarts.

Parameters

- **optimizer** (*Optimizer or* `OptimWrapper`) – Wrapped optimizer.
- **T_max** (*int*) – Maximum number of iterations.

- **eta_min** (*float*) – Minimum learning rate. Defaults to 0.
- **begin** (*int*) – Step at which to start updating the learning rate. Defaults to 0.
- **end** (*int*) – Step at which to stop updating the learning rate. Defaults to INF.
- **last_step** (*int*) – The index of last step. Used for resume without state dict. Defaults to -1.
- **by_epoch** (*bool*) – Whether the scheduled learning rate is updated by epochs. Defaults to True.
- **verbose** (*bool*) – Whether to print the learning rate for each update. Defaults to False.

41.2.6 CosineAnnealingMomentum

class mmengine.optim.**CosineAnnealingMomentum**(*optimizer*, **args*, ***kwargs*)

Set the momentum of each parameter group using a cosine annealing schedule, where η_{max} is set to the initial value and T_{cur} is the number of epochs since the last restart in SGDR:

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 + \cos\left(\frac{T_{cur}}{T_{max}}\pi\right)\right), \quad T_{cur} \neq (2k+1)T_{max};$$
$$\eta_{t+1} = \eta_t + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 - \cos\left(\frac{1}{T_{max}}\pi\right)\right), \quad T_{cur} = (2k+1)T_{max}.$$

Notice that because the schedule is defined recursively, the momentum can be simultaneously modified outside this scheduler by other operators. If the momentum is set solely by this scheduler, the momentum at each step becomes:

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 + \cos\left(\frac{T_{cur}}{T_{max}}\pi\right)\right)$$

It has been proposed in SGDR: Stochastic Gradient Descent with Warm Restarts. Note that this only implements the cosine annealing part of SGDR, and not the restarts.

Parameters

- **optimizer** (*Optimizer or* OptimWrapper) – optimizer or Wrapped optimizer.
- **T_max** (*int*) – Maximum number of iterations.
- **eta_min** (*float*) – Minimum momentum value. Defaults to 0.
- **begin** (*int*) – Step at which to start updating the momentum. Defaults to 0.
- **end** (*int*) – Step at which to stop updating the momentum. Defaults to INF.
- **last_step** (*int*) – The index of last step. Used for resume without state dict. Defaults to -1.
- **by_epoch** (*bool*) – Whether the scheduled momentum is updated by epochs. Defaults to True.
- **verbose** (*bool*) – Whether to print the momentum for each update. Defaults to False.

41.2.7 CosineAnnealingParamScheduler

class mmengine.optim.**CosineAnnealingParamScheduler**(*optimizer*, *param_name*, *T_max=None*, *eta_min=0.0*, *begin=0*, *end=1000000000*, *last_step=- 1*, *by_epoch=True*, *verbose=False*)

Set the parameter value of each parameter group using a cosine annealing schedule, where η_{max} is set to the initial value and T_{cur} is the number of epochs since the last restart in SGDR:

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 + \cos\left(\frac{T_{cur}}{T_{max}}\pi\right)\right), \quad T_{cur} \neq (2k+1)T_{max};$$
$$\eta_{t+1} = \eta_t + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 - \cos\left(\frac{1}{T_{max}}\pi\right)\right), \quad T_{cur} = (2k+1)T_{max}.$$

Notice that because the schedule is defined recursively, the parameter value can be simultaneously modified outside this scheduler by other operators. If the parameter value is set solely by this scheduler, the parameter value at each step becomes:

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 + \cos\left(\frac{T_{cur}}{T_{max}}\pi\right)\right)$$

It has been proposed in SGDR: Stochastic Gradient Descent with Warm Restarts. Note that this only implements the cosine annealing part of SGDR, and not the restarts.

Parameters

- **optimizer** (*Optimizer or* OptimWrapper) – optimizer or Wrapped optimizer.
- **param_name** (*str*) – Name of the parameter to be adjusted, such as `lr`, `momentum`.
- **T_max** (*int, optional*) – Maximum number of iterations. If not specified, use `end - begin`. Defaults to None.
- **eta_min** (*float*) – Minimum parameter value. Defaults to 0.
- **begin** (*int*) – Step at which to start updating the parameters. Defaults to 0.
- **end** (*int*) – Step at which to stop updating the parameters. Defaults to INF.
- **last_step** (*int*) – The index of last step. Used for resume without state dict. Defaults to -1.
- **by_epoch** (*bool*) – Whether the scheduled parameters are updated by epochs. Defaults to True.
- **verbose** (*bool*) – Whether to print the value for each update. Defaults to False.

classmethod **build_iter_from_epoch**(**args*, *T_max*, *begin=0*, *end=1000000000*, *by_epoch=True*, *epoch_length=None*, ***kwargs*)

Build an iter-based instance of this scheduler from an epoch-based config.

41.2.8 ExponentialLR

class mmengine.optim.**ExponentialLR**(*optimizer*, **args*, ***kwargs*)

Decays the learning rate of each parameter group by gamma every epoch.

Parameters

- **optimizer** (*Optimizer or* OptimWrapper) – Wrapped optimizer.
- **gamma** (*float*) – Multiplicative factor of learning rate decay.

- **begin** (*int*) – Step at which to start updating the learning rate. Defaults to 0.
- **end** (*int*) – Step at which to stop updating the learning rate. Defaults to INF.
- **last_step** (*int*) – The index of last step. Used for resume without state dict. Defaults to -1.
- **by_epoch** (*bool*) – Whether the scheduled learning rate is updated by epochs. Defaults to True.
- **verbose** (*bool*) – Whether to print the learning rate for each update. Defaults to False.

41.2.9 ExponentialMomentum

class mmengine.optim.**ExponentialMomentum**(*optimizer*, **args*, ***kwargs*)

Decays the momentum of each parameter group by gamma every epoch.

Parameters

- **optimizer** (*Optimizer or* OptimWrapper) – optimizer or Wrapped optimizer.
- **gamma** (*float*) – Multiplicative factor of momentum value decay.
- **begin** (*int*) – Step at which to start updating the momentum. Defaults to 0.
- **end** (*int*) – Step at which to stop updating the momentum. Defaults to INF.
- **last_step** (*int*) – The index of last step. Used for resume without state dict. Defaults to -1.
- **by_epoch** (*bool*) – Whether the scheduled momentum is updated by epochs. Defaults to True.
- **verbose** (*bool*) – Whether to print the momentum for each update. Defaults to False.

41.2.10 ExponentialParamScheduler

class mmengine.optim.**ExponentialParamScheduler**(*optimizer*, *param_name*, *gamma*, *begin=0*, *end=1000000000*, *last_step=- 1*, *by_epoch=True*, *verbose=False*)

Decays the parameter value of each parameter group by gamma every epoch.

Parameters

- **optimizer** (*Optimizer or* OptimWrapper) – optimizer or Wrapped optimizer.
- **param_name** (*str*) – Name of the parameter to be adjusted, such as `lr`, `momentum`.
- **gamma** (*float*) – Multiplicative factor of parameter value decay.
- **begin** (*int*) – Step at which to start updating the parameters. Defaults to 0.
- **end** (*int*) – Step at which to stop updating the parameters. Defaults to INF.
- **last_step** (*int*) – The index of last step. Used for resume without state dict. Defaults to -1.
- **by_epoch** (*bool*) – Whether the scheduled parameters are updated by epochs. Defaults to True.
- **verbose** (*bool*) – Whether to print the value for each update. Defaults to False.

classmethod build_iter_from_epoch(**args*, *begin=0*, *end=1000000000*, *by_epoch=True*, *epoch_length=None*, ***kwargs*)

Build an iter-based instance of this scheduler from an epoch-based config.

41.2.11 LinearLR

class mmengine.optim.**LinearLR**(*optimizer*, **args*, ***kwargs*)

Decays the learning rate of each parameter group by linearly changing small multiplicative factor until the number of epoch reaches a pre-defined milestone: end.

Notice that such decay can happen simultaneously with other changes to the learning rate from outside this scheduler. :param optimizer: Wrapped optimizer. :type optimizer: Optimizer or OptimWrapper :param start_factor: The number we multiply learning rate in the

first epoch. The multiplication factor changes towards end_factor in the following epochs. Defaults to 1./3.

Parameters

- **end_factor** (*float*) – The number we multiply learning rate at the end of linear changing process. Defaults to 1.0.
- **begin** (*int*) – Step at which to start updating the learning rate. Defaults to 0.
- **end** (*int*) – Step at which to stop updating the learning rate. Defaults to INF.
- **last_step** (*int*) – The index of last step. Used for resume without state dict. Defaults to -1.
- **by_epoch** (*bool*) – Whether the scheduled learning rate is updated by epochs. Defaults to True.
- **verbose** (*bool*) – Whether to print the learning rate for each update. Defaults to False.

41.2.12 LinearMomentum

class mmengine.optim.**LinearMomentum**(*optimizer*, **args*, ***kwargs*)

Decays the momentum of each parameter group by linearly changing small multiplicative factor until the number of epoch reaches a pre-defined milestone: end.

Notice that such decay can happen simultaneously with other changes to the momentum from outside this scheduler. :param optimizer: optimizer or Wrapped

optimizer.

Parameters

- **start_factor** (*float*) – The number we multiply momentum in the first epoch. The multiplication factor changes towards end_factor in the following epochs. Defaults to 1./3.
- **end_factor** (*float*) – The number we multiply momentum at the end of linear changing process. Defaults to 1.0.
- **begin** (*int*) – Step at which to start updating the momentum. Defaults to 0.
- **end** (*int*) – Step at which to stop updating the momentum. Defaults to INF.
- **last_step** (*int*) – The index of last step. Used for resume without state dict. Defaults to -1.

- **by_epoch** (*bool*) – Whether the scheduled momentum is updated by epochs. Defaults to True.
- **verbose** (*bool*) – Whether to print the momentum for each update. Defaults to False.

41.2.13 LinearParamScheduler

class mmengine.optim.**LinearParamScheduler**(*optimizer*, *param_name*, *start_factor=0.3333333333333333*, *end_factor=1.0*, *begin=0*, *end=1000000000*, *last_step=- 1*, *by_epoch=True*, *verbose=False*)

Decays the parameter value of each parameter group by linearly changing small multiplicative factor until the number of epoch reaches a pre-defined milestone: end.

Notice that such decay can happen simultaneously with other changes to the parameter value from outside this scheduler.

Parameters

- **optimizer** (*Optimizer or* OptimWrapper) – optimizer or Wrapped optimizer.
- **param_name** (*str*) – Name of the parameter to be adjusted, such as lr, momentum.
- **start_factor** (*float*) – The number we multiply parameter value in the first epoch. The multiplication factor changes towards end_factor in the following epochs. Defaults to 1./3.
- **end_factor** (*float*) – The number we multiply parameter value at the end of linear changing process. Defaults to 1.0.
- **begin** (*int*) – Step at which to start updating the parameters. Defaults to 0.
- **end** (*int*) – Step at which to stop updating the parameters. Defaults to INF.
- **last_step** (*int*) – The index of last step. Used for resume without state dict. Defaults to -1.
- **by_epoch** (*bool*) – Whether the scheduled parameters are updated by epochs. Defaults to True.
- **verbose** (*bool*) – Whether to print the value for each update. Defaults to False.

classmethod **build_iter_from_epoch**(**args*, *begin=0*, *end=1000000000*, *by_epoch=True*, *epoch_length=None*, ***kwargs*)

Build an iter-based instance of this scheduler from an epoch-based config.

41.2.14 MultiStepLR

class mmengine.optim.**MultiStepLR**(*optimizer*, **args*, ***kwargs*)

Decays the specified learning rate in each parameter group by gamma once the number of epoch reaches one of the milestones. Notice that such decay can happen simultaneously with other changes to the learning rate from outside this scheduler.

Parameters

- **optimizer** (*Optimizer or* OptimWrapper) – Wrapped optimizer.
- **milestones** (*list*) – List of epoch indices. Must be increasing.
- **gamma** (*float*) – Multiplicative factor of learning rate decay. Defaults to 0.1.
- **begin** (*int*) – Step at which to start updating the learning rate. Defaults to 0.
- **end** (*int*) – Step at which to stop updating the learning rate. Defaults to INF.

- **last_step** (*int*) – The index of last step. Used for resume without state dict. Defaults to -1.
- **by_epoch** (*bool*) – Whether the scheduled learning rate is updated by epochs. Defaults to True.
- **verbose** (*bool*) – Whether to print the learning rate for each update. Defaults to False.

41.2.15 MultiStepMomentum

class mmengine.optim.**MultiStepMomentum**(*optimizer*, **args*, ***kwargs*)

Decays the specified momentum in each parameter group by gamma once the number of epoch reaches one of the milestones. Notice that such decay can happen simultaneously with other changes to the momentum from outside this scheduler.

Parameters

- **optimizer** (*Optimizer or OptimWrapper*) – optimizer or Wrapped optimizer.
- **milestones** (*list*) – List of epoch indices. Must be increasing.
- **gamma** (*float*) – Multiplicative factor of momentum value decay. Defaults to 0.1.
- **begin** (*int*) – Step at which to start updating the momentum. Defaults to 0.
- **end** (*int*) – Step at which to stop updating the momentum. Defaults to INF.
- **last_step** (*int*) – The index of last step. Used for resume without state dict. Defaults to -1.
- **by_epoch** (*bool*) – Whether the scheduled momentum is updated by epochs. Defaults to True.
- **verbose** (*bool*) – Whether to print the momentum for each update. Defaults to False.

41.2.16 MultiStepParamScheduler

class mmengine.optim.**MultiStepParamScheduler**(*optimizer*, *param_name*, *milestones*, *gamma=0.1*, *last_step=- 1*, *begin=0*, *end=1000000000*, *by_epoch=True*, *verbose=False*)

Decays the specified parameter in each parameter group by gamma once the number of epoch reaches one of the milestones. Notice that such decay can happen simultaneously with other changes to the parameter from outside this scheduler.

Parameters

- **optimizer** (*OptimWrapper or Optimizer*) – Wrapped optimizer.
- **param_name** (*str*) – Name of the parameter to be adjusted, such as `lr`, `momentum`.
- **milestones** (*list*) – List of epoch indices. Must be increasing.
- **gamma** (*float*) – Multiplicative factor of parameter value decay. Defaults to 0.1.
- **begin** (*int*) – Step at which to start updating the parameters. Defaults to 0.
- **end** (*int*) – Step at which to stop updating the parameters. Defaults to INF.
- **last_step** (*int*) – The index of last step. Used for resume without state dict. Defaults to -1.

- **by_epoch** (*bool*) – Whether the scheduled parameters are updated by epochs. Defaults to True.
- **verbose** (*bool*) – Whether to print the value for each update. Defaults to False.

classmethod build_iter_from_epoch(**args*, *milestones*, *begin=0*, *end=1000000000*, *by_epoch=True*, *epoch_length=None*, ***kwargs*)

Build an iter-based instance of this scheduler from an epoch-based config.

41.2.17 OneCycleLR

class mmengine.optim.**OneCycleLR**(*optimizer*, **args*, ***kwargs*)

Sets the learning rate of each parameter group according to the 1cycle learning rate policy. The 1cycle policy anneals the learning rate from an initial learning rate to some maximum learning rate and then from that maximum learning rate to some minimum learning rate much lower than the initial learning rate. This policy was initially described in the paper Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates.

The 1cycle learning rate policy changes the learning rate after every batch. *step* should be called after a batch has been used for training.

This scheduler is not chainable.

Note also that the total number of steps in the cycle can be determined in one of two ways (listed in order of precedence):

1. A value for total_steps is explicitly provided.
2. A number of epochs (epochs) and a number of steps per epoch (steps_per_epoch) are provided. In this case, the number of total steps is inferred by total_steps = epochs * steps_per_epoch

You must either provide a value for total_steps or provide a value for both epochs and steps_per_epoch.

The default behaviour of this scheduler follows the fastai implementation of 1cycle, which claims that "unpublished work has shown even better results by using only two phases". To mimic the behaviour of the original paper instead, set `three_phase=True`.

Parameters

- **optimizer** (*Optimizer*) – Wrapped optimizer.
- **eta_max** (*float or list*) – Upper parameter value boundaries in the cycle for each parameter group.
- **total_steps** (*int*) – The total number of steps in the cycle. Note that if a value is not provided here, then it must be inferred by providing a value for epochs and steps_per_epoch. Default to None.
- **pct_start** (*float*) – The percentage of the cycle (in number of steps) spent increasing the learning rate. Default to 0.3
- **anneal_strategy** (*str*) – {'cos', 'linear'} Specifies the annealing strategy: "cos" for cosine annealing, "linear" for linear annealing. Default to 'cos'
- **div_factor** (*float*) – Determines the initial learning rate via initial_param = eta_max/div_factor Default to 25
- **final_div_factor** (*float*) – Determines the minimum learning rate via eta_min = initial_param/final_div_factor Default to 1e4
- **three_phase** (*bool*) – If `True`, use a third phase of the schedule to annihilate the learning rate according to 'final_div_factor' instead of modifying the second phase (the first two phases will be symmetrical about the step indicated by 'pct_start').

- **last_step** (*int*) – The index of last step. Used for resume without state dict. Defaults to -1.
- **by_epoch** (*bool*) – Whether the scheduled parameters are updated by epochs. Defaults to True.
- **verbose** (*bool*) – Whether to print the value for each update. Defaults to False.

41.2.18 OneCycleParamScheduler

class mmengine.optim.**OneCycleParamScheduler**(*optimizer*, *param_name*, *eta_max=0*, *total_steps=None*, *pct_start=0.3*, *anneal_strategy='cos'*, *div_factor=25.0*, *final_div_factor=10000.0*, *three_phase=False*, *begin=0*, *end=1000000000*, *last_step=- 1*, *by_epoch=True*, *verbose=False*)

Sets the parameters of each parameter group according to the 1cycle learning rate policy. The 1cycle policy anneals the learning rate from an initial learning rate to some maximum learning rate and then from that maximum learning rate to some minimum learning rate much lower than the initial learning rate. This policy was initially described in the paper Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates.

The 1cycle learning rate policy changes the learning rate after every batch. *step* should be called after a batch has been used for training.

This scheduler is not chainable.

Note also that the total number of steps in the cycle can be determined in one of two ways (listed in order of precedence):

1. A value for total_steps is explicitly provided.
2. If total_steps is not defined, begin and end of the ParamSchedul will works for it. In this case, the number of total steps is inferred by total_steps = end - begin

The default behaviour of this scheduler follows the fastai implementation of 1cycle, which claims that "unpublished work has shown even better results by using only two phases". To mimic the behaviour of the original paper instead, set `three_phase=True`.

Parameters

- **optimizer** (*Optimizer*) – Wrapped optimizer.
- **param_name** (*str*) – Name of the parameter to be adjusted, such as `lr`, `momentum`.
- **eta_max** (*float or list*) – Upper parameter value boundaries in the cycle for each parameter group.
- **total_steps** (*int*) – The total number of steps in the cycle. Note that if a value is not provided here, then it will be equal to `end - begin`. Default to None
- **pct_start** (*float*) – The percentage of the cycle (in number of steps) spent increasing the learning rate. Default to 0.3
- **anneal_strategy** (*str*) – {'cos', 'linear'} Specifies the annealing strategy: "cos" for cosine annealing, "linear" for linear annealing. Default to 'cos'
- **div_factor** (*float*) – Determines the initial learning rate via initial_param = eta_max/div_factor Default to 25
- **final_div_factor** (*float*) – Determines the minimum learning rate via eta_min = initial_param/final_div_factor Default to 1e4

- **three_phase** (*bool*) – If `True`, use a third phase of the schedule to annihilate the learning rate according to 'final_div_factor' instead of modifying the second phase (the first two phases will be symmetrical about the step indicated by 'pct_start').
- **last_step** (*int*) – The index of last step. Used for resume without state dict. Defaults to -1.
- **by_epoch** (*bool*) – Whether the scheduled parameters are updated by epochs. Defaults to True.
- **verbose** (*bool*) – Whether to print the value for each update. Defaults to False.
- **begin** (*int*) –
- **end** (*int*) –

classmethod build_iter_from_epoch(**args*, *begin=0*, *end=1000000000*, *total_steps=None*, *by_epoch=True*, *epoch_length=None*, ***kwargs*)

Build an iter-based instance of this scheduler from an epoch-based config.

41.2.19 PolyLR

class mmengine.optim.**PolyLR**(*optimizer*, **args*, ***kwargs*)

Decays the learning rate of each parameter group in a polynomial decay scheme.

Notice that such decay can happen simultaneously with other changes to the parameter value from outside this scheduler.

Parameters

- **optimizer** (*Optimizer or* `OptimWrapper`) – Wrapped optimizer.
- **eta_min** (*float*) – Minimum learning rate at the end of scheduling. Defaults to 0.
- **power** (*float*) – The power of the polynomial. Defaults to 1.0.
- **begin** (*int*) – Step at which to start updating the parameters. Defaults to 0.
- **end** (*int*) – Step at which to stop updating the parameters. Defaults to INF.
- **last_step** (*int*) – The index of last step. Used for resume without state dict. Defaults to -1.
- **by_epoch** (*bool*) – Whether the scheduled parameters are updated by epochs. Defaults to True.
- **verbose** (*bool*) – Whether to print the value for each update. Defaults to False.

41.2.20 PolyMomentum

class mmengine.optim.**PolyMomentum**(*optimizer*, **args*, ***kwargs*)

Decays the momentum of each parameter group in a polynomial decay scheme.

Notice that such decay can happen simultaneously with other changes to the parameter value from outside this scheduler.

Parameters

- **optimizer** (*Optimizer or* `OptimWrapper`) – optimizer or Wrapped optimizer.
- **eta_min** (*float*) – Minimum momentum at the end of scheduling. Defaults to 0.

- **power** (*float*) – The power of the polynomial. Defaults to 1.0.
- **begin** (*int*) – Step at which to start updating the parameters. Defaults to 0.
- **end** (*int*) – Step at which to stop updating the parameters. Defaults to INF.
- **last_step** (*int*) – The index of last step. Used for resume without state dict. Defaults to -1.
- **by_epoch** (*bool*) – Whether the scheduled parameters are updated by epochs. Defaults to True.
- **verbose** (*bool*) – Whether to print the value for each update. Defaults to False.

41.2.21 PolyParamScheduler

class mmengine.optim.**PolyParamScheduler**(*optimizer*, *param_name*, *eta_min=0*, *power=1.0*, *begin=0*, *end=1000000000*, *last_step=- 1*, *by_epoch=True*, *verbose=False*)

Decays the parameter value of each parameter group in a polynomial decay scheme.

Notice that such decay can happen simultaneously with other changes to the parameter value from outside this scheduler.

Parameters

- **optimizer** (*Optimizer or* OptimWrapper) – optimizer or Wrapped optimizer.
- **param_name** (*str*) – Name of the parameter to be adjusted, such as `lr`, `momentum`.
- **eta_min** (*float*) – Minimum parameter value at the end of scheduling. Defaults to 0.
- **power** (*float*) – The power of the polynomial. Defaults to 1.0.
- **begin** (*int*) – Step at which to start updating the parameters. Defaults to 0.
- **end** (*int*) – Step at which to stop updating the parameters. Defaults to INF.
- **last_step** (*int*) – The index of last step. Used for resume without state dict. Defaults to -1.
- **by_epoch** (*bool*) – Whether the scheduled parameters are updated by epochs. Defaults to True.
- **verbose** (*bool*) – Whether to print the value for each update. Defaults to False.

classmethod **build_iter_from_epoch**(**args*, *begin=0*, *end=1000000000*, *by_epoch=True*, *epoch_length=None*, ***kwargs*)

Build an iter-based instance of this scheduler from an epoch-based config.

41.2.22 StepLR

class mmengine.optim.**StepLR**(*optimizer*, **args*, ***kwargs*)

Decays the learning rate of each parameter group by gamma every step_size epochs. Notice that such decay can happen simultaneously with other changes to the learning rate from outside this scheduler.

Parameters

- **optimizer** (*Optimizer or* OptimWrapper) – Wrapped optimizer.
- **step_size** (*int*) – Period of learning rate decay.
- **gamma** (*float*) – Multiplicative factor of learning rate decay. Defaults to 0.1.

- **begin** (*int*) – Step at which to start updating the learning rate. Defaults to 0.
- **end** (*int*) – Step at which to stop updating the learning rate. Defaults to INF.
- **last_step** (*int*) – The index of last step. Used for resume without state dict. Defaults to -1.
- **by_epoch** (*bool*) – Whether the scheduled learning rate is updated by epochs. Defaults to True.
- **verbose** (*bool*) – Whether to print the learning rate for each update. Defaults to False.

41.2.23 StepMomentum

class mmengine.optim.**StepMomentum**(*optimizer*, **args*, ***kwargs*)

Decays the momentum of each parameter group by gamma every step_size epochs. Notice that such decay can happen simultaneously with other changes to the momentum from outside this scheduler.

Parameters

- **optimizer** (*Optimizer or* OptimWrapper) – optimizer or Wrapped optimizer.
- **step_size** (*int*) – Period of momentum value decay.
- **gamma** (*float*) – Multiplicative factor of momentum value decay. Defaults to 0.1.
- **begin** (*int*) – Step at which to start updating the momentum. Defaults to 0.
- **end** (*int*) – Step at which to stop updating the momentum. Defaults to INF.
- **last_step** (*int*) – The index of last step. Used for resume without state dict. Defaults to -1.
- **by_epoch** (*bool*) – Whether the scheduled momentum is updated by epochs. Defaults to True.
- **verbose** (*bool*) – Whether to print the momentum for each update. Defaults to False.

41.2.24 StepParamScheduler

class mmengine.optim.**StepParamScheduler**(*optimizer*, *param_name*, *step_size*, *gamma=0.1*, *begin=0*, *end=1000000000*, *last_step=- 1*, *by_epoch=True*, *verbose=False*)

Decays the parameter value of each parameter group by gamma every step_size epochs. Notice that such decay can happen simultaneously with other changes to the parameter value from outside this scheduler.

Parameters

- **optimizer** (OptimWrapper *or Optimizer*) – Wrapped optimizer.
- **param_name** (*str*) – Name of the parameter to be adjusted, such as `lr`, `momentum`.
- **step_size** (*int*) – Period of parameter value decay.
- **gamma** (*float*) – Multiplicative factor of parameter value decay. Defaults to 0.1.
- **begin** (*int*) – Step at which to start updating the parameters. Defaults to 0.
- **end** (*int*) – Step at which to stop updating the parameters. Defaults to INF.
- **last_step** (*int*) – The index of last step. Used for resume without state dict. Defaults to -1.

- **by_epoch** (*bool*) – Whether the scheduled parameters are updated by epochs. Defaults to True.
- **verbose** (*bool*) – Whether to print the value for each update. Defaults to False.

classmethod build_iter_from_epoch(**args*, *step_size*, *begin=0*, *end=1000000000*, *by_epoch=True*, *epoch_length=None*, ***kwargs*)

Build an iter-based instance of this scheduler from an epoch-based config.

CHAPTER

FORTYTWO

MMENGINE.EVALUATOR

mmengine.evaluator

- *Evaluator*
- *Metric*
- *Utils*

42.1 Evaluator

Evaluator	Wrapper class to compose multiple *BaseMetric* instances.

42.1.1 Evaluator

class mmengine.evaluator.**Evaluator**(*metrics*)

Wrapper class to compose multiple *BaseMetric* instances.

Parameters **metrics** (*dict or BaseMetric or Sequence*) – The config of metrics.

property dataset_meta: Optional[dict]

Meta info of the dataset.

Type Optional[dict]

evaluate(*size*)

Invoke `evaluate` method of each metric and collect the metrics dictionary.

Parameters **size** (*int*) – Length of the entire validation dataset. When batch size > 1, the dataloader may pad some data samples to make sure all ranks have the same length of dataset slice. The `collect_results` function will drop the padded data based on this size.

Returns Evaluation results of all metrics. The keys are the names of the metrics, and the values are corresponding results.

Return type dict

offline_evaluate(*data_samples*, *data=None*, *chunk_size=1*)

Offline evaluate the dumped predictions on the given data .

Parameters

- **data_samples** (*Sequence*) – All predictions and ground truth of the model and the validation set.
- **data** (*Sequence, optional*) – All data of the validation set.
- **chunk_size** (*int*) – The number of data samples and predictions to be processed in a batch.

process(*data_samples*, *data_batch=None*)
Convert `BaseDataSample` to dict and invoke process method of each metric.

Parameters

- **data_samples** (*Sequence[BaseDataElement]*) – predictions of the model, and the ground truth of the validation set.
- **data_batch** (*Any, optional*) – A batch of data from the dataloader.

42.2 Metric

BaseMetric	Base class for a metric.
DumpResults	Dump model predictions to a pickle file for offline evaluation.

42.2.1 BaseMetric

class mmengine.evaluator.**BaseMetric**(*collect_device='cpu'*, *prefix=None*)
Base class for a metric.

The metric first processes each batch of data_samples and predictions, and appends the processed results to the results list. Then it collects all results together from all ranks if distributed training is used. Finally, it computes the metrics of the entire dataset.

A subclass of class:*BaseMetric* should assign a meaningful value to the class attribute *default_prefix*. See the argument *prefix* for details.

Parameters

- **collect_device** (*str*) – Device name used for collecting results from different ranks during distributed training. Must be 'cpu' or 'gpu'. Defaults to 'cpu'.
- **prefix** (*str, optional*) – The prefix that will be added in the metric names to disambiguate homonymous metrics of different evaluators. If prefix is not provided in the argument, self.default_prefix will be used instead. Default: None

Return type None

abstract compute_metrics(*results*)
Compute the metrics from processed results.

Parameters **results** (*list*) – The processed results of each batch.

Returns The computed metrics. The keys are the names of the metrics, and the values are corresponding results.

Return type dict

property dataset_meta: Optional[dict]
Meta info of the dataset.

Type Optional[dict]

evaluate(*size*)
Evaluate the model performance of the whole dataset after processing all batches.

Parameters **size** (*int*) – Length of the entire validation dataset. When batch size > 1, the dataloader may pad some data samples to make sure all ranks have the same length of dataset slice. The `collect_results` function will drop the padded data based on this size.

Returns Evaluation metrics dict on the val dataset. The keys are the names of the metrics, and the values are corresponding results.

Return type dict

abstract process(*data_batch*, *data_samples*)
Process one batch of data samples and predictions. The processed results should be stored in `self.results`, which will be used to compute the metrics when all batches have been processed.

Parameters

- **data_batch** (*Any*) – A batch of data from the dataloader.
- **data_samples** (*Sequence[dict]*) – A batch of outputs from the model.

Return type None

42.2.2 DumpResults

class mmengine.evaluator.**DumpResults**(*out_file_path*, *collect_device='cpu'*)
Dump model predictions to a pickle file for offline evaluation.

Parameters

- **out_file_path** (*str*) – Path of the dumped file. Must end with '.pkl' or '.pickle'.
- **collect_device** (*str*) – Device name used for collecting results from different ranks during distributed training. Must be 'cpu' or 'gpu'. Defaults to 'cpu'.

Return type None

compute_metrics(*results*)
dump the prediction results to a pickle file.

Parameters **results** (*list*) –

Return type dict

process(*data_batch*, *predictions*)
transfer tensors in predictions to CPU.

Parameters

- **data_batch** (*Any*) –
- **predictions** (*Sequence[dict]*) –

Return type None

42.3 Utils

get_metric_value	Get the metric value specified by an indicator, which can be either a metric name or a full name with evaluator prefix.

42.3.1 mmengine.evaluator.get_metric_value

mmengine.evaluator.**get_metric_value**(*indicator*, *metrics*)

Get the metric value specified by an indicator, which can be either a metric name or a full name with evaluator prefix.

Parameters

- **indicator** (*str*) – The metric indicator, which can be the metric name (e.g. 'AP') or the full name with prefix (e.g. 'COCO/AP')
- **metrics** (*dict*) – The evaluation results output by the evaluator

Returns The specified metric value

Return type Any

CHAPTER

FORTYTHREE

MMENGINE.STRUCTURES

BaseDataElement	A base data interface that supports Tensor-like and dict-like operations.
InstanceData	Data structure for instance-level annotations or predictions.
LabelData	Data structure for label-level annotations or predictions.
PixelData	Data structure for pixel-level annotations or predictions.

43.1 BaseDataElement

class mmengine.structures.**BaseDataElement**(*, *metainfo=None*, ***kwargs*)

A base data interface that supports Tensor-like and dict-like operations.

A typical data elements refer to predicted results or ground truth labels on a task, such as predicted bboxes, instance masks, semantic segmentation masks, etc. Because groundtruth labels and predicted results often have similar properties (for example, the predicted bboxes and the groundtruth bboxes), MMEngine uses the same abstract data interface to encapsulate predicted results and groundtruth labels, and it is recommended to use different name conventions to distinguish them, such as using `gt_instances` and `pred_instances` to distinguish between labels and predicted results. Additionally, we distinguish data elements at instance level, pixel level, and label level. Each of these types has its own characteristics. Therefore, MMEngine defines the base class `BaseDataElement`, and implement `InstanceData`, `PixelData`, and `LabelData` inheriting from `BaseDataElement` to represent different types of ground truth labels or predictions.

Another common data element is sample data. A sample data consists of input data (such as an image) and its annotations and predictions. In general, an image can have multiple types of annotations and/or predictions at the same time (for example, both pixel-level semantic segmentation annotations and instance-level detection bboxes annotations). All labels and predictions of a training sample are often passed between Dataset, Model, Visualizer, and Evaluator components. In order to simplify the interface between components, we can treat them as a large data element and encapsulate them. Such data elements are generally called XXDataSample in the OpenMMLab. Therefore, Similar to *nn.Module*, the *BaseDataElement* allows *BaseDataElement* as its attribute. Such a class generally encapsulates all the data of a sample in the algorithm library, and its attributes generally are various types of data elements. For example, MMDetection is assigned by the BaseDataElement to encapsulate all the data elements of the sample labeling and prediction of a sample in the algorithm library.

The attributes in `BaseDataElement` are divided into two parts, the `metainfo` and the `data` respectively.

- `metainfo`: Usually contains the information about the image such as filename, image_shape, pad_shape, etc. The attributes can be accessed or modified by dict-like or object-like operations, such as .``(for data access and modification) , ``in, `del`, `pop(str)`, `get(str)`, `metainfo_keys()`, `metainfo_values()`, `metainfo_items()`, ``set_metainfo()``(for set or change key-value pairs in metainfo).

- `data`: Annotations or model predictions are stored. The attributes can be accessed or modified by dict-like or object-like operations, such as `.`, `in`, `del`, `pop(str)` `get(str)`, `keys()`, `values()`, `items()`. Users can also apply tensor-like methods to all obj:`torch.Tensor` in the `data_fileds`, such as `.cuda()`, `.cpu()`, `.numpy()`, , `.to()` `to_tensor()`, `.detach()`.

  Parameters

  - **metainfo** (*dict, optional*) – A dict contains the meta information of single image. such as `dict(img_shape=(512, 512, 3), scale_factor=(1, 1, 1, 1))`. Defaults to None.
  - **kwargs** (*dict, optional*) – A dict contains annotations of single image or model predictions. Defaults to None.

  Return type None

Examples

```
>>> from mmengine.structures import BaseDataElement
>>> gt_instances = BaseDataElement()
>>> bboxes = torch.rand((5, 4))
>>> scores = torch.rand((5,))
>>> img_id = 0
>>> img_shape = (800, 1333)
>>> gt_instances = BaseDataElement(
...     metainfo=dict(img_id=img_id, img_shape=img_shape),
...     bboxes=bboxes, scores=scores)
>>> gt_instances = BaseDataElement(
...                     metainfo=dict(img_id=img_id,
...                                   img_shape=(H, W)))
```

```
>>> # new
>>> gt_instances1 = gt_instance.new(
...                     metainfo=dict(img_id=1, img_shape=(640, 640)),
...                     bboxes=torch.rand((5, 4)),
...                     scores=torch.rand((5,)))
>>> gt_instances2 = gt_instances1.new()
```

```
>>> # add and process property
>>> gt_instances = BaseDataElement()
>>> gt_instances.set_metainfo(dict(img_id=9, img_shape=(100, 100))
>>> assert 'img_shape' in gt_instances.metainfo_keys()
>>> assert 'img_shape' in gt_instances
>>> assert 'img_shape' not in gt_instances.keys()
>>> assert 'img_shape' in gt_instances.all_keys()
>>> print(gt_instances.img_shape)
>>> gt_instances.scores = torch.rand((5,))
>>> assert 'scores' in gt_instances.keys()
>>> assert 'scores' in gt_instances
>>> assert 'scores' in gt_instances.all_keys()
>>> assert 'scores' not in gt_instances.metainfo_keys()
>>> print(gt_instances.scores)
>>> gt_instances.bboxes = torch.rand((5, 4))
```

(continues on next page)

(continued from previous page)

```
>>> assert 'bboxes' in gt_instances.keys()
>>> assert 'bboxes' in gt_instances
>>> assert 'bboxes' in gt_instances.all_keys()
>>> assert 'bboxes' not in gt_instances.metainfo_keys()
>>> print(gt_instances.bboxes)
```

```
>>> # delete and change property
>>> gt_instances = BaseDataElement(
...  metainfo=dict(img_id=0, img_shape=(640, 640)),
...  bboxes=torch.rand((6, 4)), scores=torch.rand((6,)))
>>> gt_instances.img_shape = (1280, 1280)
>>> gt_instances.img_shape  # (1280, 1280)
>>> gt_instances.bboxes = gt_instances.bboxes * 2
>>> gt_instances.get('img_shape', None)  # (640, 640)
>>> gt_instances.get('bboxes', None)    # 6x4 tensor
>>> del gt_instances.img_shape
>>> del gt_instances.bboxes
>>> assert 'img_shape' not in gt_instances
>>> assert 'bboxes' not in gt_instances
>>> gt_instances.pop('img_shape', None)  # None
>>> gt_instances.pop('bboxes', None)  # None
```

```
>>> # Tensor-like
>>> cuda_instances = gt_instances.cuda()
>>> cuda_instances = gt_instancess.to('cuda:0')
>>> cpu_instances = cuda_instances.cpu()
>>> cpu_instances = cuda_instances.to('cpu')
>>> fp16_instances = cuda_instances.to(
...  device=None, dtype=torch.float16, non_blocking=False, copy=False,
...  memory_format=torch.preserve_format)
>>> cpu_instances = cuda_instances.detach()
>>> np_instances = cpu_instances.numpy()
```

```
>>> # print
>>> metainfo = dict(img_shape=(800, 1196, 3))
>>> gt_instances = BaseDataElement(
>>>     metainfo=metainfo, det_labels=torch.LongTensor([0, 1, 2, 3]))
>>> sample = BaseDataElement(metainfo=metainfo,
...                          gt_instances=gt_instances)
>>> print(sample)
<BaseDataElement(
    META INFORMATION
    img_shape: (800, 1196, 3)
    DATA FIELDS
    gt_instances: <BaseDataElement(
            META INFORMATION
            img_shape: (800, 1196, 3)
            DATA FIELDS
            det_labels: tensor([0, 1, 2, 3])
        ) at 0x7f0ec5eadc70>
) at 0x7f0fea49e130>
```

```
>>> # inheritance
>>> class DetDataSample(BaseDataElement):
...     @property
...     def proposals(self):
...         return self._proposals
...     @proposals.setter
...     def proposals(self, value):
...         self.set_field(value, '_proposals', dtype=BaseDataElement)
...     @proposals.deleter
...     def proposals(self):
...         del self._proposals
...     @property
...     def gt_instances(self):
...         return self._gt_instances
...     @gt_instances.setter
...     def gt_instances(self, value):
...         self.set_field(value, '_gt_instances',
...                        dtype=BaseDataElement)
...     @gt_instances.deleter
...     def gt_instances(self):
...         del self._gt_instances
...     @property
...     def pred_instances(self):
...         return self._pred_instances
...     @pred_instances.setter
...     def pred_instances(self, value):
...         self.set_field(value,'_pred_instances',
...                        dtype=BaseDataElement)
...     @pred_instances.deleter
...     def pred_instances(self):
...         del self._pred_instances
>>> det_sample = DetDataSample()
>>> proposals = BaseDataElement(bboxes=torch.rand((5, 4)))
>>> det_sample.proposals = proposals
>>> assert 'proposals' in det_sample
>>> assert det_sample.proposals == proposals
>>> del det_sample.proposals
>>> assert 'proposals' not in det_sample
>>> with self.assertRaises(AssertionError):
...     det_sample.proposals = torch.rand((5, 4))
```

all_items()

Returns an iterator object whose element is (key, value) tuple pairs for `metainfo` and `data`.

Return type iterator

all_keys()

Returns Contains all keys in metainfo and data.

Return type list

all_values()

Returns Contains all values in metainfo and data.

Return type list

clone()
Deep copy the current data element.

Returns the copy of current data element.

Return type *BaseDataElement*

cpu()
Convert all tensors to CPU in data.

Return type *mmengine.structures.base_data_element.BaseDataElement*

cuda()
Convert all tensors to GPU in data.

Return type *mmengine.structures.base_data_element.BaseDataElement*

detach()
Detach all tensors in data.

Return type *mmengine.structures.base_data_element.BaseDataElement*

get(*key*, *default=None*)
get property in data and metainfo as the same as python.

Return type Any

items()

Returns an iterator object whose element is (key, value) tuple pairs for `data`.

Return type iterator

keys()

Returns Contains all keys in data_fields.

Return type list

property metainfo: dict
A dict contains metainfo of current data element.

Type dict

metainfo_items()

Returns an iterator object whose element is (key, value) tuple pairs for `metainfo`.

Return type iterator

metainfo_keys()

Returns Contains all keys in metainfo_fields.

Return type list

metainfo_values()

Returns Contains all values in metainfo.

Return type list

new(*, *metainfo=None*, ***kwargs*)
Return a new data element with same type. If `metainfo` and `data` are None, the new data element will have same metainfo and data. If metainfo or data is not None, the new result will overwrite it with the input value.

Parameters

- **metainfo** (*dict, optional*) – A dict contains the meta information of image, such as `img_shape`, `scale_factor`, etc. Defaults to None.
- **kwargs** (*dict*) – A dict contains annotations of image or model predictions.

Returns a new data element with same type.

Return type *BaseDataElement*

numpy()
Convert all tensor to np.narray in data.

Return type *mmengine.structures.base_data_element.BaseDataElement*

pop(**args*)
pop property in data and metainfo as the same as python.

Return type Any

set_data(*data*)
Set or change key-value pairs in `data_field` by parameter `data`.

Parameters **data** (*dict*) – A dict contains annotations of image or model predictions.

Return type None

set_field(*value*, *name*, *dtype=None*, *field_type='data'*)
Special method for set union field, used as property.setter functions.

Parameters

- **value** (*Any*) –
- **name** (*str*) –
- **dtype** (*Optional[Union[Type, Tuple[Type, ...]]]*) –
- **field_type** (*str*) –

Return type None

set_metainfo(*metainfo*)
Set or change key-value pairs in `metainfo_field` by parameter `metainfo`.

Parameters **metainfo** (*dict*) – A dict contains the meta information of image, such as `img_shape`, `scale_factor`, etc.

Return type None

to(**args*, ***kwargs*)
Apply same name function to all tensors in data_fields.

Return type *mmengine.structures.base_data_element.BaseDataElement*

to_dict()
Convert BaseDataElement to dict.

Return type dict

to_tensor()
Convert all np.narray to tensor in data.

Return type *mmengine.structures.base_data_element.BaseDataElement*

update(*instance*)
The update() method updates the BaseDataElement with the elements from another BaseDataElement object.

Parameters

- **instance** (BaseDataElement) – Another BaseDataElement object for
- **the current object.** (*update*) –

Return type None

values()

Returns Contains all values in data.

Return type list

43.2 InstanceData

class mmengine.structures.**InstanceData**(*, *metainfo=None*, ***kwargs*)
Data structure for instance-level annotations or predictions.

Subclass of *BaseDataElement*. All value in *data_fields* should have the same length. This design refer to https://github.com/facebookresearch/detectron2/blob/master/detectron2/structures/instances.py # noqa E501 InstanceData also support extra functions: `index`, `slice` and `cat` for data field. The type of value in data field can be base data structure such as *torch.tensor*, *numpy.ndarray*, *list*, *str*, *tuple*, and can be customized data structure that has `__len__`, `__getitem__` and `cat` attributes.

Examples

```
>>> # custom data structure
>>> class TmpObject:
...     def __init__(self, tmp) -> None:
...         assert isinstance(tmp, list)
...         self.tmp = tmp
...     def __len__(self):
...         return len(self.tmp)
...     def __getitem__(self, item):
...         if type(item) == int:
...             if item >= len(self) or item < -len(self):  # type:ignore
...                 raise IndexError(f'Index {item} out of range!')
...             else:
...                 # keep the dimension
...                 item = slice(item, None, len(self))
```

(continues on next page)

(continued from previous page)

```
...         return TmpObject(self.tmp[item])
...     @staticmethod
...     def cat(tmp_objs):
...         assert all(isinstance(results, TmpObject) for results in tmp_objs)
...         if len(tmp_objs) == 1:
...             return tmp_objs[0]
...         tmp_list = [tmp_obj.tmp for tmp_obj in tmp_objs]
...         tmp_list = list(itertools.chain(*tmp_list))
...         new_data = TmpObject(tmp_list)
...         return new_data
...     def __repr__(self):
...         return str(self.tmp)
>>> from mmengine.structures import InstanceData
>>> import numpy as np
>>> img_meta = dict(img_shape=(800, 1196, 3), pad_shape=(800, 1216, 3))
>>> instance_data = InstanceData(metainfo=img_meta)
>>> 'img_shape' in instance_data
True
>>> instance_data.det_labels = torch.LongTensor([2, 3])
>>> instance_data["det_scores"] = torch.Tensor([0.8, 0.7])
>>> instance_data.bboxes = torch.rand((2, 4))
>>> instance_data.polygons = TmpObject([[1, 2, 3, 4], [5, 6, 7, 8]])
>>> len(instance_data)
2
>>> print(instance_data)
<InstanceData(
    META INFORMATION
    pad_shape: (800, 1196, 3)
    img_shape: (800, 1216, 3)
    DATA FIELDS
    det_labels: tensor([2, 3])
    det_scores: tensor([0.8, 0.7000])
    bboxes: tensor([[0.4997, 0.7707, 0.0595, 0.4188],
        [0.8101, 0.3105, 0.5123, 0.6263]])
    polygons: [[1, 2, 3, 4], [5, 6, 7, 8]]
    ) at 0x7fb492de6280>
>>> sorted_results = instance_data[instance_data.det_scores.sort().indices]
>>> sorted_results.det_scores
tensor([0.7000, 0.8000])
>>> print(instance_data[instance_data.det_scores > 0.75])
<InstanceData(
    META INFORMATION
    pad_shape: (800, 1216, 3)
    img_shape: (800, 1196, 3)
    DATA FIELDS
    det_labels: tensor([2])
    masks: [[11, 21, 31, 41]]
    det_scores: tensor([0.8000])
    bboxes: tensor([[0.9308, 0.4000, 0.6077, 0.5554]])
    polygons: [[1, 2, 3, 4]]
) at 0x7f64ecf0ec40>
>>> print(instance_data[instance_data.det_scores > 1])
```

(continues on next page)

(continued from previous page)

```
<InstanceData(
    META INFORMATION
    pad_shape: (800, 1216, 3)
    img_shape: (800, 1196, 3)
    DATA FIELDS
    det_labels: tensor([], dtype=torch.int64)
    masks: []
    det_scores: tensor([])
    bboxes: tensor([], size=(0, 4))
    polygons: [[]]
) at 0x7f660a6a7f70>
>>> print(instance_data.cat([instance_data, instance_data]))
<InstanceData(
    META INFORMATION
    img_shape: (800, 1196, 3)
    pad_shape: (800, 1216, 3)
    DATA FIELDS
    det_labels: tensor([2, 3, 2, 3])
    bboxes: tensor([[0.7404, 0.6332, 0.1684, 0.9961],
                [0.2837, 0.8112, 0.5416, 0.2810],
                [0.7404, 0.6332, 0.1684, 0.9961],
                [0.2837, 0.8112, 0.5416, 0.2810]])
    data:
    polygons: [[1, 2, 3, 4], [5, 6, 7, 8],
               [1, 2, 3, 4], [5, 6, 7, 8]]
    det_scores: tensor([0.8000, 0.7000, 0.8000, 0.7000])
    masks: [[11, 21, 31, 41], [51, 61, 71, 81],
            [11, 21, 31, 41], [51, 61, 71, 81]]
) at 0x7f203542feb0>
```

Parameters **metainfo** (*Optional[dict]*) –

Return type None

static cat(*instances_list*)

Concat the instances of all *InstanceData* in the list.

Note: To ensure that cat returns as expected, make sure that all elements in the list must have exactly the same keys.

Parameters **instances_list** (list[*InstanceData*]) – A list of *InstanceData*.

Returns *InstanceData*

Return type obj

43.3 LabelData

class mmengine.structures.**LabelData**(*, *metainfo=None*, ***kwargs*)

Data structure for label-level annotations or predictions.

Parameters **metainfo** (*Optional[dict]*) –

Return type None

static **label_to_onehot**(*label*, *num_classes*)

Convert the label-format input to one-hot.

Parameters

- **label** (*torch.Tensor*) – The label-format input. The format of item must be label-format.
- **num_classes** (*int*) – The number of classes.

Returns The converted results.

Return type torch.Tensor

static **onehot_to_label**(*onehot*)

Convert the one-hot input to label.

Parameters **onehot** (*torch.Tensor, optional*) – The one-hot input. The format of input must be one-hot.

Returns The converted results.

Return type torch.Tensor

43.4 PixelData

class mmengine.structures.**PixelData**(*, *metainfo=None*, ***kwargs*)

Data structure for pixel-level annotations or predictions.

All data items in `data_fields` of `PixelData` meet the following requirements:

- They all have 3 dimensions in orders of channel, height, and width.
- They should have the same height and width.

Examples

```
>>> metainfo = dict(
...     img_id=random.randint(0, 100),
...     img_shape=(random.randint(400, 600), random.randint(400, 600)))
>>> image = np.random.randint(0, 255, (4, 20, 40))
>>> featmap = torch.randint(0, 255, (10, 20, 40))
>>> pixel_data = PixelData(metainfo=metainfo,
...                        image=image,
...                        featmap=featmap)
>>> print(pixel_data)
>>> (20, 40)
```

```
>>> # slice
>>> slice_data = pixel_data[10:20, 20:40]
>>> assert slice_data.shape == (10, 10)
>>> slice_data = pixel_data[10, 20]
>>> assert slice_data.shape == (1, 1)
```

```
>>> # set
>>> pixel_data.map3 = torch.randint(0, 255, (20, 40))
>>> assert tuple(pixel_data.map3.shape) == (1, 20, 40)
>>> with self.assertRaises(AssertionError):
...     # The dimension must be 3 or 2
...     pixel_data.map2 = torch.randint(0, 255, (1, 3, 20, 40))
```

Parameters **metainfo** (*Optional[dict]*) –

Return type None

property shape

The shape of pixel data.

CHAPTER

FORTYFOUR

MMENGINE.DATASET

mmengine.dataset

- *Dataset*
- *Dataset Wrapper*
- *Sampler*
- *Utils*

44.1 Dataset

BaseDataset	BaseDataset for open source projects in OpenMMLab.
Compose	Compose multiple transforms sequentially.

44.1.1 BaseDataset

class mmengine.dataset.**BaseDataset**(*ann_file=''*, *metainfo=None*, *data_root=''*, *data_prefix={'img_path': ''}*, *filter_cfg=None*, *indices=None*, *serialize_data=True*, *pipeline=[]*, *test_mode=False*, *lazy_init=False*, *max_refetch=1000*)

BaseDataset for open source projects in OpenMMLab.

The annotation format is shown as follows.

```
{
    "metainfo":
    {
      "dataset_type": "test_dataset",
      "task_name": "test_task"
    },
    "data_list":
    [
      {
        "img_path": "test_img.jpg",
        "height": 604,
        "width": 640,
        "instances":
```

(continues on next page)

(continued from previous page)

```
        [
          {
            "bbox": [0, 0, 10, 20],
            "bbox_label": 1,
            "mask": [[0,0],[0,10],[10,20],[20,0]],
            "extra_anns": [1,2,3]
          },
          {
            "bbox": [10, 10, 110, 120],
            "bbox_label": 2,
            "mask": [[10,10],[10,110],[110,120],[120,10]],
            "extra_anns": [4,5,6]
          }
        ]
      },
    ]
}
```

Parameters

- **ann_file** (*str*) – Annotation file path. Defaults to ‘’.
- **metainfo** (*dict, optional*) – Meta information for dataset, such as class information. Defaults to None.
- **data_root** (*str*) – The root directory for `data_prefix` and `ann_file`. Defaults to ‘’.
- **data_prefix** (*dict*) – Prefix for training data. Defaults to dict(img_path=”).
- **filter_cfg** (*dict, optional*) – Config for filter data. Defaults to None.
- **indices** (*int or Sequence[int], optional*) – Support using first few data in annotation file to facilitate training/testing on a smaller dataset. Defaults to None which means using all `data_infos`.
- **serialize_data** (*bool, optional*) – Whether to hold memory using serialized objects, when enabled, data loader workers can use shared RAM from master process instead of making a copy. Defaults to True.
- **pipeline** (*list, optional*) – Processing pipeline. Defaults to [].
- **test_mode** (*bool, optional*) – `test_mode=True` means in test phase. Defaults to False.
- **lazy_init** (*bool, optional*) – Whether to load annotation during instantiation. In some cases, such as visualization, only the meta information of the dataset is needed, which is not necessary to load annotation file. `Basedataset` can skip load annotations to save time by set `lazy_init=False`. Defaults to False.
- **max_refetch** (*int, optional*) – If `Basedataset.prepare_data` get a None img. The maximum extra number of cycles to get a valid image. Defaults to 1000.

Note: BaseDataset collects meta information from `annotation file` (the lowest priority), `BaseDataset.METAINFO``(medium) and ``metainfo parameter` (highest) passed to constructors. The lower priority meta information will be overwritten by higher one.

Note: Dataset wrapper such as `ConcatDataset`, `RepeatDataset` .etc. should not inherit from `BaseDataset` since `get_subset` and `get_subset_` could produce ambiguous meaning sub-dataset which conflicts with original dataset.

Examples

```
>>> # Assume the annotation file is given above.
>>> class CustomDataset(BaseDataset):
>>>     METAINFO: dict = dict(task_name='custom_task',
>>>                           dataset_type='custom_type')
>>> metainfo=dict(task_name='custom_task_name')
>>> custom_dataset = CustomDataset(
>>>                       'path/to/ann_file',
>>>                       metainfo=metainfo)
>>> # meta information of annotation file will be overwritten by
>>> # `CustomDataset.METAINFO`. The merged meta information will
>>> # further be overwritten by argument `metainfo`.
>>> custom_dataset.metainfo
{'task_name': custom_task_name, dataset_type: custom_type}
```

filter_data()

Filter annotations according to filter_cfg. Defaults return all `data_list`.

If some `data_list` could be filtered according to specific logic, the subclass should override this method.

Returns Filtered results.

Return type list[int]

full_init()

Load annotation file and set `BaseDataset._fully_initialized` to True.

If `lazy_init=False`, `full_init` will be called during the instantiation and `self._fully_initialized` will be set to True. If `obj._fully_initialized=False`, the class method decorated by `force_full_init` will call `full_init` automatically.

Several steps to initialize annotation:

- load_data_list: Load annotations from annotation file.
- filter data information: Filter annotations according to filter_cfg.
- slice_data: Slice dataset according to `self._indices`
- serialize_data: Serialize `self.data_list` if

`self.serialize_data` is True.

get_cat_ids(*idx*)

Get category ids by index. Dataset wrapped by ClassBalancedDataset must implement this method.

The `ClassBalancedDataset` requires a subclass which implements this method.

Parameters **idx** (*int*) – The index of data.

Returns All categories in the image of specified index.

Return type list[int]

get_data_info(*idx*)

Get annotation by index and automatically call `full_init` if the dataset has not been fully initialized.

Parameters **idx** (*int*) – The index of data.

Returns The idx-th annotation of the dataset.

Return type dict

get_subset(*indices*)

Return a subset of dataset.

This method will return a subset of original dataset. If type of indices is int, `get_subset_` will return a subdataset which contains the first or last few data information according to indices is positive or negative. If type of indices is a sequence of int, the subdataset will extract the information according to the index given in indices.

Examples

```
>>> dataset = BaseDataset('path/to/ann_file')
>>> len(dataset)
100
>>> subdataset = dataset.get_subset(90)
>>> len(sub_dataset)
90
>>> # if type of indices is list, extract the corresponding
>>> # index data information
>>> subdataset = dataset.get_subset([0, 1, 2, 3, 4, 5, 6, 7,
>>>                                  8, 9])
>>> len(sub_dataset)
10
>>> subdataset = dataset.get_subset(-3)
>>> len(subdataset) # Get the latest few data information.
3
```

Parameters **indices** (*int or Sequence[int]*) – If type of indices is int, indices represents the first or last few data of dataset according to indices is positive or negative. If type of indices is Sequence, indices represents the target data information index of dataset.

Returns A subset of dataset.

Return type *BaseDataset*

get_subset_(*indices*)

The in-place version of ``get_subset `` to convert dataset to a subset of original dataset.

This method will convert the original dataset to a subset of dataset. If type of indices is int, `get_subset_` will return a subdataset which contains the first or last few data information according to indices is positive or negative. If type of indices is a sequence of int, the subdataset will extract the data information according to the index given in indices.

Examples

```
>>> dataset = BaseDataset('path/to/ann_file')
>>> len(dataset)
100
>>> dataset.get_subset_(90)
>>> len(dataset)
90
>>> # if type of indices is sequence, extract the corresponding
>>> # index data information
>>> dataset.get_subset_([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> len(dataset)
10
>>> dataset.get_subset_(-3)
>>> len(dataset) # Get the latest few data information.
3
```

Parameters **indices** (*int or Sequence[int]*) – If type of indices is int, indices represents the first or last few data of dataset according to indices is positive or negative. If type of indices is Sequence, indices represents the target data information index of dataset.

Return type None

load_data_list()

Load annotations from an annotation file named as `self.ann_file`

If the annotation file does not follow OpenMMLab 2.0 format dataset . The subclass must override this method for load annotations. The meta information of annotation file will be overwritten `METAINFO` and `metainfo` argument of constructor.

Returns A list of annotation.

Return type list[dict]

property metainfo: dict

Get meta information of dataset.

Returns meta information collected from `BaseDataset.METAINFO`, annotation file and metainfo argument during instantiation.

Return type dict

parse_data_info(*raw_data_info*)

Parse raw annotation to target format.

This method should return dict or list of dict. Each dict or list contains the data information of a training sample. If the protocol of the sample annotations is changed, this function can be overridden to update the parsing logic while keeping compatibility.

Parameters **raw_data_info** (*dict*) – Raw data information load from `ann_file`

Returns Parsed annotation.

Return type list or list[dict]

prepare_data(*idx*)

Get data processed by `self.pipeline`.

Parameters **idx** (*int*) – The index of `data_info`.

Returns Depends on `self.pipeline`.

Return type Any

44.1.2 Compose

class mmengine.dataset.**Compose**(*transforms*)

Compose multiple transforms sequentially.

Parameters **transforms** (*Sequence[dict, callable], optional*) – Sequence of transform object or config dict to be composed.

44.2 Dataset Wrapper

ClassBalancedDataset	A wrapper of class balanced dataset.
ConcatDataset	A wrapper of concatenated dataset.
RepeatDataset	A wrapper of repeated dataset.

44.2.1 ClassBalancedDataset

class mmengine.dataset.**ClassBalancedDataset**(*dataset*, *oversample_thr*, *lazy_init=False*)

A wrapper of class balanced dataset.

Suitable for training on class imbalanced datasets like LVIS. Following the sampling strategy in the paper, in each epoch, an image may appear multiple times based on its "repeat factor". The repeat factor for an image is a function of the frequency the rarest category labeled in that image. The "frequency of category c" in [0, 1] is defined by the fraction of images in the training set (without repeats) in which category c appears. The dataset needs to instantiate *get_cat_ids()* to support ClassBalancedDataset.

The repeat factor is computed as followed.

1. For each category c, compute the fraction # of images that contain it: $f(c)$
2. For each category c, compute the category-level repeat factor: $r(c) = max(1, sqrt(t/f(c)))$
3. For each image I, compute the image-level repeat factor: $r(I) = max_{cinI} r(c)$

Note: `ClassBalancedDataset` should not inherit from `BaseDataset` since `get_subset` and `get_subset_` could produce ambiguous meaning sub-dataset which conflicts with original dataset. If you want to use a sub-dataset of `ClassBalancedDataset`, you should set `indices` arguments for wrapped dataset which inherit from `BaseDataset`.

Parameters

- **dataset** (*BaseDataset or dict*) – The dataset to be repeated.
- **oversample_thr** (*float*) – frequency threshold below which data is repeated. For categories with `f_c >= oversample_thr`, there is no oversampling. For categories with `f_c < oversample_thr`, the degree of oversampling following the square-root inverse frequency heuristic above.
- **lazy_init** (*bool, optional*) – whether to load annotation during instantiation. Defaults to False

full_init()
Loop to full_init each dataset.

get_cat_ids(*idx*)
Get category ids of class balanced dataset by index.

Parameters **idx** (*int*) – Index of data.

Returns All categories in the image of specified index.

Return type List[int]

get_data_info(*idx*)
Get annotation by index.

Parameters **idx** (*int*) – Global index of ConcatDataset.

Returns The idx-th annotation of the dataset.

Return type dict

get_subset(*indices*)
Not supported in ClassBalancedDataset for the ambiguous meaning of sub-dataset.

Parameters **indices** (*Union[List[int], int]*) –

Return type *mmengine.dataset.base_dataset.BaseDataset*

get_subset_(*indices*)
Not supported in ClassBalancedDataset for the ambiguous meaning of sub-dataset.

Parameters **indices** (*Union[List[int], int]*) –

Return type None

property metainfo: dict
Get the meta information of the repeated dataset.

Returns The meta information of repeated dataset.

Return type dict

44.2.2 ConcatDataset

class mmengine.dataset.**ConcatDataset**(*datasets*, *lazy_init=False*, *ignore_keys=None*)
A wrapper of concatenated dataset.

Same as torch.utils.data.dataset.ConcatDataset and support lazy_init.

Note: ConcatDataset should not inherit from BaseDataset since get_subset and get_subset_ could produce ambiguous meaning sub-dataset which conflicts with original dataset. If you want to use a sub-dataset of ConcatDataset, you should set indices arguments for wrapped dataset which inherit from BaseDataset.

Parameters

- **datasets** (*Sequence[BaseDataset] or Sequence[dict]*) – A list of datasets which will be concatenated.
- **lazy_init** (*bool, optional*) – Whether to load annotation during instantiation. Defaults to False.

- **ignore_keys** (*List[str] or str*) – Ignore the keys that can be unequal in *dataset.metainfo*. Defaults to None. *New in version 0.3.0.*

full_init()
Loop to full_init each dataset.

get_data_info(*idx*)
Get annotation by index.

Parameters **idx** (*int*) – Global index of ConcatDataset.

Returns The idx-th annotation of the datasets.

Return type dict

get_subset(*indices*)
Not supported in ConcatDataset for the ambiguous meaning of sub- dataset.

Parameters **indices** (*Union[List[int], int]*) –

Return type *mmengine.dataset.base_dataset.BaseDataset*

get_subset_(*indices*)
Not supported in ConcatDataset for the ambiguous meaning of sub- dataset.

Parameters **indices** (*Union[List[int], int]*) –

Return type None

property metainfo: dict
Get the meta information of the first dataset in self.datasets.

Returns Meta information of first dataset.

Return type dict

44.2.3 RepeatDataset

class mmengine.dataset.RepeatDataset(*dataset*, *times*, *lazy_init=False*)
A wrapper of repeated dataset.

The length of repeated dataset will be *times* larger than the original dataset. This is useful when the data loading time is long but the dataset is small. Using RepeatDataset can reduce the data loading time between epochs.

Note: RepeatDataset should not inherit from BaseDataset since get_subset and get_subset_ could produce ambiguous meaning sub-dataset which conflicts with original dataset. If you want to use a sub-dataset of RepeatDataset, you should set indices arguments for wrapped dataset which inherit from BaseDataset.

Parameters

- **dataset** (BaseDataset *or* *dict*) – The dataset to be repeated.
- **times** (*int*) – Repeat times.
- **lazy_init** (*bool*) – Whether to load annotation during instantiation. Defaults to False.

full_init()
Loop to full_init each dataset.

get_data_info(*idx*)
Get annotation by index.

Parameters **idx** (*int*) – Global index of ConcatDataset.

Returns The idx-th annotation of the datasets.

Return type dict

get_subset(*indices*)
Not supported in RepeatDataset for the ambiguous meaning of sub- dataset.

Parameters **indices** (*Union[List[int], int]*) –

Return type *mmengine.dataset.base_dataset.BaseDataset*

get_subset_(*indices*)
Not supported in RepeatDataset for the ambiguous meaning of sub- dataset.

Parameters **indices** (*Union[List[int], int]*) –

Return type None

property metainfo: dict
Get the meta information of the repeated dataset.

Returns The meta information of repeated dataset.

Return type dict

44.3 Sampler

DefaultSampler	The default data sampler for both distributed and non-distributed environment.
InfiniteSampler	It's designed for iteration-based runner and yields a mini-batch indices each time.

44.3.1 DefaultSampler

class mmengine.dataset.**DefaultSampler**(*dataset*, *shuffle=True*, *seed=None*, *round_up=True*)
The default data sampler for both distributed and non-distributed environment.

It has several differences from the PyTorch DistributedSampler as below:

1. This sampler supports non-distributed environment.
2. The round up behaviors are a little different.
   - If round_up=True, this sampler will add extra samples to make the number of samples is evenly divisible by the world size. And this behavior is the same as the DistributedSampler with drop_last=False.
   - If round_up=False, this sampler won't remove or add any samples while the DistributedSampler with drop_last=True will remove tail samples.

Parameters

- **dataset** (*Sized*) – The dataset.
- **shuffle** (*bool*) – Whether shuffle the dataset or not. Defaults to True.

- **seed** (*int, optional*) – Random seed used to shuffle the sampler if `shuffle=True`. This number should be identical across all processes in the distributed group. Defaults to None.
- **round_up** (*bool*) – Whether to add extra samples to make the number of samples evenly divisible by the world size. Defaults to True.

set_epoch(*epoch*)

Sets the epoch for this sampler.

When `shuffle=True`, this ensures all replicas use a different random ordering for each epoch. Otherwise, the next iteration of this sampler will yield the same ordering.

Parameters **epoch** (*int*) – Epoch number.

Return type None

44.3.2 InfiniteSampler

class mmengine.dataset.**InfiniteSampler**(*dataset*, *shuffle=True*, *seed=None*)

It's designed for iteration-based runner and yields a mini-batch indices each time.

The implementation logic is referred to https://github.com/facebookresearch/detectron2/blob/main/detectron2/data/samplers/distributed_sampler.py

Parameters

- **dataset** (*Sized*) – The dataset.
- **shuffle** (*bool*) – Whether shuffle the dataset or not. Defaults to True.
- **seed** (*int, optional*) – Random seed. If None, set a random seed. Defaults to None.

set_epoch(*epoch*)

Not supported in iteration-based runner.

Parameters **epoch** (*int*) –

Return type None

44.4 Utils

default_collate	Convert list of data sampled from dataset into a batch of data, of which type consistent with the type of each data_itement in `data_batch`.
pseudo_collate	Convert list of data sampled from dataset into a batch of data, of which type consistent with the type of each data_itement in `data_batch`.
worker_init_fn	This function will be called on each worker subprocess after seeding and before data loading.

44.4.1 mmengine.dataset.default_collate

mmengine.dataset.**default_collate**(*data_batch*)

Convert list of data sampled from dataset into a batch of data, of which type consistent with the type of each data_itement in `data_batch`.

Different from *pseudo_collate()*, `default_collate` will stack tensor contained in `data_batch` into a batched tensor with the first dimension batch size, and then move input tensor to the target device.

Different from `default_collate` in pytorch, `default_collate` will not process `BaseDataElement`.

This code is referenced from: Pytorch default_collate. # noqa: E501

Note: `default_collate` only accept input tensor with the same shape.

Parameters **data_batch** (*Sequence*) – Data sampled from dataset.

Returns Data in the same format as the data_itement of `data_batch`, of which tensors have been stacked, and ndarray, int, float have been converted to tensors.

Return type Any

44.4.2 mmengine.dataset.pseudo_collate

mmengine.dataset.**pseudo_collate**(*data_batch*)

Convert list of data sampled from dataset into a batch of data, of which type consistent with the type of each data_itement in `data_batch`.

The default behavior of dataloader is to merge a list of samples to form a mini-batch of Tensor(s). However, in MMEngine, `pseudo_collate` will not stack tensors to batch tensors, and convert int, float, ndarray to tensors.

This code is referenced from: Pytorch default_collate. # noqa: E501 :param data_batch: Batch of data from dataloader. :type data_batch: Sequence

Returns Transversed Data in the same format as the data_itement of `data_batch`.

Return type Any

Parameters **data_batch** (*Sequence*) –

44.4.3 mmengine.dataset.worker_init_fn

mmengine.dataset.**worker_init_fn**(*worker_id*, *num_workers*, *rank*, *seed*)

This function will be called on each worker subprocess after seeding and before data loading.

Parameters

- **worker_id** (*int*) – Worker id in [0, num_workers - 1].
- **num_workers** (*int*) – How many subprocesses to use for data loading.
- **rank** (*int*) – Rank of process in distributed environment. If in non-distributed environment, it is a constant number *0*.
- **seed** (*int*) – Random seed.

Return type None

CHAPTER

FORTYFIVE

MMENGINE.DEVICE

get_device	Returns the currently existing device type.
get_max_cuda_memory	Returns the maximum GPU memory occupied by tensors in megabytes (MB) for a given device.
is_cuda_available	Returns True if cuda devices exist.
is_npu_available	Returns True if Ascend PyTorch and npu devices exist.
is_mlu_available	Returns True if Cambricon PyTorch and mlu devices exist.
is_mps_available	Return True if mps devices exist.

45.1 mmengine.device.get_device

mmengine.device.**get_device**()

Returns the currently existing device type.

Returns cuda | npu | mlu | mps | cpu.

Return type str

45.2 mmengine.device.get_max_cuda_memory

mmengine.device.**get_max_cuda_memory**(*device=None*)

Returns the maximum GPU memory occupied by tensors in megabytes (MB) for a given device. By default, this returns the peak allocated memory since the beginning of this program.

Parameters **device** (*torch.device, optional*) – selected device. Returns statistic for the current device, given by `current_device()`, if `device` is None. Defaults to None.

Returns The maximum GPU memory occupied by tensors in megabytes for a given device.

Return type int

45.3 mmengine.device.is_cuda_available

mmengine.device.**is_cuda_available**()
Returns True if cuda devices exist.

Return type bool

45.4 mmengine.device.is_npu_available

mmengine.device.**is_npu_available**()
Returns True if Ascend PyTorch and npu devices exist.

Return type bool

45.5 mmengine.device.is_mlu_available

mmengine.device.**is_mlu_available**()
Returns True if Cambricon PyTorch and mlu devices exist.

Return type bool

45.6 mmengine.device.is_mps_available

mmengine.device.**is_mps_available**()
Return True if mps devices exist.

It's specialized for mac m1 chips and require torch version 1.12 or higher.

Return type bool

CHAPTER

FORTYSIX

MMENGINE.HUB

get_config	Get config from external package.
get_model	Get built model from external package.

46.1 mmengine.hub.get_config

mmengine.hub.**get_config**(*cfg_path*, *pretrained=False*)

Get config from external package.

Parameters

- **cfg_path** (*str*) – External relative config path.
- **pretrained** (*bool*) – Whether to save pretrained model path. If `pretrained==True`, the url of pretrained model can be accessed by `cfg.model_path`. Defaults to False.

Return type *mmengine.config.config.Config*

Examples

```
>>> cfg = get_config('mmdet::faster_rcnn/faster-rcnn_r50_fpn_1x_coco.py',␣
↪pretrained=True)
>>> # Equivalent to
>>> # cfg = Config.fromfile('/path/to/faster-rcnn_r50_fpn_1x_coco.py')
>>> cfg.model_path
https://download.openmmlab.com/mmdetection/v2.0/faster_rcnn/faster_rcnn_r50_fpn_1x_
↪coco/faster_rcnn_r50_fpn_1x_coco_20200130-047c8118.pth
```

Returns A *Config* parsed from external package.

Return type *Config*

Parameters

- **cfg_path** (*str*) –
- **pretrained** (*bool*) –

46.2 mmengine.hub.get_model

mmengine.hub.**get_model**(*cfg_path*, *pretrained=False*, ***kwargs*)

Get built model from external package.

Parameters

- **cfg_path** (*str*) – External relative config path with prefix 'package::' and without suffix.
- **pretrained** (*bool*) – Whether to load pretrained model. Defaults to False.
- **kwargs** (*dict*) – Default arguments to build model.

Examples

```
>>> model = get_model('mmdet::faster_rcnn/faster-rcnn_r50_fpn_1x_coco.py',␣
↪pretrained=True)
>>> type(model)
<class 'mmdet.models.detectors.faster_rcnn.FasterRCNN'>
```

Returns Built model.

Return type nn.Module

Parameters

- **cfg_path** (*str*) –
- **pretrained** (*bool*) –

CHAPTER

FORTYSEVEN

MMENGINE.LOGGING

MMLogger	Formatted logger used to record messages.
MessageHub	Message hub for component interaction.
HistoryBuffer	Unified storage format for different log types.

47.1 MMLogger

class mmengine.logging.**MMLogger**(*name*, *logger_name='mmengine'*, *log_file=None*, *log_level='INFO'*, *file_mode='w'*, *distributed=False*)

Formatted logger used to record messages.

`MMLogger` can create formatted logger to log message with different log levels and get instance in the same way as `ManagerMixin`. `MMLogger` has the following features:

- Distributed log storage, `MMLogger` can choose whether to save log of different ranks according to *log_file*.
- Message with different log levels will have different colors and format when displayed on terminal.

Note:

- The *name* of logger and the `instance_name` of `MMLogger` could be different. We can only get `MMLogger` instance by `MMLogger.get_instance` but not `logging.getLogger`. This feature ensures `MMLogger` will not be incluenced by third-party logging config.
- Different from `logging.Logger`, `MMLogger` will not log warning or error message without `Handler`.

Examples

```
>>> logger = MMLogger.get_instance(name='MMLogger',
>>>                                logger_name='Logger')
>>> # Although logger has name attribute just like `logging.Logger`
>>> # We cannot get logger instance by `logging.getLogger`.
>>> assert logger.name == 'Logger'
>>> assert logger.instance_name = 'MMLogger'
>>> assert id(logger) != id(logging.getLogger('Logger'))
>>> # Get logger that do not store logs.
>>> logger1 = MMLogger.get_instance('logger1')
>>> # Get logger only save rank0 logs.
```

(continues on next page)

(continued from previous page)

```
>>> logger2 = MMLogger.get_instance('logger2', log_file='out.log')
>>> # Get logger only save multiple ranks logs.
>>> logger3 = MMLogger.get_instance('logger3', log_file='out.log',
>>>                                 distributed=True)
```

Parameters

- **name** (*str*) – Global instance name.
- **logger_name** (*str*) – `name` attribute of `Logging.Logger` instance. If *logger_name* is not defined, defaults to 'mmengine'.
- **log_file** (*str, optional*) – The log filename. If specified, a `FileHandler` will be added to the logger. Defaults to None.
- **log_level** (*str*) – The log level of the handler and logger. Defaults to "NOTSET".
- **file_mode** (*str*) – The file mode used to open log file. Defaults to 'w'.
- **distributed** (*bool*) – Whether to save distributed logs, Defaults to false.

`callHandlers`(*record*)

Pass a record to all relevant handlers.

Override `callHandlers` method in `logging.Logger` to avoid multiple warning messages in DDP mode. Loop through all handlers of the logger instance and its parents in the logger hierarchy. If no handler was found, the record will not be output.

Parameters **record** (*LogRecord*) – A `LogRecord` instance contains logged message.

Return type None

`classmethod get_current_instance`()

Get latest created `MMLogger` instance.

MMLogger can call *get_current_instance()* before any instance has been created, and return a logger with the instance name "mmengine".

Returns Configured logger instance.

Return type *MMLogger*

`setLevel`(*level*)

Set the logging level of this logger.

If `logging.Logger.selLevel` is called, all `logging.Logger` instances managed by `logging.Manager` will clear the cache. Since `MMLogger` is not managed by `logging.Manager` anymore, `MMLogger` should override this method to clear caches of all `MMLogger` instance which is managed by `ManagerMixin`.

level must be an int or a str.

47.2 MessageHub

class mmengine.logging.**MessageHub**(*name*, *log_scalars=None*, *runtime_info=None*, *resumed_keys=None*)

Message hub for component interaction. MessageHub is created and accessed in the same way as ManagerMixin.

MessageHub will record log information and runtime information. The log information refers to the learning rate, loss, etc. of the model during training phase, which will be stored as `HistoryBuffer`. The runtime information refers to the iter times, meta information of runner etc., which will be overwritten by next update.

Parameters

- **name** (*str*) – Name of message hub used to get corresponding instance globally.
- **log_scalars** (*OrderedDict, optional*) – Each key-value pair in the dictionary is the name of the log information such as "loss", "lr", "metric" and their corresponding values. The type of value must be HistoryBuffer. Defaults to None.
- **runtime_info** (*OrderedDict, optional*) – Each key-value pair in the dictionary is the name of the runtime information and their corresponding values. Defaults to None.
- **resumed_keys** (*OrderedDict, optional*) – Each key-value pair in the dictionary decides whether the key in `_log_scalars` and `_runtime_info` will be serialized.

Note: Key in `_resumed_keys` belongs to `_log_scalars` or `_runtime_info`. The corresponding value cannot be set repeatedly.

Examples

```
>>> # create empty `MessageHub`.
>>> message_hub1 = MessageHub()
>>> log_scalars = OrderedDict(loss=HistoryBuffer())
>>> runtime_info = OrderedDict(task='task')
>>> resumed_keys = dict(loss=True)
>>> # create `MessageHub` from data.
>>> message_hub2 = MessageHub(
>>>     name='name',
>>>     log_scalars=log_scalars,
>>>     runtime_info=runtime_info,
>>>     resumed_keys=resumed_keys)
```

classmethod get_current_instance()

Get latest created `MessageHub` instance.

MessageHub can call *get_current_instance()* before any instance has been created, and return a message hub with the instance name "mmengine".

Returns Empty `MessageHub` instance.

Return type *MessageHub*

get_info(*key*)

Get runtime information by key.

Parameters **key** (*str*) – Key of runtime information.

Returns A copy of corresponding runtime information if the key exists.

Return type Any

get_scalar(*key*)
Get `HistoryBuffer` instance by key.

Note: Considering the large memory footprint of history buffers in the post-training, *get_scalar()* will not return a reference of history buffer rather than a copy.

Parameters **key** (*str*) – Key of `HistoryBuffer`.

Returns Corresponding `HistoryBuffer` instance if the key exists.

Return type *HistoryBuffer*

load_state_dict(*state_dict*)
Loads log scalars, runtime information and resumed keys from `state_dict` or `message_hub`.

If `state_dict` is a dictionary returned by *state_dict()*, it will only make copies of data which should be resumed from the source `message_hub`.

If `state_dict` is a `message_hub` instance, it will make copies of all data from the source message_hub. We suggest to load data from `dict` rather than a `MessageHub` instance.

Parameters **state_dict** (*dict or MessageHub*) – A dictionary contains key `log_scalars` `runtime_info` and `resumed_keys`, or a MessageHub instance.

Return type None

property log_scalars: collections.OrderedDict
Get all `HistoryBuffer` instances.

Note: Considering the large memory footprint of history buffers in the post-training, *get_scalar()* will return a reference of history buffer rather than a copy.

Returns All `HistoryBuffer` instances.

Return type OrderedDict

property runtime_info: collections.OrderedDict
Get all runtime information.

Returns A copy of all runtime information.

Return type OrderedDict

state_dict()
Returns a dictionary containing log scalars, runtime information and resumed keys, which should be resumed.

The returned `state_dict` can be loaded by *load_state_dict()*.

Returns A dictionary contains `log_scalars`, `runtime_info` and `resumed_keys`.

Return type dict

update_info(*key*, *value*, *resumed=True*)
Update runtime information.

The key corresponding runtime information will be overwritten each time calling `update_info`.

Note: The `resumed` argument needs to be consistent for the same `key`.

Examples

```
>>> message_hub = MessageHub()
>>> message_hub.update_info('iter', 100)
```

Parameters

- **key** (*str*) – Key of runtime information.
- **value** (*Any*) – Value of runtime information.
- **resumed** (*bool*) – Whether the corresponding `HistoryBuffer` could be resumed.

Return type None

update_info_dict(*info_dict*, *resumed=True*)

Update runtime information with dictionary.

The key corresponding runtime information will be overwritten each time calling `update_info`.

Note: The `resumed` argument needs to be consistent for the same `info_dict`.

Examples

```
>>> message_hub = MessageHub()
>>> message_hub.update_info({'iter': 100})
```

Parameters

- **info_dict** (*str*) – Runtime information dictionary.
- **resumed** (*bool*) – Whether the corresponding `HistoryBuffer` could be resumed.

Return type None

update_scalar(*key*, *value*, *count=1*, *resumed=True*)

Update :attr:_log_scalars.

Update `HistoryBuffer` in `_log_scalars`. If corresponding key `HistoryBuffer` has been created, `value` and `count` is the argument of `HistoryBuffer.update`, Otherwise, `update_scalar` will create an `HistoryBuffer` with value and count via the constructor of `HistoryBuffer`.

Examples

```
>>> message_hub = MessageHub
>>> # create loss `HistoryBuffer` with value=1, count=1
>>> message_hub.update_scalar('loss', 1)
>>> # update loss `HistoryBuffer` with value
>>> message_hub.update_scalar('loss', 3)
>>> message_hub.update_scalar('loss', 3, resumed=False)
AssertionError: loss used to be true, but got false now. resumed
keys cannot be modified repeatedly'
```

Note: The `resumed` argument needs to be consistent for the same `key`.

Parameters

- **key** (*str*) – Key of `HistoryBuffer`.
- **value** (*torch.Tensor or np.ndarray or int or float*) – Value of log.
- **count** (*torch.Tensor or np.ndarray or int or float*) – Accumulation times of log, defaults to 1. *count* will be used in smooth statistics.
- **resumed** (*str*) – Whether the corresponding `HistoryBuffer` could be resumed. Defaults to True.

Return type None

update_scalars(*log_dict*, *resumed=True*)

Update `_log_scalars` with a dict.

`update_scalars` iterates through each pair of log_dict key-value, and calls `update_scalar`. If type of value is dict, the value should be `dict(value=xxx) or dict(value=xxx, count=xxx)`. Item in `log_dict` has the same resume option.

Note: The `resumed` argument needs to be consistent for the same `log_dict`.

Parameters

- **log_dict** (*str*) – Used for batch updating `_log_scalars`.
- **resumed** (*bool*) – Whether all `HistoryBuffer` referred in log_dict should be resumed. Defaults to True.

Return type None

Examples

```
>>> message_hub = MessageHub.get_instance('mmengine')
>>> log_dict = dict(a=1, b=2, c=3)
>>> message_hub.update_scalars(log_dict)
>>> # The default count of  `a`, `b` and `c` is 1.
>>> log_dict = dict(a=1, b=2, c=dict(value=1, count=2))
>>> message_hub.update_scalars(log_dict)
>>> # The count of `c` is 2.
```

47.3 HistoryBuffer

class mmengine.logging.**HistoryBuffer**(*log_history=[]*, *count_history=[]*, *max_length=1000000*)

Unified storage format for different log types.

`HistoryBuffer` records the history of log for further statistics.

Examples

```
>>> history_buffer = HistoryBuffer()
>>> # Update history_buffer.
>>> history_buffer.update(1)
>>> history_buffer.update(2)
>>> history_buffer.min()  # minimum of (1, 2)
1
>>> history_buffer.max()  # maximum of (1, 2)
2
>>> history_buffer.mean()  # mean of (1, 2)
1.5
>>> history_buffer.statistics('mean')  # access method by string.
1.5
```

Parameters

- **log_history** (*Sequence*) – History logs. Defaults to [].
- **count_history** (*Sequence*) – Counts of history logs. Defaults to [].
- **max_length** (*int*) – The max length of history logs. Defaults to 1000000.

current()

Return the recently updated values in log histories.

Returns Recently updated values in log histories.

Return type np.ndarray

property data: Tuple[numpy.ndarray, numpy.ndarray]

Get the `_log_history` and `_count_history`.

Returns History logs and the counts of the history logs.

Return type Tuple[np.ndarray, np.ndarray]

max(*window_size=None*)
Return the maximum value of the latest `window_size` values in log histories.

If `window_size is None` or `window_size > len(self._log_history)`, return the global maximum value of history logs.

Parameters **window_size** (*int, optional*) – Size of statistics window.

Returns The maximum value within the window.

Return type np.ndarray

mean(*window_size=None*)
Return the mean of the latest `window_size` values in log histories.

If `window_size is None` or `window_size > len(self._log_history)`, return the global mean value of history logs.

Parameters **window_size** (*int, optional*) – Size of statistics window.

Returns Mean value within the window.

Return type np.ndarray

min(*window_size=None*)
Return the minimum value of the latest `window_size` values in log histories.

If `window_size is None` or `window_size > len(self._log_history)`, return the global minimum value of history logs.

Parameters **window_size** (*int, optional*) – Size of statistics window.

Returns The minimum value within the window.

Return type np.ndarray

classmethod register_statistics(*method*)
Register custom statistics method to `_statistics_methods`.

The registered method can be called by `history_buffer.statistics` with corresponding method name and arguments.

Examples

```
>>> @HistoryBuffer.register_statistics
>>> def weighted_mean(self, window_size, weight):
>>>     assert len(weight) == window_size
>>>     return (self._log_history[-window_size:] *
>>>             np.array(weight)).sum() /             >>>             self._
↪count_history[-window_size:]
```

```
>>> log_buffer = HistoryBuffer([1, 2], [1, 1])
>>> log_buffer.statistics('weighted_mean', 2, [2, 1])
2
```

Parameters **method** (*Callable*) – Custom statistics method.

Returns Original custom statistics method.

Return type Callable

statistics(*method_name*, **arg*, ***kwargs*)
Access statistics method by name.

Parameters **method_name** (*str*) – Name of method.

Returns Depends on corresponding method.

Return type Any

update(*log_val*, *count=1*)
update the log history.

If the length of the buffer exceeds `self._max_length`, the oldest element will be removed from the buffer.

Parameters

- **log_val** (*int or float*) – The value of log.
- **count** (*int*) – The accumulation times of log, defaults to 1.
- **will be used in smooth statistics.** (*count*) –

Return type None

print_log	Print a log message.

47.4 mmengine.logging.print_log

mmengine.logging.**print_log**(*msg*, *logger=None*, *level=20*)
Print a log message.

Parameters

- **msg** (*str*) – The message to be logged.
- **logger** (*Logger or str, optional*) – If the type of logger is
- **logging.Logger** – Some special loggers are:
  - ”silent”: No message will be printed.
  - ”current”: Use latest created logger to log message.
  - other str: Instance name of logger. The corresponding logger will log message if it has been created, otherwise `print_log` will raise a *ValueError*.
  - None: The *print()* method will be used to print log messages.
- **directly use logger to log messages.** (*we*) – Some special loggers are:
  - ”silent”: No message will be printed.
  - ”current”: Use latest created logger to log message.
  - other str: Instance name of logger. The corresponding logger will log message if it has been created, otherwise `print_log` will raise a *ValueError*.
  - None: The *print()* method will be used to print log messages.
- **level** (*int*) – Logging level. Only available when *logger* is a Logger object, “current”, or a created logger instance name.

Return type None

CHAPTER

FORTYEIGHT

MMENGINE.VISUALIZATION

mmengine.visualization

- *Visualizer*
- *visualization Backend*

48.1 Visualizer

Visualizer	MMEngine provides a Visualizer class that uses the `Matplotlib` library as the backend.

48.1.1 Visualizer

class mmengine.visualization.**Visualizer**(*name='visualizer'*, *image=None*, *vis_backends=None*, *save_dir=None*, *fig_save_cfg={'frameon': False}*, *fig_show_cfg={'frameon': False}*)

MMEngine provides a Visualizer class that uses the `Matplotlib` library as the backend. It has the following functions:

- Basic drawing methods
  - draw_bboxes: draw single or multiple bounding boxes
  - draw_texts: draw single or multiple text boxes
  - draw_points: draw single or multiple points
  - draw_lines: draw single or multiple line segments
  - draw_circles: draw single or multiple circles
  - draw_polygons: draw single or multiple polygons
  - draw_binary_masks: draw single or multiple binary masks
  - draw_featmap: draw feature map
- Basic visualizer backend methods
  - add_configs: write config to all vis storage backends
  - add_graph: write model graph to all vis storage backends

  - add_image: write image to all vis storage backends
  - add_scalar: write scalar to all vis storage backends
  - add_scalars: write scalars to all vis storage backends
  - add_datasample: write datasample to all vis storage backends. The abstract drawing interface used by the user

- Basic info methods
  - set_image: sets the original image data
  - get_image: get the image data in Numpy format after drawing
  - show: visualization
  - close: close all resources that have been opened
  - get_backend: get the specified vis backend

All the basic drawing methods support chain calls, which is convenient for overlaydrawing and display. Each downstream algorithm library can inherit `Visualizer` and implement the add_datasample logic. For example, `DetLocalVisualizer` in MMDetection inherits from `Visualizer` and implements functions, such as visual detection boxes, instance masks, and semantic segmentation maps in the add_datasample interface.

Parameters

- **name** (*str*) – Name of the instance. Defaults to 'visualizer'.
- **image** (*np.ndarray, optional*) – the origin image to draw. The format should be RGB. Defaults to None.
- **vis_backends** (*list, optional*) – Visual backend config list. Default to None.
- **save_dir** (*str, optional*) – Save file dir for all storage backends. If it is None, the backend storage will not save any data.
- **fig_save_cfg** (*dict*) – Keyword parameters of figure for saving. Defaults to empty dict.
- **fig_show_cfg** (*dict*) – Keyword parameters of figure for showing. Defaults to empty dict.

Return type None

Examples

```
>>> # Basic info methods
>>> vis = Visualizer()
>>> vis.set_image(image)
>>> vis.get_image()
>>> vis.show()
```

```
>>> # Basic drawing methods
>>> vis = Visualizer(image=image)
>>> vis.draw_bboxes(np.array([0, 0, 1, 1]), edge_colors='g')
>>> vis.draw_bboxes(bbox=np.array([[1, 1, 2, 2], [2, 2, 3, 3]]),
>>>                 edge_colors=['g', 'r'])
>>> vis.draw_lines(x_datas=np.array([1, 3]),
>>>                y_datas=np.array([1, 3]),
>>>                colors='r', line_widths=1)
>>> vis.draw_lines(x_datas=np.array([[1, 3], [2, 4]]),
```

(continues on next page)

(continued from previous page)

```
>>>                y_datas=np.array([[1, 3], [2, 4]]),
>>>                colors=['r', 'r'], line_widths=[1, 2])
>>> vis.draw_texts(text='MMEngine',
>>>                position=np.array([2, 2]),
>>>                colors='b')
>>> vis.draw_texts(text=['MMEngine','OpenMMLab'],
>>>                position=np.array([[2, 2], [5, 5]]),
>>>                colors=['b', 'b'])
>>> vis.draw_circles(circle_coord=np.array([2, 2]), radius=np.array[1])
>>> vis.draw_circles(circle_coord=np.array([[2, 2], [3, 5]),
>>>                  radius=np.array[1, 2], colors=['g', 'r'])
>>> vis.draw_polygons(np.array([0, 0, 1, 0, 1, 1, 0, 1]),
>>>                   edge_colors='g')
>>> vis.draw_polygons(bbox=[np.array([0, 0, 1, 0, 1, 1, 0, 1],
>>>                       np.array([2, 2, 3, 2, 3, 3, 2, 3]],
>>>                   edge_colors=['g', 'r'])
>>> vis.draw_binary_masks(binary_mask, alpha=0.6)
>>> heatmap = vis.draw_featmap(featmap, img,
>>>                            channel_reduction='select_max')
>>> heatmap = vis.draw_featmap(featmap, img, channel_reduction=None,
>>>                            topk=8, arrangement=(4, 2))
>>> heatmap = vis.draw_featmap(featmap, img, channel_reduction=None,
>>>                            topk=-1)
```

```
>>> # chain calls
>>> vis.draw_bboxes().draw_texts().draw_circle().draw_binary_masks()
```

```
>>> # Backend related methods
>>> vis = Visualizer(vis_backends=[dict(type='LocalVisBackend')],
>>>                                save_dir='temp_dir')
>>> cfg = Config(dict(a=1, b=dict(b1=[0, 1])))
>>> vis.add_config(cfg)
>>> image=np.random.randint(0, 256, size=(10, 10, 3)).astype(np.uint8)
>>> vis.add_image('image',image)
>>> vis.add_scaler('mAP', 0.6)
>>> vis.add_scalars({'loss': 0.1,'acc':0.8})
```

```
>>> # inherit
>>> class DetLocalVisualizer(Visualizer):
>>>        def add_datasample(self,
>>>                           name,
>>>                           image: np.ndarray,
>>>                           gt_sample:
>>>                               Optional['BaseDataElement'] = None,
>>>                           pred_sample:
>>>                               Optional['BaseDataElement'] = None,
>>>                           draw_gt: bool = True,
>>>                           draw_pred: bool = True,
>>>                           show: bool = False,
>>>                           wait_time: int = 0,
>>>                           step: int = 0) -> None:
```

(continues on next page)

(continued from previous page)

```
>>>             pass
```

add_config(*config*, ***kwargs*)
Record the config.

Parameters **config** (Config) – The Config object.

add_datasample(*name*, *image*, *data_sample=None*, *draw_gt=True*, *draw_pred=True*, *show=False*, *wait_time=0*, *step=0*)
Draw datasample.

Parameters

- **image** (*numpy.ndarray*) –
- **data_sample** (*Optional[mmengine.structures.base_data_element.BaseDataElement]*) –
- **draw_gt** (*bool*) –
- **draw_pred** (*bool*) –
- **show** (*bool*) –
- **wait_time** (*int*) –
- **step** (*int*) –

Return type None

add_graph(*model*, *data_batch*, ***kwargs*)
Record the model graph.

Parameters

- **model** (*torch.nn.Module*) – Model to draw.
- **data_batch** (*Sequence[dict]*) – Batch of data from dataloader.

Return type None

add_image(*name*, *image*, *step=0*)
Record the image.

Parameters

- **name** (*str*) – The image identifier.
- **image** (*np.ndarray, optional*) – The image to be saved. The format should be RGB. Default to None.
- **step** (*int*) – Global step value to record. Default to 0.

Return type None

add_scalar(*name*, *value*, *step=0*, ***kwargs*)
Record the scalar data.

Parameters

- **name** (*str*) – The scalar identifier.
- **value** (*float, int*) – Value to save.
- **step** (*int*) – Global step value to record. Default to 0.

Return type None

add_scalars(*scalar_dict*, *step=0*, *file_path=None*, ***kwargs*)
Record the scalars' data.

Parameters

- **scalar_dict** (*dict*) – Key-value pair storing the tag and corresponding values.
- **step** (*int*) – Global step value to record. Default to 0.
- **file_path** (*str, optional*) – The scalar's data will be saved to the *file_path* file at the same time if the *file_path* parameter is specified. Default to None.

Return type None

close()
close an opened object.

Return type None

property dataset_meta: Optional[dict]
Meta info of the dataset.

Type Optional[dict]

draw_bboxes(*bboxes*, *edge_colors='g'*, *line_styles='-'*, *line_widths=2*, *face_colors='none'*, *alpha=0.8*)
Draw single or multiple bboxes.

Parameters

- **bboxes** (*Union[np.ndarray, torch.Tensor]*) – The bboxes to draw with the format of(x1,y1,x2,y2).
- **edge_colors** (*Union[str, tuple, List[str], List[tuple]]*) – The colors of bboxes. `colors` can have the same length with lines or just single value. If `colors` is single value, all the lines will have the same colors. Refer to *matplotlib. colors* for full list of formats that are accepted. Defaults to 'g'.
- **line_styles** (*Union[str, List[str]]*) – The linestyle of lines. `line_styles` can have the same length with texts or just single value. If `line_styles` is single value, all the lines will have the same linestyle. Reference to https://matplotlib.org/stable/api/collections_api.html?highlight=collection#matplotlib.collections.AsteriskPolygonCollection.set_linestyle for more details. Defaults to '-'.
- **line_widths** (*Union[Union[int, float], List[Union[int, float]]]*) – The linewidth of lines. `line_widths` can have the same length with lines or just single value. If `line_widths` is single value, all the lines will have the same linewidth. Defaults to 2.
- **face_colors** (*Union[str, tuple, List[str], List[tuple]]*) – The face colors. Default to None.
- **alpha** (*Union[int, float]*) – The transparency of bboxes. Defaults to 0.8.

Return type *mmengine.visualization.visualizer.Visualizer*

draw_binary_masks(*binary_masks*, *colors='g'*, *alphas=0.8*)
Draw single or multiple binary masks.

Parameters

- **binary_masks** (*np.ndarray, torch.Tensor*) – The binary_masks to draw with of shape (N, H, W), where H is the image height and W is the image width. Each value in the array is either a 0 or 1 value of uint8 type.
- **colors** (*np.ndarray*) – The colors which binary_masks will convert to. `colors` can have the same length with binary_masks or just single value. If `colors` is single value, all

the binary_masks will convert to the same colors. The colors format is RGB. Defaults to np.array([0, 255, 0]).

- **alphas** (*Union[int, List[int]]*) – The transparency of masks. Defaults to 0.8.

Return type *mmengine.visualization.visualizer.Visualizer*

draw_circles(*center*, *radius*, *edge_colors='g'*, *line_styles='-'*, *line_widths=2*, *face_colors='none'*, *alpha=0.8*)

Draw single or multiple circles.

Parameters

- **center** (*Union[np.ndarray, torch.Tensor]*) – The x coordinate of each line' start and end points.
- **radius** (*Union[np.ndarray, torch.Tensor]*) – The y coordinate of each line' start and end points.
- **edge_colors** (*Union[str, tuple, List[str], List[tuple]]*) – The colors of circles. `colors` can have the same length with lines or just single value. If `colors` is single value, all the lines will have the same colors. Reference to https://matplotlib.org/stable/gallery/color/named_colors.html for more details. Defaults to 'g.
- **line_styles** (*Union[str, List[str]]*) – The linestyle of lines. `line_styles` can have the same length with texts or just single value. If `line_styles` is single value, all the lines will have the same linestyle. Reference to https://matplotlib.org/stable/api/collections_api.html?highlight=collection#matplotlib.collections.AsteriskPolygonCollection.set_linestyle for more details. Defaults to '-'.
- **line_widths** (*Union[Union[int, float], List[Union[int, float]]]*) – The linewidth of lines. `line_widths` can have the same length with lines or just single value. If `line_widths` is single value, all the lines will have the same linewidth. Defaults to 2.
- **face_colors** (*Union[str, tuple, List[str], List[tuple]]*) – The face colors. Default to None.
- **alpha** (*Union[int, float]*) – The transparency of circles. Defaults to 0.8.

Return type *mmengine.visualization.visualizer.Visualizer*

static draw_featmap(*featmap*, *overlaid_image=None*, *channel_reduction='squeeze_mean'*, *topk=20*, *arrangement=(4, 5)*, *resize_shape=None*, *alpha=0.5*)

Draw featmap.

- If *overlaid_image* is not None, the final output image will be the

weighted sum of img and featmap.

- If *resize_shape* is specified, *featmap* and *overlaid_image*

are interpolated.

- If *resize_shape* is None and *overlaid_image* is not None,

the feature map will be interpolated to the spatial size of the image in the case where the spatial dimensions of *overlaid_image* and *featmap* are different.

- If *channel_reduction* is "squeeze_mean" and "select_max",

it will compress featmap to single channel image and weighted sum to *overlaid_image*.

- if *channel_reduction* is None

- If topk <= 0, featmap is assert to be one or three

channel and treated as image and will be weighted sum to `overlaid_image`. - If topk > 0, it will select topk channel to show by the sum of each channel. At the same time, you can specify the *arrangement* to set the window layout.

Parameters

- **featmap** (*torch.Tensor*) – The featmap to draw which format is (C, H, W).
- **overlaid_image** (*np.ndarray, optional*) – The overlaid image. Default to None.
- **channel_reduction** (*str, optional*) – Reduce multiple channels to a single channel. The optional value is 'squeeze_mean' or 'select_max'. Defaults to 'squeeze_mean'.
- **topk** (*int*) – If channel_reduction is not None and topk > 0, it will select topk channel to show by the sum of each channel. if topk <= 0, tensor_chw is assert to be one or three. Defaults to 20.
- **arrangement** (*Tuple[int, int]*) – The arrangement of featmap when channel_reduction is not None and topk > 0. Defaults to (4, 5).
- **resize_shape** (*tuple, optional*) – The shape to scale the feature map. Default to None.
- **alpha** (*Union[int, List[int]]*) – The transparency of featmap. Defaults to 0.5.

Returns RGB image.

Return type np.ndarray

draw_lines(*x_datas*, *y_datas*, *colors='g'*, *line_styles='-'*, *line_widths=2*)

Draw single or multiple line segments.

Parameters

- **x_datas** (*Union[np.ndarray, torch.Tensor]*) – The x coordinate of each line' start and end points.
- **y_datas** (*Union[np.ndarray, torch.Tensor]*) – The y coordinate of each line' start and end points.
- **colors** (*Union[str, tuple, List[str], List[tuple]]*) – The colors of lines. `colors` can have the same length with lines or just single value. If `colors` is single value, all the lines will have the same colors. Reference to https://matplotlib.org/stable/gallery/color/named_colors.html for more details. Defaults to 'g'.
- **line_styles** (*Union[str, List[str]]*) – The linestyle of lines. `line_styles` can have the same length with texts or just single value. If `line_styles` is single value, all the lines will have the same linestyle. Reference to https://matplotlib.org/stable/api/collections_api.html?highlight=collection#matplotlib.collections.AsteriskPolygonCollection.set_linestyle for more details. Defaults to '-'.
- **line_widths** (*Union[Union[int, float], List[Union[int, float]]]*) – The linewidth of lines. `line_widths` can have the same length with lines or just single value. If `line_widths` is single value, all the lines will have the same linewidth. Defaults to 2.

Return type *mmengine.visualization.visualizer.Visualizer*

draw_points(*positions*, *colors='g'*, *marker=None*, *sizes=None*)

Draw single or multiple points.

Parameters

- **positions** (*Union[np.ndarray, torch.Tensor]*) – Positions to draw.
- **colors** (*Union[str, tuple, List[str], List[tuple]]*) – The colors of points. `colors` can have the same length with points or just single value. If `colors` is single value, all the points will have the same colors. Reference to https://matplotlib.org/stable/gallery/color/named_colors.html for more details. Defaults to 'g.
- **marker** (*str, optional*) – The marker style. See `matplotlib.markers` for more information about marker styles. Default to None.
- **sizes** (*Optional[Union[np.ndarray, torch.Tensor]]*) – The marker size. Default to None.

draw_polygons(*polygons*, *edge_colors='g'*, *line_styles='-'*, *line_widths=2*, *face_colors='none'*, *alpha=0.8*)
Draw single or multiple bboxes.

Parameters

- **(Union[Union[np.ndarray** (*polygons*) – List[Union[np.ndarray, torch.Tensor]]]): The polygons to draw with the format of (x1,y1,x2,y2,...,xn,yn).
- **torch.Tensor]** – List[Union[np.ndarray, torch.Tensor]]]): The polygons to draw with the format of (x1,y1,x2,y2,...,xn,yn).
- **polygons** (*Union[numpy.ndarray, torch.Tensor, List[Union[numpy.ndarray, torch.Tensor]]]*) –
- **edge_colors** (*Union[str, tuple, List[str], List[tuple]]*) –
- **line_styles** (*Union[str, List[str]]*) –
- **line_widths** (*Union[int, float, List[Union[int, float]]]*) –
- **face_colors** (*Union[str, tuple, List[str], List[tuple]]*) –
- **alpha** (*Union[int, float]*) –

Return type *mmengine.visualization.visualizer.Visualizer*

:param [List[Union[np.ndarray, torch.Tensor]]]): The polygons to draw] with the format of (x1,y1,x2,y2,...,xn,yn).

Parameters

- **edge_colors** (*Union[str, tuple, List[str], List[tuple]]*) – The colors of polygons. `colors` can have the same length with lines or just single value. If `colors` is single value, all the lines will have the same colors. Refer to *matplotlib.colors* for full list of formats that are accepted. Defaults to 'g.
- **line_styles** (*Union[str, List[str]]*) – The linestyle of lines. `line_styles` can have the same length with texts or just single value. If `line_styles` is single value, all the lines will have the same linestyle. Reference to https://matplotlib.org/stable/api/collections_api.html?highlight=collection#matplotlib.collections.AsteriskPolygonCollection.set_linestyle for more details. Defaults to '-'.
- **line_widths** (*Union[Union[int, float], List[Union[int, float]]]*) – The linewidth of lines. `line_widths` can have the same length with lines or just single value. If `line_widths` is single value, all the lines will have the same linewidth. Defaults to 2.
- **face_colors** (*Union[str, tuple, List[str], List[tuple]]*) – The face colors. Default to None.
- **alpha** (*Union[int, float]*) – The transparency of polygons. Defaults to 0.8.

- **polygons** (*Union[numpy.ndarray, torch.Tensor, List[Union[numpy.ndarray, torch.Tensor]]]*) –

Return type *mmengine.visualization.visualizer.Visualizer*

draw_texts(*texts*, *positions*, *font_sizes=None*, *colors='g'*, *vertical_alignments='top'*, *horizontal_alignments='left'*, *font_families='sans-serif'*, *bboxes=None*)

Draw single or multiple text boxes.

Parameters

- **texts** (*Union[str, List[str]]*) – Texts to draw.
- **positions** (*Union[np.ndarray, torch.Tensor]*) – The position to draw the texts, which should have the same length with texts and each dim contain x and y.
- **font_sizes** (*Union[int, List[int]], optional*) – The font size of texts. font_sizes can have the same length with texts or just single value. If font_sizes is single value, all the texts will have the same font size. Defaults to None.
- **colors** (*Union[str, tuple, List[str], List[tuple]]*) – The colors of texts. colors can have the same length with texts or just single value. If colors is single value, all the texts will have the same colors. Reference to https://matplotlib.org/stable/gallery/color/named_colors.html for more details. Defaults to 'g.
- **vertical_alignments** (*Union[str, List[str]]*) – The verticalalignment of texts. verticalalignment controls whether the y positional argument for the text indicates the bottom, center or top side of the text bounding box. vertical_alignments can have the same length with texts or just single value. If vertical_alignments is single value, all the texts will have the same verticalalignment. verticalalignment can be 'center' or 'top', 'bottom' or 'baseline'. Defaults to 'top'.
- **horizontal_alignments** (*Union[str, List[str]]*) – The horizontalalignment of texts. Horizontalalignment controls whether the x positional argument for the text indicates the left, center or right side of the text bounding box. horizontal_alignments can have the same length with texts or just single value. If horizontal_alignments is single value, all the texts will have the same horizontalalignment. Horizontalalignment can be 'center','right' or 'left'. Defaults to 'left'.
- **font_families** (*Union[str, List[str]]*) – The font family of texts. font_families can have the same length with texts or just single value. If font_families is single value, all the texts will have the same font family. font_familiy can be 'serif', 'sans-serif', 'cursive', 'fantasy'

  or 'monospace'. Defaults to 'sans-serif'.
- **bboxes** (*Union[dict, List[dict]], optional*) – The bounding box of the texts. If bboxes is None, there are no bounding box around texts. bboxes can have the same length with texts or just single value. If bboxes is single value, all the texts will have the same bbox. Reference to https://matplotlib.org/stable/api/_as_gen/matplotlib.patches.FancyBboxPatch.html#matplotlib.patches.FancyBboxPatch for more details. Defaults to None.

Return type *mmengine.visualization.visualizer.Visualizer*

get_backend(*name*)

get vis backend by name.

Parameters **name** (*str*) – The name of vis backend

Returns The vis backend.

Return type *BaseVisBackend*

get_image()
Get the drawn image. The format is RGB.

Returns the drawn image which channel is RGB.

Return type np.ndarray

classmethod get_instance(*name*, ***kwargs*)
Make subclass can get latest created instance by `Visualizer.get_current_instance()`.

Downstream codebase may need to get the latest created instance without knowing the specific Visualizer type. For example, mmdetection builds visualizer in runner and some component which cannot access runner wants to get latest created visualizer. In this case, the component does not know which type of visualizer has been built and cannot get target instance. Therefore, *Visualizer* overrides the *get_instance()* and its subclass will register the created instance to `_instance_dict` additionally. `get_current_instance()` will return the latest created subclass instance.

Examples

```
>>> class DetLocalVisualizer(Visualizer):
>>>     def __init__(self, name):
>>>         super().__init__(name)
>>>
>>> visualizer1 = DetLocalVisualizer.get_instance('name1')
>>> visualizer2 = Visualizer.get_current_instance()
>>> visualizer3 = DetLocalVisualizer.get_current_instance()
>>> assert id(visualizer1) == id(visualizer2) == id(visualizer3)
```

Parameters **name** (*str*) – Name of instance.

Returns Corresponding name instance.

Return type object

set_image(*image*)
Set the image to draw.

Parameters **image** (*np.ndarray*) – The image to draw.

Return type None

show(*drawn_img=None*, *win_name='image'*, *wait_time=0*, *continue_key=' '*)
Show the drawn image.

Parameters

- **drawn_img** (*np.ndarray, optional*) – The image to show. If drawn_img is None, it will show the image got by Visualizer. Defaults to None.
- **win_name** (*str*) – The image title. Defaults to 'image'.
- **wait_time** (*int*) – Delay in milliseconds. 0 is the special value that means "forever". Defaults to 0.
- **continue_key** (*str*) – The key for users to continue. Defaults to the space key.

Return type None

48.2 visualization Backend

BaseVisBackend	Base class for visualization backend.
LocalVisBackend	Local visualization backend class.
TensorboardVisBackend	Tensorboard visualization backend class.
WandbVisBackend	Wandb visualization backend class.

48.2.1 BaseVisBackend

class mmengine.visualization.**BaseVisBackend**(*save_dir*)

Base class for visualization backend.

All backends must inherit `BaseVisBackend` and implement the required functions.

Parameters **save_dir** (*str, optional*) – The root directory to save the files produced by the backend.

add_config(*config*, ***kwargs*)

Record the config.

Parameters **config** (*Config*) – The Config object

Return type None

add_graph(*model*, *data_batch*, ***kwargs*)

Record the model graph.

Parameters

- **model** (*torch.nn.Module*) – Model to draw.
- **data_batch** (*Sequence[dict]*) – Batch of data from dataloader.

Return type None

add_image(*name*, *image*, *step=0*, ***kwargs*)

Record the image.

Parameters

- **name** (*str*) – The image identifier.
- **image** (*np.ndarray*) – The image to be saved. The format should be RGB. Default to None.
- **step** (*int*) – Global step value to record. Default to 0.

Return type None

add_scalar(*name*, *value*, *step=0*, ***kwargs*)

Record the scalar.

Parameters

- **name** (*str*) – The scalar identifier.
- **value** (*int, float*) – Value to save.
- **step** (*int*) – Global step value to record. Default to 0.

Return type None

add_scalars(*scalar_dict*, *step=0*, *file_path=None*, ***kwargs*)
Record the scalars' data.

Parameters

- **scalar_dict** (*dict*) – Key-value pair storing the tag and corresponding values.
- **step** (*int*) – Global step value to record. Default to 0.
- **file_path** (*str, optional*) – The scalar's data will be saved to the *file_path* file at the same time if the *file_path* parameter is specified. Default to None.

Return type None

close()
close an opened object.

Return type None

abstract property experiment: Any
Return the experiment object associated with this visualization backend.

The experiment attribute can get the visualization backend, such as wandb, tensorboard. If you want to write other data, such as writing a table, you can directly get the visualization backend through experiment.

48.2.2 LocalVisBackend

class mmengine.visualization.**LocalVisBackend**(*save_dir*, *img_save_dir='vis_image'*, *config_save_file='config.py'*, *scalar_save_file='scalars.json'*)
Local visualization backend class.

It can write image, config, scalars, etc. to the local hard disk. You can get the drawing backend through the experiment property for custom drawing.

Examples

```
>>> from mmengine.visualization import LocalVisBackend
>>> import numpy as np
>>> local_vis_backend = LocalVisBackend(save_dir='temp_dir')
>>> img = np.random.randint(0, 256, size=(10, 10, 3))
>>> local_vis_backend.add_image('img', img)
>>> local_vis_backend.add_scalar('mAP', 0.6)
>>> local_vis_backend.add_scalars({'loss': [1, 2, 3], 'acc': 0.8})
>>> cfg = Config(dict(a=1, b=dict(b1=[0, 1])))
>>> local_vis_backend.add_config(cfg)
```

Parameters

- **save_dir** (*str, optional*) – The root directory to save the files produced by the visualizer. If it is none, it means no data is stored.
- **img_save_dir** (*str*) – The directory to save images. Default to 'vis_image'.
- **config_save_file** (*str*) – The file name to save config. Default to 'config.py'.
- **scalar_save_file** (*str*) – The file name to save scalar values. Default to 'scalars.json'.

add_config(*config*, ***kwargs*)
Record the config to disk.

Parameters **config** (Config) – The Config object

Return type None

add_image(*name*, *image*, *step=0*, ***kwargs*)
Record the image to disk.

Parameters

- **name** (*str*) – The image identifier.
- **image** (*np.ndarray*) – The image to be saved. The format should be RGB. Default to None.
- **step** (*int*) – Global step value to record. Default to 0.

Return type None

add_scalar(*name*, *value*, *step=0*, ***kwargs*)
Record the scalar data to disk.

Parameters

- **name** (*str*) – The scalar identifier.
- **value** (*int*, *float*, *torch.Tensor*, *np.ndarray*) – Value to save.
- **step** (*int*) – Global step value to record. Default to 0.

Return type None

add_scalars(*scalar_dict*, *step=0*, *file_path=None*, ***kwargs*)
Record the scalars to disk.

The scalar dict will be written to the default and specified files if `file_path` is specified.

Parameters

- **scalar_dict** (*dict*) – Key-value pair storing the tag and corresponding values. The value must be dumped into json format.
- **step** (*int*) – Global step value to record. Default to 0.
- **file_path** (*str*, *optional*) – The scalar's data will be saved to the `file_path` file at the same time if the `file_path` parameter is specified. Default to None.

Return type None

property experiment: mmengine.visualization.vis_backend.LocalVisBackend
Return the experiment object associated with this visualization backend.

48.2.3 TensorboardVisBackend

class mmengine.visualization.**TensorboardVisBackend**(*save_dir*)
Tensorboard visualization backend class.

It can write images, config, scalars, etc. to a tensorboard file.

Examples

```
>>> from mmengine.visualization import TensorboardVisBackend
>>> import numpy as np
>>> tensorboard_vis_backend =          >>>     TensorboardVisBackend(save_dir='temp_
↪dir')
>>> img=np.random.randint(0, 256, size=(10, 10, 3))
>>> tensorboard_vis_backend.add_image('img', img)
>>> tensorboard_vis_backend.add_scaler('mAP', 0.6)
>>> tensorboard_vis_backend.add_scalars({'loss': 0.1,'acc':0.8})
>>> cfg = Config(dict(a=1, b=dict(b1=[0, 1])))
>>> tensorboard_vis_backend.add_config(cfg)
```

Parameters **save_dir** (*str*) – The root directory to save the files produced by the backend.

add_config(*config*, ***kwargs*)
Record the config to tensorboard.

Parameters **config** (Config) – The Config object

Return type None

add_image(*name*, *image*, *step=0*, ***kwargs*)
Record the image to tensorboard.

Parameters

- **name** (*str*) – The image identifier.
- **image** (*np.ndarray*) – The image to be saved. The format should be RGB.
- **step** (*int*) – Global step value to record. Default to 0.

Return type None

add_scalar(*name*, *value*, *step=0*, ***kwargs*)
Record the scalar data to tensorboard.

Parameters

- **name** (*str*) – The scalar identifier.
- **value** (*int*, *float*, *torch.Tensor*, *np.ndarray*) – Value to save.
- **step** (*int*) – Global step value to record. Default to 0.

Return type None

add_scalars(*scalar_dict*, *step=0*, *file_path=None*, ***kwargs*)
Record the scalar's data to tensorboard.

Parameters

- **scalar_dict** (*dict*) – Key-value pair storing the tag and corresponding values.
- **step** (*int*) – Global step value to record. Default to 0.
- **file_path** (*str*, *optional*) – Useless parameter. Just for interface unification. Default to None.

Return type None

close()
close an opened tensorboard object.

property experiment
Return Tensorboard object.

48.2.4 WandbVisBackend

class mmengine.visualization.**WandbVisBackend**(*save_dir*, *init_kwargs=None*, *define_metric_cfg=None*, *commit=True*, *log_code_name=None*)

Wandb visualization backend class.

Examples

```
>>> from mmengine.visualization import WandbVisBackend
>>> import numpy as np
>>> wandb_vis_backend = WandbVisBackend()
>>> img=np.random.randint(0, 256, size=(10, 10, 3))
>>> wandb_vis_backend.add_image('img', img)
>>> wandb_vis_backend.add_scaler('mAP', 0.6)
>>> wandb_vis_backend.add_scalars({'loss': [1, 2, 3],'acc': 0.8})
>>> cfg = Config(dict(a=1, b=dict(b1=[0, 1])))
>>> wandb_vis_backend.add_config(cfg)
```

Parameters

- **save_dir** (*str, optional*) – The root directory to save the files produced by the visualizer.
- **init_kwargs** (*dict, optional*) – wandb initialization input parameters. Default to None.
- **define_metric_cfg** (*dict, optional*) – A dict of metrics and summary for wandb.define_metric. The key is metric and the value is summary. When `define_metric_cfg={'coco/bbox_mAP': 'max'}`, The maximum value of``coco/bbox_mAP`` is logged on wandb UI. See wandb docs for details. Default: None
- **commit** (*Optional[bool]*) – (bool, optional) Save the metrics dict to the wandb server and increment the step. If false *wandb.log* just updates the current metrics dict with the row argument and metrics won't be saved until *wandb.log* is called with *commit=True*. Default to True.
- **log_code_name** (*Optional[str]*) – (str, optional) The name of code artifact. By default, the artifact will be named source-$PROJECT_ID-$ENTRYPOINT_RELPATH. See wandb docs for details. Defaults to None. New in version 0.3.0.

add_config(*config*, ***kwargs*)
Record the config to wandb.

Parameters **config** (Config) – The Config object

Return type None

add_image(*name*, *image*, *step=0*, ***kwargs*)
Record the image to wandb.

Parameters

- **name** (*str*) – The image identifier.

- **image** (*np.ndarray*) – The image to be saved. The format should be RGB.
- **step** (*int*) – Useless parameter. Wandb does not need this parameter. Default to 0.

Return type None

add_scalar(*name*, *value*, *step=0*, ***kwargs*)
Record the scalar data to wandb.

Parameters

- **name** (*str*) – The scalar identifier.
- **value** (*int, float, torch.Tensor, np.ndarray*) – Value to save.
- **step** (*int*) – Useless parameter. Wandb does not need this parameter. Default to 0.

Return type None

add_scalars(*scalar_dict*, *step=0*, *file_path=None*, ***kwargs*)
Record the scalar's data to wandb.

Parameters

- **scalar_dict** (*dict*) – Key-value pair storing the tag and corresponding values.
- **step** (*int*) – Useless parameter. Wandb does not need this parameter. Default to 0.
- **file_path** (*str, optional*) – Useless parameter. Just for interface unification. Default to None.

Return type None

close()
close an opened wandb object.

Return type None

property experiment
Return wandb object.

The experiment attribute can get the wandb backend, If you want to write other data, such as writing a table, you can directly get the wandb backend through experiment.

CHAPTER

FORTYNINE

MMENGINE.FILEIO

mmengine.fileio

- *File Backend*
- *File Handler*
- *File IO*
- *Parse File*

49.1 File Backend

BaseStorageBackend	Abstract class of storage backends.
FileClient	A general file client to access files in different backends.
HardDiskBackend	Raw hard disks storage backend.
LocalBackend	Raw local storage backend.
HTTPBackend	HTTP and HTTPS storage bachend.
LmdbBackend	Lmdb storage backend.
MemcachedBackend	Memcached storage backend.
PetrelBackend	Petrel storage backend (for internal usage).

49.1.1 BaseStorageBackend

class mmengine.fileio.**BaseStorageBackend**

Abstract class of storage backends.

All backends need to implement two apis: *get()* and *get_text()*.

- *get()* reads the file as a byte stream.
- *get_text()* reads the file as texts.

49.1.2 FileClient

class mmengine.fileio.**FileClient**(*backend=None*, *prefix=None*, ***kwargs*)

A general file client to access files in different backends.

The client loads a file or text in a specified backend from its path and returns it as a binary or text file. There are two ways to choose a backend, the name of backend and the prefix of path. Although both of them can be used to choose a storage backend, `backend` has a higher priority that is if they are all set, the storage backend will be chosen by the backend argument. If they are all *None*, the disk backend will be chosen. Note that It can also register other backend accessor with a given name, prefixes, and backend class. In addition, We use the singleton pattern to avoid repeated object creation. If the arguments are the same, the same object will be returned.

Warning: *FileClient* will be deprecated in future. Please use io functions in https://mmengine.readthedocs.io/en/latest/api/fileio.html#file-io

Parameters

- **backend** (*str, optional*) – The storage backend type. Options are "disk", "memcached", "lmdb", "http" and "petrel". Default: None.
- **prefix** (*str, optional*) – The prefix of the registered storage backend. Options are "s3", "http", "https". Default: None.

Examples

```
>>> # only set backend
>>> file_client = FileClient(backend='petrel')
>>> # only set prefix
>>> file_client = FileClient(prefix='s3')
>>> # set both backend and prefix but use backend to choose client
>>> file_client = FileClient(backend='petrel', prefix='s3')
>>> # if the arguments are the same, the same object is returned
>>> file_client1 = FileClient(backend='petrel')
>>> file_client1 is file_client
True
```

client

The backend object.

Type *BaseStorageBackend*

exists(*filepath*)

Check whether a file path exists.

Parameters **filepath** (*str or Path*) – Path to be checked whether exists.

Returns Return `True` if `filepath` exists, `False` otherwise.

Return type bool

get(*filepath*)

Read data from a given `filepath` with 'rb' mode.

Note: There are two types of return values for `get`, one is `bytes` and the other is `memoryview`. The advantage of using memoryview is that you can avoid copying, and if you want to convert it to `bytes`, you

can use `.tobytes()`.

Parameters **filepath** (*str or Path*) – Path to read data.

Returns Expected bytes object or a memory view of the bytes object.

Return type bytes | memoryview

get_local_path(*filepath*)

Download data from `filepath` and write the data to local path.

`get_local_path` is decorated by `contxtlib.contextmanager()`. It can be called with `with` statement, and when exists from the `with` statement, the temporary path will be released.

Note: If the `filepath` is a local path, just return itself.

Warning: `get_local_path` is an experimental interface that may change in the future.

Parameters **filepath** (*str or Path*) – Path to be read data.

Return type Generator[Union[str, pathlib.Path], None, None]

Examples

```
>>> file_client = FileClient(prefix='s3')
>>> with file_client.get_local_path('s3://bucket/abc.jpg') as path:
...     # do something here
```

Yields *Iterable[str]* – Only yield one path.

Parameters **filepath** (*Union[str, pathlib.Path]*) –

Return type Generator[Union[str, pathlib.Path], None, None]

get_text(*filepath*, *encoding='utf-8'*)

Read data from a given `filepath` with 'r' mode.

Parameters

- **filepath** (*str or Path*) – Path to read data.
- **encoding** (*str*) – The encoding format used to open the `filepath`. Default: 'utf-8'.

Returns Expected text reading from `filepath`.

Return type str

classmethod infer_client(*file_client_args=None*, *uri=None*)

Infer a suitable file client based on the URI and arguments.

Parameters

- **file_client_args** (*dict, optional*) – Arguments to instantiate a FileClient. Default: None.

- **uri** (*str | Path, optional*) – Uri to be parsed that contains the file prefix. Default: None.

Return type *mmengine.fileio.file_client.FileClient*

Examples

```
>>> uri = 's3://path/of/your/file'
>>> file_client = FileClient.infer_client(uri=uri)
>>> file_client_args = {'backend': 'petrel'}
>>> file_client = FileClient.infer_client(file_client_args)
```

Returns Instantiated FileClient object.

Return type *FileClient*

Parameters

- **file_client_args** (*Optional[dict]*) –
- **uri** (*Optional[Union[str, pathlib.Path]]*) –

isdir(*filepath*)
Check whether a file path is a directory.

Parameters **filepath** (*str or Path*) – Path to be checked whether it is a directory.

Returns Return `True` if `filepath` points to a directory, `False` otherwise.

Return type bool

isfile(*filepath*)
Check whether a file path is a file.

Parameters **filepath** (*str or Path*) – Path to be checked whether it is a file.

Returns Return `True` if `filepath` points to a file, `False` otherwise.

Return type bool

join_path(*filepath*, **filepaths*)
Concatenate all file paths.

Join one or more filepath components intelligently. The return value is the concatenation of filepath and any members of *filepaths.

Parameters

- **filepath** (*str or Path*) – Path to be concatenated.
- **filepaths** (*Union[str, pathlib.Path]*) –

Returns The result of concatenation.

Return type str

list_dir_or_file(*dir_path*, *list_dir=True*, *list_file=True*, *suffix=None*, *recursive=False*)
Scan a directory to find the interested directories or files in arbitrary order.

Note: *list_dir_or_file()* returns the path relative to `dir_path`.

Parameters

- **dir_path** (*str | Path*) – Path of the directory.
- **list_dir** (*bool*) – List the directories. Default: True.
- **list_file** (*bool*) – List the path of files. Default: True.
- **suffix** (*str or tuple[str], optional*) – File suffix that we are interested in. Default: None.
- **recursive** (*bool*) – If set to True, recursively scan the directory. Default: False.

Yields *Iterable[str]* – A relative path to `dir_path`.

Return type Iterator[str]

static parse_uri_prefix(*uri*)

Parse the prefix of a uri.

Parameters **uri** (*str | Path*) – Uri to be parsed that contains the file prefix.

Return type Optional[str]

Examples

```
>>> FileClient.parse_uri_prefix('s3://path/of/your/file')
's3'
```

Returns Return the prefix of uri if the uri contains '://' else `None`.

Return type str | None

Parameters **uri** (*Union[str, pathlib.Path]*) –

put(*obj*, *filepath*)

Write data to a given `filepath` with 'wb' mode.

Note: `put` should create a directory if the directory of `filepath` does not exist.

Parameters

- **obj** (*bytes*) – Data to be written.
- **filepath** (*str or Path*) – Path to write data.

Return type None

put_text(*obj*, *filepath*)

Write data to a given `filepath` with 'w' mode.

Note: `put_text` should create a directory if the directory of `filepath` does not exist.

Parameters

- **obj** (*str*) – Data to be written.

- **filepath** (*str or Path*) – Path to write data.
- **encoding** (*str, optional*) – The encoding format used to open the *filepath*. Default: 'utf-8'.

Return type None

classmethod register_backend(*name*, *backend=None*, *force=False*, *prefixes=None*)

Register a backend to FileClient.

This method can be used as a normal class method or a decorator.

```
class NewBackend(BaseStorageBackend):

    def get(self, filepath):
        return filepath

    def get_text(self, filepath):
        return filepath

FileClient.register_backend('new', NewBackend)
```

or

```
@FileClient.register_backend('new')
class NewBackend(BaseStorageBackend):

    def get(self, filepath):
        return filepath

    def get_text(self, filepath):
        return filepath
```

Parameters

- **name** (*str*) – The name of the registered backend.
- **backend** (*class, optional*) – The backend class to be registered, which must be a subclass of *BaseStorageBackend*. When this method is used as a decorator, backend is None. Defaults to None.
- **force** (*bool, optional*) – Whether to override the backend if the name has already been registered. Defaults to False.
- **prefixes** (*str or list[str] or tuple[str], optional*) – The prefixes of the registered storage backend. Default: None. *New in version 1.3.15.*

remove(*filepath*)

Remove a file.

Parameters **filepath** (*str, Path*) – Path to be removed.

Return type None

49.1.3 HardDiskBackend

class mmengine.fileio.**HardDiskBackend**
Raw hard disks storage backend.

Return type None

49.1.4 LocalBackend

class mmengine.fileio.**LocalBackend**
Raw local storage backend.

copy_if_symlink_fails(*src*, *dst*)
Create a symbolic link pointing to src named dst.

If failed to create a symbolic link pointing to src, directly copy src to dst instead.

Parameters

- **src** (*str or Path*) – Create a symbolic link pointing to src.
- **dst** (*str or Path*) – Create a symbolic link named dst.

Returns Return True if successfully create a symbolic link pointing to src. Otherwise, return False.

Return type bool

Examples

```
>>> backend = LocalBackend()
>>> src = '/path/of/file'
>>> dst = '/path1/of/file1'
>>> backend.copy_if_symlink_fails(src, dst)
True
>>> src = '/path/of/dir'
>>> dst = '/path1/of/dir1'
>>> backend.copy_if_symlink_fails(src, dst)
True
```

copyfile(*src*, *dst*)
Copy a file src to dst and return the destination file.

src and dst should have the same prefix. If dst specifies a directory, the file will be copied into dst using the base filename from src. If dst specifies a file that already exists, it will be replaced.

Parameters

- **src** (*str or Path*) – A file to be copied.
- **dst** (*str or Path*) – Copy file to dst.

Returns The destination file.

Return type str

Raises **SameFileError** – If src and dst are the same file, a SameFileError will be raised.

Examples

```
>>> backend = LocalBackend()
>>> # dst is a file
>>> src = '/path/of/file'
>>> dst = '/path1/of/file1'
>>> # src will be copied to '/path1/of/file1'
>>> backend.copyfile(src, dst)
'/path1/of/file1'
```

```
>>> # dst is a directory
>>> dst = '/path1/of/dir'
>>> # src will be copied to '/path1/of/dir/file'
>>> backend.copyfile(src, dst)
'/path1/of/dir/file'
```

copyfile_from_local(*src*, *dst*)

Copy a local file src to dst and return the destination file. Same as *copyfile()*.

Parameters

- **src** (*str or Path*) – A local file to be copied.
- **dst** (*str or Path*) – Copy file to dst.

Returns If dst specifies a directory, the file will be copied into dst using the base filename from src.

Return type str

Raises **SameFileError** – If src and dst are the same file, a SameFileError will be raised.

Examples

```
>>> backend = LocalBackend()
>>> # dst is a file
>>> src = '/path/of/file'
>>> dst = '/path1/of/file1'
>>> # src will be copied to '/path1/of/file1'
>>> backend.copyfile_from_local(src, dst)
'/path1/of/file1'
```

```
>>> # dst is a directory
>>> dst = '/path1/of/dir'
>>> # src will be copied to
>>> backend.copyfile_from_local(src, dst)
'/path1/of/dir/file'
```

copyfile_to_local(*src*, *dst*)

Copy the file src to local dst and return the destination file. Same as *copyfile()*.

If dst specifies a directory, the file will be copied into dst using the base filename from src. If dst specifies a file that already exists, it will be replaced.

Parameters

- **src** (*str or Path*) – A file to be copied.

- **dst** (*str or Path*) – Copy file to to local dst.

Returns If dst specifies a directory, the file will be copied into dst using the base filename from src.

Return type str

Examples

```
>>> backend = LocalBackend()
>>> # dst is a file
>>> src = '/path/of/file'
>>> dst = '/path1/of/file1'
>>> # src will be copied to '/path1/of/file1'
>>> backend.copyfile_to_local(src, dst)
'/path1/of/file1'
```

```
>>> # dst is a directory
>>> dst = '/path1/of/dir'
>>> # src will be copied to
>>> backend.copyfile_to_local(src, dst)
'/path1/of/dir/file'
```

copytree(*src*, *dst*)

Recursively copy an entire directory tree rooted at src to a directory named dst and return the destination directory.

src and dst should have the same prefix and dst must not already exist.

TODO: Whether to support dirs_exist_ok parameter.

Parameters

- **src** (*str or Path*) – A directory to be copied.
- **dst** (*str or Path*) – Copy directory to dst.

Returns The destination directory.

Return type str

Raises **FileExistsError** – If dst had already existed, a FileExistsError will be raised.

Examples

```
>>> backend = LocalBackend()
>>> src = '/path/of/dir1'
>>> dst = '/path/of/dir2'
>>> backend.copytree(src, dst)
'/path/of/dir2'
```

copytree_from_local(*src*, *dst*)

Recursively copy an entire directory tree rooted at src to a directory named dst and return the destination directory. Same as *copytree()*.

Parameters

- **src** (*str or Path*) – A local directory to be copied.

- **dst** (*str or Path*) – Copy directory to dst.

Returns The destination directory.

Return type str

Examples

```
>>> backend = LocalBackend()
>>> src = '/path/of/dir1'
>>> dst = '/path/of/dir2'
>>> backend.copytree_from_local(src, dst)
'/path/of/dir2'
```

copytree_to_local(*src*, *dst*)

Recursively copy an entire directory tree rooted at src to a local directory named dst and return the destination directory.

Parameters

- **src** (*str or Path*) – A directory to be copied.
- **dst** (*str or Path*) – Copy directory to local dst.
- **backend_args** (*dict, optional*) – Arguments to instantiate the preifx of uri corresponding backend. Defaults to None.

Returns The destination directory.

Return type str

Examples

```
>>> backend = LocalBackend()
>>> src = '/path/of/dir1'
>>> dst = '/path/of/dir2'
>>> backend.copytree_from_local(src, dst)
'/path/of/dir2'
```

exists(*filepath*)

Check whether a file path exists.

Parameters **filepath** (*str or Path*) – Path to be checked whether exists.

Returns Return `True` if `filepath` exists, `False` otherwise.

Return type bool

Examples

```
>>> backend = LocalBackend()
>>> filepath = '/path/of/file'
>>> backend.exists(filepath)
True
```

get(*filepath*)

Read bytes from a given `filepath` with 'rb' mode.

Parameters **filepath** (*str or Path*) – Path to read data.

Returns Expected bytes object.

Return type bytes

Examples

```
>>> backend = LocalBackend()
>>> filepath = '/path/of/file'
>>> backend.get(filepath)
b'hello world'
```

get_local_path(*filepath*)

Only for unified API and do nothing.

Parameters

- **filepath** (*str or Path*) – Path to be read data.
- **backend_args** (*dict, optional*) – Arguments to instantiate the corresponding backend. Defaults to None.

Return type Generator[Union[str, pathlib.Path], None, None]

Examples

```
>>> backend = LocalBackend()
>>> with backend.get_local_path('s3://bucket/abc.jpg') as path:
...     # do something here
```

get_text(*filepath*, *encoding='utf-8'*)

Read text from a given `filepath` with 'r' mode.

Parameters

- **filepath** (*str or Path*) – Path to read data.
- **encoding** (*str*) – The encoding format used to open the `filepath`. Defaults to 'utf-8'.

Returns Expected text reading from `filepath`.

Return type str

Examples

```
>>> backend = LocalBackend()
>>> filepath = '/path/of/file'
>>> backend.get_text(filepath)
'hello world'
```

isdir(*filepath*)

Check whether a file path is a directory.

Parameters **filepath** (*str or Path*) – Path to be checked whether it is a directory.

Returns Return `True` if `filepath` points to a directory, `False` otherwise.

Return type bool

Examples

```
>>> backend = LocalBackend()
>>> filepath = '/path/of/dir'
>>> backend.isdir(filepath)
True
```

isfile(*filepath*)

Check whether a file path is a file.

Parameters **filepath** (*str or Path*) – Path to be checked whether it is a file.

Returns Return `True` if `filepath` points to a file, `False` otherwise.

Return type bool

Examples

```
>>> backend = LocalBackend()
>>> filepath = '/path/of/file'
>>> backend.isfile(filepath)
True
```

join_path(*filepath*, **filepaths*)

Concatenate all file paths.

Join one or more filepath components intelligently. The return value is the concatenation of filepath and any members of *filepaths.

Parameters

- **filepath** (*str or Path*) – Path to be concatenated.
- **filepaths** (*Union[str, pathlib.Path]*) –

Returns The result of concatenation.

Return type str

Examples

```
>>> backend = LocalBackend()
>>> filepath1 = '/path/of/dir1'
>>> filepath2 = 'dir2'
>>> filepath3 = 'path/of/file'
>>> backend.join_path(filepath1, filepath2, filepath3)
'/path/of/dir/dir2/path/of/file'
```

list_dir_or_file(*dir_path*, *list_dir=True*, *list_file=True*, *suffix=None*, *recursive=False*)

Scan a directory to find the interested directories or files in arbitrary order.

Note: *list_dir_or_file()* returns the path relative to `dir_path`.

Parameters

- **dir_path** (*str or Path*) – Path of the directory.
- **list_dir** (*bool*) – List the directories. Defaults to True.
- **list_file** (*bool*) – List the path of files. Defaults to True.
- **suffix** (*str or tuple[str], optional*) – File suffix that we are interested in. Defaults to None.
- **recursive** (*bool*) – If set to True, recursively scan the directory. Defaults to False.

Yields *Iterable[str]* – A relative path to `dir_path`.

Return type Iterator[str]

Examples

```
>>> backend = LocalBackend()
>>> dir_path = '/path/of/dir'
>>> # list those files and directories in current directory
>>> for file_path in backend.list_dir_or_file(dir_path):
...     print(file_path)
>>> # only list files
>>> for file_path in backend.list_dir_or_file(dir_path, list_dir=False):
...     print(file_path)
>>> # only list directories
>>> for file_path in backend.list_dir_or_file(dir_path, list_file=False):
...     print(file_path)
>>> # only list files ending with specified suffixes
>>> for file_path in backend.list_dir_or_file(dir_path, suffix='.txt'):
...     print(file_path)
>>> # list all files and directory recursively
>>> for file_path in backend.list_dir_or_file(dir_path, recursive=True):
...     print(file_path)
```

put(*obj*, *filepath*)

Write bytes to a given `filepath` with 'wb' mode.

Note: `put` will create a directory if the directory of `filepath` does not exist.

Parameters

- **obj** (*bytes*) – Data to be written.
- **filepath** (*str or Path*) – Path to write data.

Return type None

Examples

```
>>> backend = LocalBackend()
>>> filepath = '/path/of/file'
>>> backend.put(b'hello world', filepath)
```

put_text(*obj*, *filepath*, *encoding='utf-8'*)

Write text to a given `filepath` with 'w' mode.

Note: `put_text` will create a directory if the directory of `filepath` does not exist.

Parameters

- **obj** (*str*) – Data to be written.
- **filepath** (*str or Path*) – Path to write data.
- **encoding** (*str*) – The encoding format used to open the `filepath`. Defaults to 'utf-8'.

Return type None

Examples

```
>>> backend = LocalBackend()
>>> filepath = '/path/of/file'
>>> backend.put_text('hello world', filepath)
```

remove(*filepath*)

Remove a file.

Parameters **filepath** (*str or Path*) – Path to be removed.

Raises

- **IsADirectoryError** – If filepath is a directory, an IsADirectoryError will be raised.
- **FileNotFoundError** – If filepath does not exist, an FileNotFoundError will be raised.

Return type None

Examples

```
>>> backend = LocalBackend()
>>> filepath = '/path/of/file'
>>> backend.remove(filepath)
```

rmtree(*dir_path*)
Recursively delete a directory tree.

Parameters **dir_path** (*str or Path*) – A directory to be removed.

Return type None

Examples

```
>>> dir_path = '/path/of/dir'
>>> backend.rmtree(dir_path)
```

49.1.5 HTTPBackend

class mmengine.fileio.**HTTPBackend**
HTTP and HTTPS storage bachend.

get(*filepath*)
Read bytes from a given `filepath`.

Parameters **filepath** (*str*) – Path to read data.

Returns Expected bytes object.

Return type bytes

Examples

```
>>> backend = HTTPBackend()
>>> backend.get('http://path/of/file')
b'hello world'
```

get_local_path(*filepath*)
Download a file from `filepath` to a local temporary directory, and return the temporary path.

`get_local_path` is decorated by `contxtlib.contextmanager()`. It can be called with `with` statement, and when exists from the `with` statement, the temporary path will be released.

Parameters **filepath** (*str*) – Download a file from `filepath`.

Yields *Iterable[str]* – Only yield one temporary path.

Return type Generator[Union[str, pathlib.Path], None, None]

Examples

```
>>> backend = HTTPBackend()
>>> # After existing from the ``with`` clause,
>>> # the path will be removed
>>> with backend.get_local_path('http://path/of/file') as path:
...     # do something here
```

get_text(*filepath*, *encoding='utf-8'*)
Read text from a given `filepath`.

Parameters

- **filepath** (*str*) – Path to read data.
- **encoding** (*str*) – The encoding format used to open the `filepath`. Defaults to 'utf-8'.

Returns Expected text reading from `filepath`.

Return type str

Examples

```
>>> backend = HTTPBackend()
>>> backend.get_text('http://path/of/file')
'hello world'
```

49.1.6 LmdbBackend

class mmengine.fileio.**LmdbBackend**(*db_path*, *readonly=True*, *lock=False*, *readahead=False*, ***kwargs*)
Lmdb storage backend.

Parameters

- **db_path** (*str*) – Lmdb database path.
- **readonly** (*bool*) – Lmdb environment parameter. If True, disallow any write operations. Defaults to True.
- **lock** (*bool*) – Lmdb environment parameter. If False, when concurrent access occurs, do not lock the database. Defaults to False.
- **readahead** (*bool*) – Lmdb environment parameter. If False, disable the OS filesystem readahead mechanism, which may improve random read performance when a database is larger than RAM. Defaults to False.
- ****kwargs** – Keyword arguments passed to *lmdb.open*.

db_path
Lmdb database path.

Type str

get(*filepath*)
Get values according to the filepath.

Parameters **filepath** (*str or Path*) – Here, filepath is the lmdb key.

Returns Expected bytes object.

Return type bytes

Examples

```
>>> backend = LmdbBackend('path/to/lmdb')
>>> backend.get('key')
b'hello world'
```

49.1.7 MemcachedBackend

class mmengine.fileio.**MemcachedBackend**(*server_list_cfg*, *client_cfg*, *sys_path=None*)

Memcached storage backend.

server_list_cfg

Config file for memcached server list.

Type str

client_cfg

Config file for memcached client.

Type str

sys_path

Additional path to be appended to *sys.path*. Defaults to None.

Type str, optional

get(*filepath*)

Get values according to the filepath.

Parameters **filepath** (*str or Path*) – Path to read data.

Returns Expected bytes object.

Return type bytes

Examples

```
>>> server_list_cfg = '/path/of/server_list.conf'
>>> client_cfg = '/path/of/mc.conf'
>>> backend = MemcachedBackend(server_list_cfg, client_cfg)
>>> backend.get('/path/of/file')
b'hello world'
```

49.1.8 PetrelBackend

class mmengine.fileio.**PetrelBackend**(*path_mapping=None*, *enable_mc=True*)

Petrel storage backend (for internal usage).

PetrelBackend supports reading and writing data to multiple clusters. If the file path contains the cluster name, PetrelBackend will read data from specified cluster or write data to it. Otherwise, PetrelBackend will access the default cluster.

Parameters

- **path_mapping** (*dict, optional*) – Path mapping dict from local path to Petrel path. When `path_mapping={'src': 'dst'}`, `src` in `filepath` will be replaced by `dst`. Defaults to None.
- **enable_mc** (*bool, optional*) – Whether to enable memcached support. Defaults to True.

Examples

```
>>> backend = PetrelBackend()
>>> filepath1 = 'petrel://path/of/file'
>>> filepath2 = 'cluster-name:petrel://path/of/file'
>>> backend.get(filepath1)  # get data from default cluster
>>> client.get(filepath2)  # get data from 'cluster-name' cluster
```

copy_if_symlink_fails(*src*, *dst*)

Create a symbolic link pointing to src named dst.

Directly copy src to dst because PetrelBacekend does not support create a symbolic link.

Parameters

- **src** (*str or Path*) – A file or directory to be copied.
- **dst** (*str or Path*) – Copy a file or directory to dst.
- **backend_args** (*dict, optional*) – Arguments to instantiate the preifx of uri corresponding backend. Defaults to None.

Returns Return False because PetrelBackend does not support create a symbolic link.

Return type bool

Examples

```
>>> backend = PetrelBackend()
>>> src = 'petrel://path/of/file'
>>> dst = 'petrel://path/of/your/file'
>>> backend.copy_if_symlink_fails(src, dst)
False
>>> src = 'petrel://path/of/dir'
>>> dst = 'petrel://path/of/your/dir'
>>> backend.copy_if_symlink_fails(src, dst)
False
```

copyfile(*src*, *dst*)

Copy a file src to dst and return the destination file.

src and dst should have the same prefix. If dst specifies a directory, the file will be copied into dst using the base filename from src. If dst specifies a file that already exists, it will be replaced.

Parameters

- **src** (*str or Path*) – A file to be copied.
- **dst** (*str or Path*) – Copy file to dst.

Returns The destination file.

Return type str

Raises **SameFileError** – If src and dst are the same file, a SameFileError will be raised.

Examples

```
>>> backend = PetrelBackend()
>>> # dst is a file
>>> src = 'petrel://path/of/file'
>>> dst = 'petrel://path/of/file1'
>>> backend.copyfile(src, dst)
'petrel://path/of/file1'
```

```
>>> # dst is a directory
>>> dst = 'petrel://path/of/dir'
>>> backend.copyfile(src, dst)
'petrel://path/of/dir/file'
```

copyfile_from_local(*src*, *dst*)

Upload a local file src to dst and return the destination file.

Parameters

- **src** (*str or Path*) – A local file to be copied.
- **dst** (*str or Path*) – Copy file to dst.
- **backend_args** (*dict, optional*) – Arguments to instantiate the preifx of uri corresponding backend. Defaults to None.

Returns If dst specifies a directory, the file will be copied into dst using the base filename from src.

Return type str

Examples

```
>>> backend = PetrelBackend()
>>> # dst is a file
>>> src = 'path/of/your/file'
>>> dst = 'petrel://path/of/file1'
>>> backend.copyfile_from_local(src, dst)
'petrel://path/of/file1'
```

```
>>> # dst is a directory
>>> dst = 'petrel://path/of/dir'
>>> backend.copyfile_from_local(src, dst)
'petrel://path/of/dir/file'
```

copyfile_to_local(*src*, *dst*)

Copy the file src to local dst and return the destination file.

If dst specifies a directory, the file will be copied into dst using the base filename from src. If dst specifies a file that already exists, it will be replaced.

Parameters

- **src** (*str or Path*) – A file to be copied.
- **dst** (*str or Path*) – Copy file to to local dst.

Returns If dst specifies a directory, the file will be copied into dst using the base filename from src.

Return type str

Examples

```
>>> backend = PetrelBackend()
>>> # dst is a file
>>> src = 'petrel://path/of/file'
>>> dst = 'path/of/your/file'
>>> backend.copyfile_to_local(src, dst)
'path/of/your/file'
```

```
>>> # dst is a directory
>>> dst = 'path/of/your/dir'
>>> backend.copyfile_to_local(src, dst)
'path/of/your/dir/file'
```

copytree(*src*, *dst*)

Recursively copy an entire directory tree rooted at src to a directory named dst and return the destination directory.

src and dst should have the same prefix.

Parameters

- **src** (*str or Path*) – A directory to be copied.
- **dst** (*str or Path*) – Copy directory to dst.
- **backend_args** (*dict, optional*) – Arguments to instantiate the preifx of uri corresponding backend. Defaults to None.

Returns The destination directory.

Return type str

Raises **FileExistsError** – If dst had already existed, a FileExistsError will be raised.

Examples

```
>>> backend = PetrelBackend()
>>> src = 'petrel://path/of/dir'
>>> dst = 'petrel://path/of/dir1'
>>> backend.copytree(src, dst)
'petrel://path/of/dir1'
```

copytree_from_local(*src*, *dst*)

Recursively copy an entire directory tree rooted at src to a directory named dst and return the destination directory.

Parameters

- **src** (*str or Path*) – A local directory to be copied.
- **dst** (*str or Path*) – Copy directory to dst.

Returns The destination directory.

Return type str

Raises **FileExistsError** – If dst had already existed, a FileExistsError will be raised.

Examples

```
>>> backend = PetrelBackend()
>>> src = 'path/of/your/dir'
>>> dst = 'petrel://path/of/dir1'
>>> backend.copytree_from_local(src, dst)
'petrel://path/of/dir1'
```

copytree_to_local(*src*, *dst*)

Recursively copy an entire directory tree rooted at src to a local directory named dst and return the destination directory.

Parameters

- **src** (*str or Path*) – A directory to be copied.
- **dst** (*str or Path*) – Copy directory to local dst.
- **backend_args** (*dict, optional*) – Arguments to instantiate the preifx of uri corresponding backend. Defaults to None.

Returns The destination directory.

Return type str

Examples

```
>>> backend = PetrelBackend()
>>> src = 'petrel://path/of/dir'
>>> dst = 'path/of/your/dir'
>>> backend.copytree_to_local(src, dst)
'path/of/your/dir'
```

exists(*filepath*)

Check whether a file path exists.

Parameters **filepath** (*str or Path*) – Path to be checked whether exists.

Returns Return `True` if `filepath` exists, `False` otherwise.

Return type bool

Examples

```
>>> backend = PetrelBackend()
>>> filepath = 'petrel://path/of/file'
>>> backend.exists(filepath)
True
```

generate_presigned_url(*url*, *client_method='get_object'*, *expires_in=3600*)

Generate the presigned url of video stream which can be passed to mmcv.VideoReader. Now only work on Petrel backend.

Note: Now only work on Petrel backend.

Parameters

- **url** (*str*) – Url of video stream.
- **client_method** (*str*) – Method of client, 'get_object' or 'put_object'. Default: 'get_object'.
- **expires_in** (*int*) – expires, in seconds. Default: 3600.

Returns Generated presigned url.

Return type str

get(*filepath*)

Read bytes from a given `filepath` with 'rb' mode.

Parameters **filepath** (*str or Path*) – Path to read data.

Returns Return bytes read from filepath.

Return type bytes

Examples

```
>>> backend = PetrelBackend()
>>> filepath = 'petrel://path/of/file'
>>> backend.get(filepath)
b'hello world'
```

get_local_path(*filepath*)

Download a file from `filepath` to a local temporary directory, and return the temporary path.

`get_local_path` is decorated by `contxtlib.contextmanager()`. It can be called with `with` statement, and when exists from the `with` statement, the temporary path will be released.

Parameters **filepath** (*str or Path*) – Download a file from `filepath`.

Yields *Iterable[str]* – Only yield one temporary path.

Return type Generator[Union[str, pathlib.Path], None, None]

Examples

```
>>> backend = PetrelBackend()
>>> # After existing from the ``with`` clause,
>>> # the path will be removed
>>> filepath = 'petrel://path/of/file'
>>> with backend.get_local_path(filepath) as path:
...     # do something here
```

get_text(*filepath*, *encoding='utf-8'*)

Read text from a given `filepath` with 'r' mode.

Parameters

- **filepath** (*str or Path*) – Path to read data.
- **encoding** (*str*) – The encoding format used to open the `filepath`. Defaults to 'utf-8'.

Returns Expected text reading from `filepath`.

Return type str

Examples

```
>>> backend = PetrelBackend()
>>> filepath = 'petrel://path/of/file'
>>> backend.get_text(filepath)
'hello world'
```

isdir(*filepath*)

Check whether a file path is a directory.

Parameters **filepath** (*str or Path*) – Path to be checked whether it is a directory.

Returns Return `True` if `filepath` points to a directory, `False` otherwise.

Return type bool

Examples

```
>>> backend = PetrelBackend()
>>> filepath = 'petrel://path/of/dir'
>>> backend.isdir(filepath)
True
```

isfile(*filepath*)

Check whether a file path is a file.

Parameters **filepath** (*str or Path*) – Path to be checked whether it is a file.

Returns Return `True` if `filepath` points to a file, `False` otherwise.

Return type bool

Examples

```
>>> backend = PetrelBackend()
>>> filepath = 'petrel://path/of/file'
>>> backend.isfile(filepath)
True
```

join_path(*filepath*, **filepaths*)

Concatenate all file paths.

Join one or more filepath components intelligently. The return value is the concatenation of filepath and any members of *filepaths.

Parameters

- **filepath** (*str or Path*) – Path to be concatenated.
- **filepaths** (*Union[str, pathlib.Path]*) –

Returns The result after concatenation.

Return type str

Examples

```
>>> backend = PetrelBackend()
>>> filepath = 'petrel://path/of/file'
>>> backend.join_path(filepath, 'another/path')
'petrel://path/of/file/another/path'
>>> backend.join_path(filepath, '/another/path')
'petrel://path/of/file/another/path'
```

list_dir_or_file(*dir_path*, *list_dir=True*, *list_file=True*, *suffix=None*, *recursive=False*)

Scan a directory to find the interested directories or files in arbitrary order.

Note: Petrel has no concept of directories but it simulates the directory hierarchy in the filesystem through public prefixes. In addition, if the returned path ends with '/', it means the path is a public prefix which is a logical directory.

Note: *list_dir_or_file()* returns the path relative to `dir_path`. In addition, the returned path of directory will not contains the suffix '/' which is consistent with other backends.

Parameters

- **dir_path** (*str | Path*) – Path of the directory.
- **list_dir** (*bool*) – List the directories. Defaults to True.
- **list_file** (*bool*) – List the path of files. Defaults to True.
- **suffix** (*str or tuple[str], optional*) – File suffix that we are interested in. Defaults to None.
- **recursive** (*bool*) – If set to True, recursively scan the directory. Defaults to False.

Yields *Iterable[str]* – A relative path to `dir_path`.

Return type Iterator[str]

Examples

```
>>> backend = PetrelBackend()
>>> dir_path = 'petrel://path/of/dir'
>>> # list those files and directories in current directory
>>> for file_path in backend.list_dir_or_file(dir_path):
...     print(file_path)
>>> # only list files
>>> for file_path in backend.list_dir_or_file(dir_path, list_dir=False):
...     print(file_path)
>>> # only list directories
>>> for file_path in backend.list_dir_or_file(dir_path, list_file=False):
...     print(file_path)
>>> # only list files ending with specified suffixes
>>> for file_path in backend.list_dir_or_file(dir_path, suffix='.txt'):
...     print(file_path)
>>> # list all files and directory recursively
>>> for file_path in backend.list_dir_or_file(dir_path, recursive=True):
...     print(file_path)
```

put(*obj*, *filepath*)

Write bytes to a given `filepath`.

Parameters

- **obj** (*bytes*) – Data to be saved.
- **filepath** (*str or Path*) – Path to write data.

Return type None

Examples

```
>>> backend = PetrelBackend()
>>> filepath = 'petrel://path/of/file'
>>> backend.put(b'hello world', filepath)
```

put_text(*obj*, *filepath*, *encoding='utf-8'*)
Write text to a given `filepath`.

Parameters

- **obj** (*str*) – Data to be written.
- **filepath** (*str or Path*) – Path to write data.
- **encoding** (*str*) – The encoding format used to encode the `obj`. Defaults to 'utf-8'.

Return type None

Examples

```
>>> backend = PetrelBackend()
>>> filepath = 'petrel://path/of/file'
>>> backend.put_text('hello world', filepath)
```

remove(*filepath*)
Remove a file.

Parameters **filepath** (*str or Path*) – Path to be removed.

Raises

- **FileNotFoundError** – If filepath does not exist, an FileNotFoundError will be raised.
- **IsADirectoryError** – If filepath is a directory, an IsADirectoryError will be raised.

Return type None

Examples

```
>>> backend = PetrelBackend()
>>> filepath = 'petrel://path/of/file'
>>> backend.remove(filepath)
```

rmtree(*dir_path*)
Recursively delete a directory tree.

Parameters **dir_path** (*str or Path*) – A directory to be removed.

Return type None

Examples

```
>>> backend = PetrelBackend()
>>> dir_path = 'petrel://path/of/dir'
>>> backend.rmtree(dir_path)
```

register_backend	Register a backend.

49.1.9 mmengine.fileio.register_backend

mmengine.fileio.**register_backend**(*name*, *backend=None*, *force=False*, *prefixes=None*)

Register a backend.

Parameters

- **name** (*str*) – The name of the registered backend.
- **backend** (*class, optional*) – The backend class to be registered, which must be a subclass of *BaseStorageBackend*. When this method is used as a decorator, backend is None. Defaults to None.
- **force** (*bool*) – Whether to override the backend if the name has already been registered. Defaults to False.
- **prefixes** (*str or list[str] or tuple[str], optional*) – The prefix of the registered storage backend. Defaults to None.

This method can be used as a normal method or a decorator.

Examples

```
>>> class NewBackend(BaseStorageBackend):
...     def get(self, filepath):
...         return filepath
...
...     def get_text(self, filepath):
...         return filepath
>>> register_backend('new', NewBackend)
```

```
>>> @register_backend('new')
... class NewBackend(BaseStorageBackend):
...     def get(self, filepath):
...         return filepath
...
...     def get_text(self, filepath):
...         return filepath
```

49.2 File Handler

BaseFileHandler
JsonHandler
PickleHandler
YamlHandler

49.2.1 BaseFileHandler

class mmengine.fileio.**BaseFileHandler**

49.2.2 JsonHandler

class mmengine.fileio.**JsonHandler**

49.2.3 PickleHandler

class mmengine.fileio.**PickleHandler**

49.2.4 YamlHandler

class mmengine.fileio.**YamlHandler**

register_handler

49.2.5 mmengine.fileio.register_handler

mmengine.fileio.**register_handler**(*file_formats*, ***kwargs*)

49.3 File IO

dump	Dump data to json/yaml/pickle strings or files.
load	Load data from json/yaml/pickle files.
copy_if_symlink_fails	Create a symbolic link pointing to src named dst.
copyfile	Copy a file src to dst and return the destination file.
copyfile_from_local	Copy a local file src to dst and return the destination file.
copyfile_to_local	Copy the file src to local dst and return the destination file.

continues on next page

Table 5 – continued from previous page

copytree	Recursively copy an entire directory tree rooted at src to a directory named dst and return the destination directory.
copytree_from_local	Recursively copy an entire directory tree rooted at src to a directory named dst and return the destination directory.
copytree_to_local	Recursively copy an entire directory tree rooted at src to a local directory named dst and return the destination directory.
exists	Check whether a file path exists.
generate_presigned_url	Generate the presigned url of video stream which can be passed to mmcv.VideoReader.
get	Read bytes from a given `filepath` with 'rb' mode.
get_file_backend	Return a file backend based on the prefix of uri or backend_args.
get_local_path	Download data from `filepath` and write the data to local path.
get_text	Read text from a given `filepath` with 'r' mode.
isdir	Check whether a file path is a directory.
isfile	Check whether a file path is a file.
join_path	Concatenate all file paths.
list_dir_or_file	Scan a directory to find the interested directories or files in arbitrary order.
put	Write bytes to a given `filepath` with 'wb' mode.
put_text	Write text to a given `filepath` with 'w' mode.
remove	Remove a file.
rmtree	Recursively delete a directory tree.

49.3.1 mmengine.fileio.dump

mmengine.fileio.**dump**(*obj*, *file=None*, *file_format=None*, *file_client_args=None*, *backend_args=None*, ***kwargs*)

Dump data to json/yaml/pickle strings or files.

This method provides a unified api for dumping data as strings or to files, and also supports custom arguments for each file format.

`dump` supports dumping data as strings or to files which is saved to different backends.

Parameters

- **obj** (*any*) – The python object to be dumped.
- **file** (str or `Path` or file-like object, optional) – If not specified, then the object is dumped to a str, otherwise to a file specified by the filename or file-like object.
- **file_format** (*str, optional*) – Same as *load()*.
- **file_client_args** (*dict, optional*) – Arguments to instantiate a FileClient. See *mmengine.fileio.FileClient* for details. Defaults to None. It will be deprecated in future. Please use `backend_args` instead.
- **backend_args** (*dict, optional*) – Arguments to instantiate the preifx of uri corresponding backend. Defaults to None. New in v0.2.0.

Examples

```
>>> dump('hello world', '/path/of/your/file')  # disk
>>> dump('hello world', 's3://path/of/your/file')  # ceph or petrel
```

Returns True for success, False otherwise.

Return type bool

49.3.2 mmengine.fileio.load

mmengine.fileio.**load**(*file*, *file_format=None*, *file_client_args=None*, *backend_args=None*, ***kwargs*)

Load data from json/yaml/pickle files.

This method provides a unified api for loading data from serialized files.

`load` supports loading data from serialized files those can be storaged in different backends.

Parameters

- **file** (str or `Path` or file-like object) – Filename or a file-like object.
- **file_format** (*str, optional*) – If not specified, the file format will be inferred from the file extension, otherwise use the specified one. Currently supported formats include "json", "yaml/yml" and "pickle/pkl".
- **file_client_args** (*dict, optional*) – Arguments to instantiate a FileClient. See *mmengine.fileio.FileClient* for details. Defaults to None. It will be deprecated in future. Please use `backend_args` instead.
- **backend_args** (*dict, optional*) – Arguments to instantiate the preifx of uri corresponding backend. Defaults to None. New in v0.2.0.

Examples

```
>>> load('/path/of/your/file')  # file is storaged in disk
>>> load('https://path/of/your/file')  # file is storaged in Internet
>>> load('s3://path/of/your/file')  # file is storaged in petrel
```

Returns The content from the file.

49.3.3 mmengine.fileio.copy_if_symlink_fails

mmengine.fileio.**copy_if_symlink_fails**(*src*, *dst*, *backend_args=None*)

Create a symbolic link pointing to src named dst.

If failed to create a symbolic link pointing to src, directory copy src to dst instead.

Parameters

- **src** (*str or Path*) – Create a symbolic link pointing to src.
- **dst** (*str or Path*) – Create a symbolic link named dst.
- **backend_args** (*dict, optional*) – Arguments to instantiate the corresponding backend. Defaults to None.

Returns Return True if successfully create a symbolic link pointing to src. Otherwise, return False.

Return type bool

Examples

```
>>> src = '/path/of/file'
>>> dst = '/path1/of/file1'
>>> copy_if_symlink_fails(src, dst)
True
>>> src = '/path/of/dir'
>>> dst = '/path1/of/dir1'
>>> copy_if_symlink_fails(src, dst)
True
```

49.3.4 mmengine.fileio.copyfile

mmengine.fileio.**copyfile**(*src*, *dst*, *backend_args=None*)

Copy a file src to dst and return the destination file.

src and dst should have the same prefix. If dst specifies a directory, the file will be copied into dst using the base filename from src. If dst specifies a file that already exists, it will be replaced.

Parameters

- **src** (*str or Path*) – A file to be copied.
- **dst** (*str or Path*) – Copy file to dst.
- **backend_args** (*dict, optional*) – Arguments to instantiate the corresponding backend. Defaults to None.

Returns The destination file.

Return type str

Raises **SameFileError** – If src and dst are the same file, a SameFileError will be raised.

Examples

```
>>> # dst is a file
>>> src = '/path/of/file'
>>> dst = '/path1/of/file1'
>>> # src will be copied to '/path1/of/file1'
>>> copyfile(src, dst)
'/path1/of/file1'
```

```
>>> # dst is a directory
>>> dst = '/path1/of/dir'
>>> # src will be copied to '/path1/of/dir/file'
>>> copyfile(src, dst)
'/path1/of/dir/file'
```

49.3.5 mmengine.fileio.copyfile_from_local

mmengine.fileio.**copyfile_from_local**(*src*, *dst*, *backend_args=None*)

Copy a local file src to dst and return the destination file.

Note: If the backend is the instance of LocalBackend, it does the same thing with *copyfile()*.

Parameters

- **src** (*str or Path*) – A local file to be copied.
- **dst** (*str or Path*) – Copy file to dst.
- **backend_args** (*dict, optional*) – Arguments to instantiate the corresponding backend. Defaults to None.

Returns If dst specifies a directory, the file will be copied into dst using the base filename from src.

Return type str

Examples

```
>>> # dst is a file
>>> src = '/path/of/file'
>>> dst = 's3://openmmlab/mmengine/file1'
>>> # src will be copied to 's3://openmmlab/mmengine/file1'
>>> copyfile_from_local(src, dst)
s3://openmmlab/mmengine/file1
```

```
>>> # dst is a directory
>>> dst = 's3://openmmlab/mmengine'
>>> # src will be copied to 's3://openmmlab/mmengine/file"
>>> copyfile_from_local(src, dst)
's3://openmmlab/mmengine/file'
```

49.3.6 mmengine.fileio.copyfile_to_local

mmengine.fileio.**copyfile_to_local**(*src*, *dst*, *backend_args=None*)

Copy the file src to local dst and return the destination file.

If dst specifies a directory, the file will be copied into dst using the base filename from src. If dst specifies a file that already exists, it will be replaced.

Note: If the backend is the instance of LocalBackend, it does the same thing with *copyfile()*.

Parameters

- **src** (*str or Path*) – A file to be copied.
- **dst** (*str or Path*) – Copy file to to local dst.
- **backend_args** (*dict, optional*) – Arguments to instantiate the corresponding backend. Defaults to None.

Returns If dst specifies a directory, the file will be copied into dst using the base filename from src.

Return type str

Examples

```
>>> # dst is a file
>>> src = 's3://openmmlab/mmengine/file'
>>> dst = '/path/of/file'
>>> # src will be copied to '/path/of/file'
>>> copyfile_to_local(src, dst)
'/path/of/file'
```

```
>>> # dst is a directory
>>> dst = '/path/of/dir'
>>> # src will be copied to '/path/of/dir/file'
>>> copyfile_to_local(src, dst)
'/path/of/dir/file'
```

49.3.7 mmengine.fileio.copytree

mmengine.fileio.**copytree**(*src*, *dst*, *backend_args=None*)

Recursively copy an entire directory tree rooted at src to a directory named dst and return the destination directory.

src and dst should have the same prefix and dst must not already exist.

Parameters

- **src** (*str or Path*) – A directory to be copied.
- **dst** (*str or Path*) – Copy directory to dst.
- **backend_args** (*dict, optional*) – Arguments to instantiate the corresponding backend. Defaults to None.

Returns The destination directory.

Return type str

Raises **FileExistsError** – If dst had already existed, a FileExistsError will be raised.

Examples

```
>>> src = '/path/of/dir1'
>>> dst = '/path/of/dir2'
>>> copytree(src, dst)
'/path/of/dir2'
```

49.3.8 mmengine.fileio.copytree_from_local

mmengine.fileio.**copytree_from_local**(*src*, *dst*, *backend_args=None*)

Recursively copy an entire directory tree rooted at src to a directory named dst and return the destination directory.

Note: If the backend is the instance of LocalBackend, it does the same thing with *copytree()*.

Parameters

- **src** (*str or Path*) – A local directory to be copied.
- **dst** (*str or Path*) – Copy directory to dst.
- **backend_args** (*dict, optional*) – Arguments to instantiate the corresponding backend. Defaults to None.

Returns The destination directory.

Return type str

Examples

```
>>> src = '/path/of/dir'
>>> dst = 's3://openmmlab/mmengine/dir'
>>> copyfile_from_local(src, dst)
's3://openmmlab/mmengine/dir'
```

49.3.9 mmengine.fileio.copytree_to_local

mmengine.fileio.**copytree_to_local**(*src*, *dst*, *backend_args=None*)

Recursively copy an entire directory tree rooted at src to a local directory named dst and return the destination directory.

Note: If the backend is the instance of LocalBackend, it does the same thing with *copytree()*.

Parameters

- **src** (*str or Path*) – A directory to be copied.
- **dst** (*str or Path*) – Copy directory to local dst.
- **backend_args** (*dict, optional*) – Arguments to instantiate the corresponding backend. Defaults to None.

Returns The destination directory.

Return type str

Examples

```
>>> src = 's3://openmmlab/mmengine/dir'
>>> dst = '/path/of/dir'
>>> copytree_to_local(src, dst)
'/path/of/dir'
```

49.3.10 mmengine.fileio.exists

mmengine.fileio.**exists**(*filepath*, *backend_args=None*)

Check whether a file path exists.

Parameters

- **filepath** (*str or Path*) – Path to be checked whether exists.
- **backend_args** (*dict, optional*) – Arguments to instantiate the corresponding backend. Defaults to None.

Returns Return `True` if `filepath` exists, `False` otherwise.

Return type bool

Examples

```
>>> filepath = '/path/of/file'
>>> exists(filepath)
True
```

49.3.11 mmengine.fileio.generate_presigned_url

mmengine.fileio.**generate_presigned_url**(*url*, *client_method='get_object'*, *expires_in=3600*, *backend_args=None*)

Generate the presigned url of video stream which can be passed to mmcv.VideoReader. Now only work on Petrel backend.

Note: Now only work on Petrel backend.

Parameters

- **url** (*str*) – Url of video stream.
- **client_method** (*str*) – Method of client, 'get_object' or 'put_object'. Default: 'get_object'.
- **expires_in** (*int*) – expires, in seconds. Default: 3600.
- **backend_args** (*dict, optional*) – Arguments to instantiate the corresponding backend. Defaults to None.

Returns Generated presigned url.

Return type str

49.3.12 mmengine.fileio.get

mmengine.fileio.**get**(*filepath*, *backend_args=None*)

Read bytes from a given `filepath` with 'rb' mode.

Parameters

- **filepath** (*str or Path*) – Path to read data.
- **backend_args** (*dict, optional*) – Arguments to instantiate the corresponding backend. Defaults to None.

Returns Expected bytes object.

Return type bytes

Examples

```
>>> filepath = '/path/of/file'
>>> get(filepath)
b'hello world'
```

49.3.13 mmengine.fileio.get_file_backend

mmengine.fileio.**get_file_backend**(*uri=None*, ***, *backend_args=None*, *enable_singleton=False*)

Return a file backend based on the prefix of uri or backend_args.

Parameters

- **uri** (*str or Path*) – Uri to be parsed that contains the file prefix.
- **backend_args** (*dict, optional*) – Arguments to instantiate the corresponding backend. Defaults to None.
- **enable_singleton** (*bool*) – Whether to enable the singleton pattern. If it is True, the backend created will be reused if the signature is same with the previous one. Defaults to False.

Returns Instantiated Backend object.

Return type *BaseStorageBackend*

Examples

```
>>> # get file backend based on the prefix of uri
>>> uri = 's3://path/of/your/file'
>>> backend = get_file_backend(uri)
>>> # get file backend based on the backend_args
>>> backend = get_file_backend(backend_args={'backend': 'petrel'})
>>> # backend name has a higher priority if 'backend' in backend_args
>>> backend = get_file_backend(uri, backend_args={'backend': 'petrel'})
```

49.3.14 mmengine.fileio.get_local_path

mmengine.fileio.**get_local_path**(*filepath*, *backend_args=None*)

Download data from `filepath` and write the data to local path.

`get_local_path` is decorated by `contxtlib.contextmanager()`. It can be called with `with` statement, and when exists from the `with` statement, the temporary path will be released.

Note: If the `filepath` is a local path, just return itself and it will not be released (removed).

Parameters

- **filepath** (*str or Path*) – Path to be read data.
- **backend_args** (*dict, optional*) – Arguments to instantiate the corresponding backend. Defaults to None.

Yields *Iterable[str]* – Only yield one path.

Return type Generator[Union[str, pathlib.Path], None, None]

Examples

```
>>> with get_local_path('s3://bucket/abc.jpg') as path:
...     # do something here
```

49.3.15 mmengine.fileio.get_text

mmengine.fileio.**get_text**(*filepath*, *encoding='utf-8'*, *backend_args=None*)

Read text from a given `filepath` with 'r' mode.

Parameters

- **filepath** (*str or Path*) – Path to read data.
- **encoding** (*str*) – The encoding format used to open the `filepath`. Defaults to 'utf-8'.
- **backend_args** (*dict, optional*) – Arguments to instantiate the corresponding backend. Defaults to None.

Returns Expected text reading from `filepath`.

Return type str

Examples

```
>>> filepath = '/path/of/file'
>>> get_text(filepath)
'hello world'
```

49.3.16 mmengine.fileio.isdir

mmengine.fileio.**isdir**(*filepath*, *backend_args=None*)

Check whether a file path is a directory.

Parameters

- **filepath** (*str or Path*) – Path to be checked whether it is a directory.
- **backend_args** (*dict, optional*) – Arguments to instantiate the corresponding backend. Defaults to None.

Returns Return `True` if `filepath` points to a directory, `False` otherwise.

Return type bool

Examples

```
>>> filepath = '/path/of/dir'
>>> isdir(filepath)
True
```

49.3.17 mmengine.fileio.isfile

mmengine.fileio.**isfile**(*filepath*, *backend_args=None*)

Check whether a file path is a file.

Parameters

- **filepath** (*str or Path*) – Path to be checked whether it is a file.
- **backend_args** (*dict, optional*) – Arguments to instantiate the corresponding backend. Defaults to None.

Returns Return `True` if `filepath` points to a file, `False` otherwise.

Return type bool

Examples

```
>>> filepath = '/path/of/file'
>>> isfile(filepath)
True
```

49.3.18 mmengine.fileio.join_path

mmengine.fileio.**join_path**(*filepath*, **filepaths*, *backend_args=None*)

Concatenate all file paths.

Join one or more filepath components intelligently. The return value is the concatenation of filepath and any members of *filepaths.

Parameters

- **filepath** (*str or Path*) – Path to be concatenated.

- ***filepaths** (*str or Path*) – Other paths to be concatenated.
- **backend_args** (*dict, optional*) – Arguments to instantiate the corresponding backend. Defaults to None.
- **filepaths** (*Union[str, pathlib.Path]*) –

Returns The result of concatenation.

Return type str

Examples

```
>>> filepath1 = '/path/of/dir1'
>>> filepath2 = 'dir2'
>>> filepath3 = 'path/of/file'
>>> join_path(filepath1, filepath2, filepath3)
'/path/of/dir/dir2/path/of/file'
```

49.3.19 mmengine.fileio.list_dir_or_file

mmengine.fileio.**list_dir_or_file**(*dir_path*, *list_dir=True*, *list_file=True*, *suffix=None*, *recursive=False*, *backend_args=None*)

Scan a directory to find the interested directories or files in arbitrary order.

Note: *list_dir_or_file()* returns the path relative to `dir_path`.

Parameters

- **dir_path** (*str or Path*) – Path of the directory.
- **list_dir** (*bool*) – List the directories. Defaults to True.
- **list_file** (*bool*) – List the path of files. Defaults to True.
- **suffix** (*str or tuple[str], optional*) – File suffix that we are interested in. Defaults to None.
- **recursive** (*bool*) – If set to True, recursively scan the directory. Defaults to False.
- **backend_args** (*dict, optional*) – Arguments to instantiate the corresponding backend. Defaults to None.

Yields *Iterable[str]* – A relative path to `dir_path`.

Return type Iterator[str]

Examples

```
>>> dir_path = '/path/of/dir'
>>> for file_path in list_dir_or_file(dir_path):
...     print(file_path)
>>> # list those files and directories in current directory
>>> for file_path in list_dir_or_file(dir_path):
...     print(file_path)
>>> # only list files
>>> for file_path in list_dir_or_file(dir_path, list_dir=False):
...     print(file_path)
>>> # only list directories
>>> for file_path in list_dir_or_file(dir_path, list_file=False):
...     print(file_path)
>>> # only list files ending with specified suffixes
>>> for file_path in list_dir_or_file(dir_path, suffix='.txt'):
...     print(file_path)
>>> # list all files and directory recursively
>>> for file_path in list_dir_or_file(dir_path, recursive=True):
...     print(file_path)
```

49.3.20 mmengine.fileio.put

mmengine.fileio.**put**(*obj*, *filepath*, *backend_args=None*)

Write bytes to a given `filepath` with 'wb' mode.

Note: `put` should create a directory if the directory of `filepath` does not exist.

Parameters

- **obj** (*bytes*) – Data to be written.
- **filepath** (*str or Path*) – Path to write data.
- **backend_args** (*dict, optional*) – Arguments to instantiate the corresponding backend. Defaults to None.

Return type None

Examples

```
>>> filepath = '/path/of/file'
>>> put(b'hello world', filepath)
```

49.3.21 mmengine.fileio.put_text

mmengine.fileio.**put_text**(*obj*, *filepath*, *backend_args=None*)

Write text to a given `filepath` with 'w' mode.

Note: `put_text` should create a directory if the directory of `filepath` does not exist.

Parameters

- **obj** (*str*) – Data to be written.
- **filepath** (*str or Path*) – Path to write data.
- **encoding** (*str, optional*) – The encoding format used to open the `filepath`. Defaults to 'utf-8'.
- **backend_args** (*dict, optional*) – Arguments to instantiate the corresponding backend. Defaults to None.

Return type None

Examples

```
>>> filepath = '/path/of/file'
>>> put_text('hello world', filepath)
```

49.3.22 mmengine.fileio.remove

mmengine.fileio.**remove**(*filepath*, *backend_args=None*)

Remove a file.

Parameters

- **filepath** (*str, Path*) – Path to be removed.
- **backend_args** (*dict, optional*) – Arguments to instantiate the corresponding backend. Defaults to None.

Raises

- **FileNotFoundError** – If filepath does not exist, an FileNotFoundError will be raised.
- **IsADirectoryError** – If filepath is a directory, an IsADirectoryError will be raised.

Return type None

Examples

```
>>> filepath = '/path/of/file'
>>> remove(filepath)
```

49.3.23 mmengine.fileio.rmtree

mmengine.fileio.**rmtree**(*dir_path*, *backend_args=None*)

Recursively delete a directory tree.

Parameters

- **dir_path** (*str or Path*) – A directory to be removed.
- **backend_args** (*dict, optional*) – Arguments to instantiate the corresponding backend. Defaults to None.

Return type None

Examples

```
>>> dir_path = '/path/of/dir'
>>> rmtree(dir_path)
```

49.4 Parse File

dict_from_file	Load a text file and parse the content as a dict.
list_from_file	Load a text file and parse the content as a list of strings.

49.4.1 mmengine.fileio.dict_from_file

mmengine.fileio.**dict_from_file**(*filename*, *key_type=<class 'str'>*, *encoding='utf-8'*, *file_client_args=None*, *backend_args=None*)

Load a text file and parse the content as a dict.

Each line of the text file will be two or more columns split by whitespaces or tabs. The first column will be parsed as dict keys, and the following columns will be parsed as dict values.

`dict_from_file` supports loading a text file which can be storaged in different backends and parsing the content as a dict.

Parameters

- **filename** (*str*) – Filename.
- **key_type** (*type*) – Type of the dict keys. str is user by default and type conversion will be performed if specified.
- **encoding** (*str*) – Encoding used to open the file. Defaults to utf-8.
- **file_client_args** (*dict, optional*) – Arguments to instantiate a FileClient. See *mmengine.fileio.FileClient* for details. Defaults to None. It will be deprecated in

future. Please use `backend_args` instead.

- **backend_args** (*dict, optional*) – Arguments to instantiate the preifx of uri corresponding backend. Defaults to None. New in v0.2.0.

Examples

```
>>> dict_from_file('/path/of/your/file')  # disk
{'key1': 'value1', 'key2': 'value2'}
>>> dict_from_file('s3://path/of/your/file')  # ceph or petrel
{'key1': 'value1', 'key2': 'value2'}
```

Returns The parsed contents.

Return type dict

49.4.2 mmengine.fileio.list_from_file

mmengine.fileio.**list_from_file**(*filename*, *prefix=''*, *offset=0*, *max_num=0*, *encoding='utf-8'*, *file_client_args=None*, *backend_args=None*)

Load a text file and parse the content as a list of strings.

`list_from_file` supports loading a text file which can be storaged in different backends and parsing the content as a list for strings.

Parameters

- **filename** (*str*) – Filename.
- **prefix** (*str*) – The prefix to be inserted to the beginning of each item.
- **offset** (*int*) – The offset of lines.
- **max_num** (*int*) – The maximum number of lines to be read, zeros and negatives mean no limitation.
- **encoding** (*str*) – Encoding used to open the file. Defaults to utf-8.
- **file_client_args** (*dict, optional*) – Arguments to instantiate a FileClient. See *mmengine.fileio.FileClient* for details. Defaults to None. It will be deprecated in future. Please use `backend_args` instead.
- **backend_args** (*dict, optional*) – Arguments to instantiate the preifx of uri corresponding backend. Defaults to None. New in v0.2.0.

Examples

```
>>> list_from_file('/path/of/your/file')  # disk
['hello', 'world']
>>> list_from_file('s3://path/of/your/file')  # ceph or petrel
['hello', 'world']
```

Returns A list of strings.

Return type list[str]

CHAPTER

FIFTY

MMENGINE.DIST

mmengine.dist

- *dist*
- *utils*

50.1 dist

gather	Gather data from the whole group to `dst` process.
gather_object	Gathers picklable objects from the whole group in a single process.
all_gather	Gather data from the whole group in a list.
all_gather_object	Gather picklable objects from the whole group into a list.
all_reduce	Reduces the tensor data across all machines in such a way that all get the final result.
all_reduce_dict	Reduces the dict across all machines in such a way that all get the final result.
all_reduce_params	All-reduce parameters.
broadcast	Broadcast the data from `src` process to the whole group.
sync_random_seed	Synchronize a random seed to all processes.
broadcast_object_list	Broadcasts picklable objects in `object_list` to the whole group.
collect_results	Collected results in distributed environments.
collect_results_cpu	Collect results under cpu mode.
collect_results_gpu	Collect results under gpu mode.

50.1.1 mmengine.dist.gather

mmengine.dist.**gather**(*data*, *dst=0*, *group=None*)

Gather data from the whole group to `dst` process.

Note: Calling `gather` in non-distributed environment dose nothing and just returns a list containing `data` itself.

Note: NCCL backend does not support `gather`.

Note: Unlike PyTorch `torch.distributed.gather`, *gather()* in MMEngine does not pass in an empty list `gather_list` and returns the `gather_list` directly, which is more convenient. The difference between their interfaces is as below:

- MMEngine: gather(data, dst, group) -> gather_list
- PyTorch: gather(data, gather_list, dst, group) -> None

Parameters

- **data** (*Tensor*) – Tensor to be gathered. CUDA tensor is not supported.
- **dst** (*int*) – Destination rank. Defaults to 0.
- **group** (*ProcessGroup, optional*) – The process group to work on. If None, the default process group will be used. Defaults to None.

Returns `dst` process will get a list of tensor gathering from the whole group. Other process will get a empty list. If in non-distributed environment, just return a list containing `data` itself.

Return type list[Tensor]

Examples

```
>>> import torch
>>> import mmengine.dist as dist
```

```
>>> # non-distributed environment
>>> data = torch.arange(2, dtype=torch.int64)
>>> data
tensor([0, 1])
>>> output = dist.gather(data)
>>> output
[tensor([0, 1])]
```

```
>>> # distributed environment
>>> # We have 2 process groups, 2 ranks.
>>> data = torch.arange(2, dtype=torch.int64) + 1 + 2 * rank
>>> data
tensor([1, 2]) # Rank 0
tensor([3, 4]) # Rank 1
>>> output = dist.gather(data)
>>> output
[tensor([1, 2]), tensor([3, 4])]  # Rank 0
[]  # Rank 1
```

50.1.2 mmengine.dist.gather_object

mmengine.dist.**gather_object**(*data*, *dst=0*, *group=None*)

Gathers picklable objects from the whole group in a single process. Similar to *gather()*, but Python objects can be passed in. Note that the object must be picklable in order to be gathered.

Note: NCCL backend does not support gather_object.

Note: Unlike PyTorch torch.distributed.gather_object, *gather_object()* in MMEngine does not pass in an empty list gather_list and returns the gather_list directly, which is more convenient. The difference between their interfaces is as below:

- MMEngine: gather_object(data, dst, group) -> gather_list
- PyTorch: gather_object(data, gather_list, data, group) -> None

---

Parameters

- **data** (*Any*) – Input object. Must be picklable.
- **dst** (*int*) – Destination rank. Defaults to 0.
- **group** (*Optional[torch.distributed.distributed_c10d.ProcessGroup]*) – (ProcessGroup, optional): The process group to work on. If None, the default process group will be used. Defaults to None.

Returns list[Any]. On the dst rank, return gather_list which contains the output of the collective.

Return type Optional[List[Any]]

Examples

```
>>> import torch
>>> import mmengine.dist as dist
```

```
>>> # non-distributed environment
>>> data = ['foo', 12, {1: 2}]  # any picklable object
>>> gather_objects = dist.gather_object(data[dist.get_rank()])
>>> output
['foo']
```

```
>>> # distributed environment
>>> # We have 3 process groups, 3 ranks.
>>> dist.gather_object(gather_objects[dist.get_rank()], dst=0)
>>> output
['foo', 12, {1: 2}]  # Rank 0
None  # Rank 1
None  # Rank 2
```

50.1.3 mmengine.dist.all_gather

mmengine.dist.**all_gather**(*data*, *group=None*)

Gather data from the whole group in a list.

Note: Calling `all_gather` in non-distributed environment does nothing and just returns a list containing `data` itself.

Note: Unlike PyTorch `torch.distributed.all_gather`, *all_gather()* in MMEngine does not pass in an empty list `gather_list` and returns the `gather_list` directly, which is more convenient. The difference between their interfaces is as below:

- MMEngine: all_gather(data, group) -> gather_list
- PyTorch: all_gather(gather_list, data, group) -> None

Parameters

- **data** (*Tensor*) – Tensor to be gathered.
- **group** (*ProcessGroup, optional*) – The process group to work on. If None, the default process group will be used. Defaults to None.

Returns Return a list containing data from the whole group if in distributed environment, otherwise a list only containing `data` itself.

Return type list[Tensor]

Examples

```
>>> import torch
>>> import mmengine.dist as dist
```

```
>>> # non-distributed environment
>>> data = torch.arange(2, dtype=torch.int64)
>>> data
tensor([0, 1])
>>> output = dist.all_gather(data)
>>> output
[tensor([0, 1])]
```

```
>>> # distributed environment
>>> # We have 2 process groups, 2 ranks.
>>> data = torch.arange(2, dtype=torch.int64) + 1 + 2 * rank
>>> data
tensor([1, 2])  # Rank 0
tensor([3, 4])  # Rank 1
>>> output = dist.all_gather(data)
>>> output
[tensor([1, 2]), tensor([3, 4])]  # Rank 0
[tensor([1, 2]), tensor([3, 4])]  # Rank 1
```

50.1.4 mmengine.dist.all_gather_object

mmengine.dist.**all_gather_object**(*data*, *group=None*)

Gather picklable objects from the whole group into a list. Similar to *all_gather()*, but Python objects can be passed in. Note that the object must be picklable in order to be gathered.

Note: Calling `all_gather_object` in non-distributed environment does nothing and just returns a list containing `data` itself.

Note: Unlike PyTorch `torch.distributed.all_gather_object`, *all_gather_object()* in MMEngine does not pass in an empty list `gather_list` and returns the `gather_list` directly, which is more convenient. The difference between their interfaces is as below:

- MMEngine: all_gather_object(data, group) -> gather_list
- PyTorch: all_gather_object(gather_list, data, group) -> None

---

Parameters

- **data** (*Any*) – Pickable Python object to be broadcast from current process.
- **group** (*ProcessGroup, optional*) – The process group to work on. If None, the default process group will be used. Defaults to None.

Returns Return a list containing data from the whole group if in distributed environment, otherwise a list only containing `data` itself.

Return type list[Tensor]

Note: For NCCL-based process groups, internal tensor representations of objects must be moved to the GPU device before communication starts. In this case, the used device is given by `torch.cuda.current_device()` and it is the user's responsibility to ensure that this is correctly set so that each rank has an individual GPU, via `torch.cuda.set_device()`.

Examples

```
>>> import torch
>>> import mmengine.dist as dist
```

```
>>> # non-distributed environment
>>> data = ['foo', 12, {1: 2}]  # any picklable object
>>> gather_objects = dist.all_gather_object(data[dist.get_rank()])
>>> output
['foo']
```

```
>>> # distributed environment
>>> # We have 3 process groups, 3 ranks.
>>> output = dist.all_gather_object(data[dist.get_rank()])
>>> output
```

(continues on next page)

(continued from previous page)

```
['foo', 12, {1: 2}]  # Rank 0
['foo', 12, {1: 2}]  # Rank 1
['foo', 12, {1: 2}]  # Rank 2
```

50.1.5 mmengine.dist.all_reduce

mmengine.dist.**all_reduce**(*data*, *op='sum'*, *group=None*)

Reduces the tensor data across all machines in such a way that all get the final result.

After the call `data` is going to be bitwise identical in all processes.

Note: Calling `all_reduce` in non-distributed environment does nothing.

Parameters

- **data** (*Tensor*) – Input and output of the collective. The function operates in-place.
- **op** (*str*) – Operation to reduce data. Defaults to 'sum'. Optional values are 'sum', 'mean' and 'produce', 'min', 'max', 'band', 'bor' and 'bxor'.
- **group** (*ProcessGroup, optional*) – The process group to work on. If None, the default process group will be used. Defaults to None.

Return type None

Examples

```
>>> import torch
>>> import mmengine.dist as dist
```

```
>>> # non-distributed environment
>>> data = torch.arange(2, dtype=torch.int64)
>>> dist.all_reduce(data)
>>> data
tensor([0, 1])
```

```
>>> # distributed environment
>>> # We have 2 process groups, 2 ranks.
>>> data = torch.arange(2, dtype=torch.int64) + 1 + 2 * rank
>>> data
tensor([1, 2]) # Rank 0
tensor([3, 4]) # Rank 1
>>> dist.all_reduce(data, op=dist.ReduceOp.SUM)
>>> data
tensor([4, 6]) # Rank 0
tensor([4, 6]) # Rank 1
```

50.1.6 mmengine.dist.all_reduce_dict

mmengine.dist.**all_reduce_dict**(*data*, *op='sum'*, *group=None*)

Reduces the dict across all machines in such a way that all get the final result.

The code is modified from https://github.com/Megvii- BaseDetection/YOLOX/blob/main/yolox/utils/allreduce_norm.py.

Parameters

- **data** (*dict[str, Tensor]*) – Data to be reduced.
- **op** (*str*) – Operation to reduce data. Defaults to 'sum'. Optional values are 'sum', 'mean' and 'produce', 'min', 'max', 'band', 'bor' and 'bxor'.
- **group** (*ProcessGroup, optional*) – The process group to work on. If None, the default process group will be used. Defaults to None.

Return type None

Examples

```
>>> import torch
>>> import mmengine.dist as dist
```

```
>>> # non-distributed environment
>>> data = {
        'key1': torch.arange(2, dtype=torch.int64),
        'key2': torch.arange(3, dtype=torch.int64)
    }
>>> dist.all_reduce_dict(data)
>>> data
    {'key1': tensor([0, 1]), 'key2': tensor([0, 1, 2])}
```

```
>>> # distributed environment
>>> # We have 2 process groups, 2 ranks.
>>> data = {
        'key1': torch.arange(2, dtype=torch.int64),
        'key2': torch.arange(3, dtype=torch.int64)
    }
>>> dist.all_reduce_dict(data)
>>> data
{'key1': tensor([0, 2]), 'key2': tensor([0, 2, 4])}  # Rank 0
{'key1': tensor([0, 2]), 'key2': tensor([0, 2, 4])}  # Rank 1
```

50.1.7 mmengine.dist.all_reduce_params

mmengine.dist.**all_reduce_params**(*params*, *coalesce=True*, *bucket_size_mb=- 1*, *op='sum'*, *group=None*)

All-reduce parameters.

Parameters

- **params** (*List or Generator[torch.Tensor, None, None]*) – List of parameters or buffers of a model.
- **coalesce** (*bool, optional*) – Whether to reduce parameters as a whole. Defaults to True.

- **bucket_size_mb** (*int, optional*) – Size of bucket, the unit is MB. Defaults to -1.
- **op** (*str*) – Operation to reduce data. Defaults to 'sum'. Optional values are 'sum', 'mean' and 'produce', 'min', 'max', 'band', 'bor' and 'bxor'.
- **group** (*ProcessGroup, optional*) – The process group to work on. If None, the default process group will be used. Defaults to None.

Return type None

Examples

```
>>> import torch
>>> import mmengine.dist as dist
```

```
>>> # non-distributed environment
>>> data = [torch.arange(2), torch.arange(3)]
>>> dist.all_reduce_params(data)
>>> data
    [tensor([0, 1]), tensor([0, 1, 2])]
```

```
>>> # distributed environment
>>> # We have 2 process groups, 2 ranks.
>>> if dist.get_rank() == 0:
...     data = [torch.tensor([1, 2]), torch.tensor([3, 4])]
... else:
...     data = [torch.tensor([2, 3]), torch.tensor([4, 5])]
```

```
>>> dist.all_reduce_params(data)
>>> data
    [torch.tensor([3, 5]), torch.tensor([7, 9])]
```

50.1.8 mmengine.dist.broadcast

mmengine.dist.**broadcast**(*data*, *src=0*, *group=None*)

Broadcast the data from `src` process to the whole group.

`data` must have the same number of elements in all processes participating in the collective.

Note: Calling `broadcast` in non-distributed environment does nothing.

Parameters

- **data** (*Tensor*) – Data to be sent if `src` is the rank of current process, and data to be used to save received data otherwise.
- **src** (*int*) – Source rank. Defaults to 0.
- **group** (*ProcessGroup, optional*) – The process group to work on. If None, the default process group will be used. Defaults to None.

Return type None

Examples

```
>>> import torch
>>> import mmengine.dist as dist
```

```
>>> # non-distributed environment
>>> data = torch.arange(2, dtype=torch.int64)
>>> data
tensor([0, 1])
>>> dist.broadcast(data)
>>> data
tensor([0, 1])
```

```
>>> # distributed environment
>>> # We have 2 process groups, 2 ranks.
>>> data = torch.arange(2, dtype=torch.int64) + 1 + 2 * rank
>>> data
tensor([1, 2]) # Rank 0
tensor([3, 4]) # Rank 1
>>> dist.broadcast(data)
>>> data
tensor([1, 2]) # Rank 0
tensor([1, 2]) # Rank 1
```

50.1.9 mmengine.dist.sync_random_seed

mmengine.dist.**sync_random_seed**(*group=None*)

Synchronize a random seed to all processes.

In distributed sampling, different ranks should sample non-overlapped data in the dataset. Therefore, this function is used to make sure that each rank shuffles the data indices in the same order based on the same seed. Then different ranks could use different indices to select non-overlapped data from the same data list.

Parameters **group** (*ProcessGroup, optional*) – The process group to work on. If None, the default process group will be used. Defaults to None.

Returns Random seed.

Return type int

Examples

```
>>> import torch
>>> import mmengine.dist as dist
```

```
>>> # non-distributed environment
>>> seed = dist.sync_random_seed()
>>> seed  # which a random number
587791752
```

```
>>> distributed environment
>>> # We have 2 process groups, 2 ranks.
>>> seed = dist.sync_random_seed()
>>> seed
587791752  # Rank 0
587791752  # Rank 1
```

50.1.10 mmengine.dist.broadcast_object_list

mmengine.dist.**broadcast_object_list**(*data*, *src=0*, *group=None*)

Broadcasts picklable objects in `object_list` to the whole group. Similar to *broadcast()*, but Python objects can be passed in. Note that all objects in `object_list` must be picklable in order to be broadcasted.

Note: Calling `broadcast_object_list` in non-distributed environment does nothing.

Parameters

- **data** (*List[Any]*) – List of input objects to broadcast. Each object must be picklable. Only objects on the `src` rank will be broadcast, but each rank must provide lists of equal sizes.
- **src** (*int*) – Source rank from which to broadcast `object_list`.
- **group** (*Optional[torch.distributed.distributed_c10d.ProcessGroup]*) – (ProcessGroup, optional): The process group to work on. If None, the default process group will be used. Default is `None`.
- **device** (`torch.device`, optional) – If not None, the objects are serialized and converted to tensors which are moved to the `device` before broadcasting. Default is `None`.

Return type None

Note: For NCCL-based process groups, internal tensor representations of objects must be moved to the GPU device before communication starts. In this case, the used device is given by `torch.cuda.current_device()` and it is the user's responsibility to ensure that this is correctly set so that each rank has an individual GPU, via `torch.cuda.set_device()`.

Examples

```
>>> import torch
>>> import mmengine.dist as dist
```

```
>>> # non-distributed environment
>>> data = ['foo', 12, {1: 2}]
>>> dist.broadcast_object_list(data)
>>> data
['foo', 12, {1: 2}]
```

```
>>> # distributed environment
>>> # We have 2 process groups, 2 ranks.
>>> if dist.get_rank() == 0:
>>>     # Assumes world_size of 3.
>>>     data = ["foo", 12, {1: 2}]  # any picklable object
>>> else:
>>>     data = [None, None, None]
>>> dist.broadcast_object_list(data)
>>> data
["foo", 12, {1: 2}]  # Rank 0
["foo", 12, {1: 2}]  # Rank 1
```

50.1.11 mmengine.dist.collect_results

mmengine.dist.**collect_results**(*results*, *size*, *device='cpu'*, *tmpdir=None*)

Collected results in distributed environments.

Parameters

- **results** (*list[object]*) – Result list containing result parts to be collected. Each item of `result_part` should be a picklable object.
- **size** (*int*) – Size of the results, commonly equal to length of the results.
- **device** (*str*) – Device name. Optional values are 'cpu' and 'gpu'.
- **tmpdir** (*str | None*) – Temporal directory for collected results to store. If set to None, it will create a temporal directory for it. `tmpdir` should be None when device is 'gpu'. Defaults to None.

Returns The collected results.

Return type list or None

Examples

```
>>> # distributed environment
>>> # We have 2 process groups, 2 ranks.
>>> import mmengine.dist as dist
>>> if dist.get_rank() == 0:
        data = ['foo', {1: 2}]
    else:
        data = [24, {'a': 'b'}]
>>> size = 4
>>> output = dist.collect_results(data, size, device='cpu')
>>> output
['foo', 24, {1: 2}, {'a': 'b'}]  # rank 0
None  # rank 1
```

50.1.12 mmengine.dist.collect_results_cpu

mmengine.dist.**collect_results_cpu**(*result_part*, *size*, *tmpdir=None*)

Collect results under cpu mode.

On cpu mode, this function will save the results on different gpus to `tmpdir` and collect them by the rank 0 worker.

Parameters

- **result_part** (*list*) – Result list containing result parts to be collected. Each item of `result_part` should be a picklable object.
- **size** (*int*) – Size of the results, commonly equal to length of the results.
- **tmpdir** (*str | None*) – Temporal directory for collected results to store. If set to None, it will create a random temporal directory for it. Defaults to None.

Returns The collected results.

Return type list or None

Examples

```
>>> # distributed environment
>>> # We have 2 process groups, 2 ranks.
>>> import mmengine.dist as dist
>>> if dist.get_rank() == 0:
        data = ['foo', {1: 2}]
    else:
        data = [24, {'a': 'b'}]
>>> size = 4
>>> output = dist.collect_results_cpu(data, size)
>>> output
['foo', 24, {1: 2}, {'a': 'b'}]  # rank 0
None  # rank 1
```

50.1.13 mmengine.dist.collect_results_gpu

mmengine.dist.**collect_results_gpu**(*result_part*, *size*)

Collect results under gpu mode.

On gpu mode, this function will encode results to gpu tensors and use gpu communication for results collection.

Parameters

- **result_part** (*list[object]*) – Result list containing result parts to be collected. Each item of `result_part` should be a picklable object.
- **size** (*int*) – Size of the results, commonly equal to length of the results.

Returns The collected results.

Return type list or None

Examples

```
>>> # distributed environment
>>> # We have 2 process groups, 2 ranks.
>>> import mmengine.dist as dist
>>> if dist.get_rank() == 0:
        data = ['foo', {1: 2}]
    else:
        data = [24, {'a': 'b'}]
>>> size = 4
>>> output = dist.collect_results_gpu(data, size)
>>> output
['foo', 24, {1: 2}, {'a': 'b'}]  # rank 0
None  # rank 1
```

50.2 utils

get_dist_info	Get distributed information of the given process group.
init_dist	Initialize distributed environment.
init_local_group	Setup the local process group.
get_backend	Return the backend of the given process group.
get_world_size	Return the number of the given process group.
get_rank	Return the rank of the given process group.
get_local_size	Return the number of the current node.
get_local_rank	Return the rank of current process in the current node.
is_main_process	Whether the current rank of the given process group is equal to 0.
master_only	Decorate those methods which should be executed in master process.
barrier	Synchronize all processes from the given process group.
is_distributed	Return True if distributed environment has been initialized.
get_local_group	Return local process group.
get_default_group	Return default process group.
get_data_device	Return the device of `data`.
get_comm_device	Return the device for communication among groups.
cast_data_device	Recursively convert Tensor in `data` to `device`.

50.2.1 mmengine.dist.get_dist_info

mmengine.dist.**get_dist_info**(*group=None*)

Get distributed information of the given process group.

Note: Calling `get_dist_info` in non-distributed environment will return (0, 1).

Parameters **group** (*ProcessGroup, optional*) – The process group to work on. If None, the default process group will be used. Defaults to None.

Returns Return a tuple containing the `rank` and `world_size`.

Return type tuple[int, int]

50.2.2 mmengine.dist.init_dist

mmengine.dist.**init_dist**(*launcher*, *backend='nccl'*, ***kwargs*)

Initialize distributed environment.

Parameters

- **launcher** (*str*) – Way to launcher multi processes. Supported launchers are 'pytorch', 'mpi' and 'slurm'.
- **backend** (*str*) – Communication Backends. Supported backends are 'nccl', 'gloo' and 'mpi'. Defaults to 'nccl'.
- ****kwargs** – keyword arguments are passed to `init_process_group`.

Return type None

50.2.3 mmengine.dist.init_local_group

mmengine.dist.**init_local_group**(*node_rank*, *num_gpus_per_node*)

Setup the local process group.

Setup a process group which only includes processes that on the same machine as the current process.

The code is modified from https://github.com/facebookresearch/detectron2/blob/main/detectron2/engine/launch.py

Parameters

- **node_rank** (*int*) – Rank of machines used for training.
- **num_gpus_per_node** (*int*) – Number of gpus used for training in a single machine.

50.2.4 mmengine.dist.get_backend

mmengine.dist.**get_backend**(*group=None*)

Return the backend of the given process group.

Note: Calling `get_backend` in non-distributed environment will return None.

Parameters **group** (*ProcessGroup, optional*) – The process group to work on. The default is the general main process group. If another specific group is specified, the calling process must be part of `group`. Defaults to None.

Returns Return the backend of the given process group as a lower case string if in distributed environment, otherwise None.

Return type str or None

50.2.5 mmengine.dist.get_world_size

mmengine.dist.**get_world_size**(*group=None*)
Return the number of the given process group.

Note: Calling `get_world_size` in non-distributed environment will return 1.

Parameters **group** (*ProcessGroup, optional*) – The process group to work on. If None, the default process group will be used. Defaults to None.

Returns Return the number of processes of the given process group if in distributed environment, otherwise 1.

Return type int

50.2.6 mmengine.dist.get_rank

mmengine.dist.**get_rank**(*group=None*)
Return the rank of the given process group.

Rank is a unique identifier assigned to each process within a distributed process group. They are always consecutive integers ranging from 0 to `world_size`.

Note: Calling `get_rank` in non-distributed environment will return 0.

Parameters **group** (*ProcessGroup, optional*) – The process group to work on. If None, the default process group will be used. Defaults to None.

Returns Return the rank of the process group if in distributed environment, otherwise 0.

Return type int

50.2.7 mmengine.dist.get_local_size

mmengine.dist.**get_local_size**()
Return the number of the current node.

Returns Return the number of processes in the current node if in distributed environment, otherwise 1.

Return type int

50.2.8 mmengine.dist.get_local_rank

mmengine.dist.**get_local_rank**()
Return the rank of current process in the current node.

Returns Return the rank of current process in the current node if in distributed environment, otherwise 0

Return type int

50.2.9 mmengine.dist.is_main_process

mmengine.dist.**is_main_process**(*group=None*)
Whether the current rank of the given process group is equal to 0.

Parameters **group** (*ProcessGroup, optional*) – The process group to work on. If None, the default process group will be used. Defaults to None.

Returns Return True if the current rank of the given process group is equal to 0, otherwise False.

Return type bool

50.2.10 mmengine.dist.master_only

mmengine.dist.**master_only**(*func*)
Decorate those methods which should be executed in master process.

Parameters **func** (*callable*) – Function to be decorated.

Returns Return decorated function.

Return type callable

50.2.11 mmengine.dist.barrier

mmengine.dist.**barrier**(*group=None*)
Synchronize all processes from the given process group.

This collective blocks processes until the whole group enters this function.

Note: Calling `barrier` in non-distributed environment will do nothing.

Parameters **group** (*ProcessGroup, optional*) – The process group to work on. If None, the default process group will be used. Defaults to None.

Return type None

50.2.12 mmengine.dist.is_distributed

mmengine.dist.**is_distributed**()
Return True if distributed environment has been initialized.

Return type bool

50.2.13 mmengine.dist.get_local_group

mmengine.dist.**get_local_group**()
Return local process group.

Return type Optional[torch.distributed.distributed_c10d.ProcessGroup]

50.2.14 mmengine.dist.get_default_group

mmengine.dist.**get_default_group**()
Return default process group.

Return type Optional[torch.distributed.distributed_c10d.ProcessGroup]

50.2.15 mmengine.dist.get_data_device

mmengine.dist.**get_data_device**(*data*)
Return the device of `data`.

If `data` is a sequence of Tensor, all items in `data` should have a same device type.

If `data` is a dict whose values are Tensor, all values should have a same device type.

Parameters **data** (*Tensor or Sequence or dict*) – Inputs to be inferred the device.

Returns The device of `data`.

Return type torch.device

Examples

```
>>> import torch
>>> from mmengine.dist import cast_data_device
>>> # data is a Tensor
>>> data = torch.tensor([0, 1])
>>> get_data_device(data)
device(type='cpu')
>>> # data is a list of Tensor
>>> data = [torch.tensor([0, 1]), torch.tensor([2, 3])]
>>> get_data_device(data)
device(type='cpu')
>>> # data is a dict
>>> data = {'key1': torch.tensor([0, 1]), 'key2': torch.tensor([0, 1])}
>>> get_data_device(data)
device(type='cpu')
```

50.2.16 mmengine.dist.get_comm_device

mmengine.dist.**get_comm_device**(*group=None*)

Return the device for communication among groups.

Parameters **group** (*ProcessGroup, optional*) – The process group to work on.

Returns The device of backend.

Return type torch.device

50.2.17 mmengine.dist.cast_data_device

mmengine.dist.**cast_data_device**(*data*, *device*, *out=None*)

Recursively convert Tensor in `data` to `device`.

If `data` has already on the `device`, it will not be casted again.

Parameters

- **data** (*Tensor or list or dict*) – Inputs to be casted.
- **device** (*torch.device*) – Destination device type.
- **out** (*Tensor or list or dict, optional*) – If `out` is specified, its value will be equal to `data`. Defaults to None.

Returns `data` was casted to `device`.

Return type Tensor or list or dict

CHAPTER

FIFTYONE

MMENGINE.UTILS

mmengine.utils

- *Manager*
- *Path*
- *Package*
- *Version*
- *Progress Bar*
- *Miscellaneous*

51.1 Manager

ManagerMeta	The metaclass for global accessible class.
ManagerMixin	`ManagerMixin` is the base class for classes that have global access requirements.

51.1.1 ManagerMeta

class mmengine.utils.**ManagerMeta**(**args*)

The metaclass for global accessible class.

The subclasses inheriting from `ManagerMeta` will manage their own `_instance_dict` and root instances. The constructors of subclasses must contain the `name` argument.

Examples

```
>>> class SubClass1(metaclass=ManagerMeta):
>>>     def __init__(self, *args, **kwargs):
>>>         pass
AssertionError: <class '__main__.SubClass1'>.__init__ must have the
name argument.
>>> class SubClass2(metaclass=ManagerMeta):
>>>     def __init__(self, name):
>>>         pass
>>> # valid format.
```

51.1.2 ManagerMixin

class mmengine.utils.**ManagerMixin**(*name=''*, ***kwargs*)

`ManagerMixin` is the base class for classes that have global access requirements.

The subclasses inheriting from `ManagerMixin` can get their global instances.

Examples

```
>>> class GlobalAccessible(ManagerMixin):
>>>     def __init__(self, name=''):
>>>         super().__init__(name)
>>>
>>> GlobalAccessible.get_instance('name')
>>> instance_1 = GlobalAccessible.get_instance('name')
>>> instance_2 = GlobalAccessible.get_instance('name')
>>> assert id(instance_1) == id(instance_2)
```

Parameters **name** (*str*) – Name of the instance. Defaults to ‘’.

classmethod **check_instance_created**(*name*)

Check whether the name corresponding instance exists.

Parameters **name** (*str*) – Name of instance.

Returns Whether the name corresponding instance exists.

Return type bool

classmethod **get_current_instance**()

Get latest created instance.

Before calling `get_current_instance`, The subclass must have called `get_instance(xxx)` at least once.

Examples

```
>>> instance = GlobalAccessible.get_current_instance()
AssertionError: At least one of name and current needs to be set
>>> instance = GlobalAccessible.get_instance('name1')
>>> instance.instance_name
```

(continues on next page)

(continued from previous page)

```
name1
>>> instance = GlobalAccessible.get_current_instance()
>>> instance.instance_name
name1
```

Returns Latest created instance.

Return type object

classmethod get_instance(*name*, ***kwargs*)

Get subclass instance by name if the name exists.

If corresponding name instance has not been created, `get_instance` will create an instance, otherwise `get_instance` will return the corresponding instance.

Examples

```
>>> instance1 = GlobalAccessible.get_instance('name1')
>>> # Create name1 instance.
>>> instance.instance_name
name1
>>> instance2 = GlobalAccessible.get_instance('name1')
>>> # Get name1 instance.
>>> assert id(instance1) == id(instance2)
```

Parameters **name** (*str*) – Name of instance. Defaults to ‘’.

Returns Corresponding name instance, the latest instance, or root instance.

Return type object

property instance_name: str

Get the name of instance.

Returns Name of instance.

Return type str

51.2 Path

check_file_exist	
fopen	
is_abs	Check if path is an absolute path in different backends.
is_filepath	
mkdir_or_exist	
scandir	Scan a directory to find the interested files.
symlink	

51.2.1 mmengine.utils.check_file_exist

mmengine.utils.**check_file_exist**(*filename*, *msg_tmpl='file "{}" does not exist'*)

51.2.2 mmengine.utils.fopen

mmengine.utils.**fopen**(*filepath*, **args*, ***kwargs*)

51.2.3 mmengine.utils.is_abs

mmengine.utils.**is_abs**(*path*)

Check if path is an absolute path in different backends.

Parameters **path** (*str*) – path of directory or file.

Returns whether path is an absolute path.

Return type bool

51.2.4 mmengine.utils.is_filepath

mmengine.utils.**is_filepath**(*x*)

51.2.5 mmengine.utils.mkdir_or_exist

mmengine.utils.**mkdir_or_exist**(*dir_name*, *mode=511*)

51.2.6 mmengine.utils.scandir

mmengine.utils.**scandir**(*dir_path*, *suffix=None*, *recursive=False*, *case_sensitive=True*)

Scan a directory to find the interested files.

Parameters

- **dir_path** (str | Path) – Path of the directory.
- **suffix** (*str | tuple(str), optional*) – File suffix that we are interested in. Default: None.
- **recursive** (*bool, optional*) – If set to True, recursively scan the directory. Default: False.
- **case_sensitive** (*bool, optional*) – If set to False, ignore the case of suffix. Default: True.

Returns A generator for all the interested files with relative paths.

51.2.7 mmengine.utils.symlink

mmengine.utils.**symlink**(*src*, *dst*, *overwrite=True*, ***kwargs*)

51.3 Package

call_command	
install_package	
get_installed_path	Get installed path of package.
is_installed	Check package whether installed.

51.3.1 mmengine.utils.call_command

mmengine.utils.**call_command**(*cmd*)

Parameters **cmd** (*list*) –

Return type None

51.3.2 mmengine.utils.install_package

mmengine.utils.**install_package**(*package*)

Parameters **package** (*str*) –

51.3.3 mmengine.utils.get_installed_path

mmengine.utils.**get_installed_path**(*package*)

Get installed path of package.

Parameters **package** (*str*) – Name of package.

Return type str

Example

```
>>> get_installed_path('mmcls')
>>> '.../lib/python3.7/site-packages/mmcls'
```

51.3.4 mmengine.utils.is_installed

mmengine.utils.**is_installed**(*package*)
Check package whether installed.

Parameters **package** (*str*) – Name of package to be checked.

Return type bool

51.4 Version

digit_version	Convert a version string into a tuple of integers.
get_git_hash	Get the git hash of the current repo.

51.4.1 mmengine.utils.digit_version

mmengine.utils.**digit_version**(*version_str*, *length=4*)
Convert a version string into a tuple of integers.

This method is usually used for comparing two versions. For pre-release versions: alpha < beta < rc.

Parameters

- **version_str** (*str*) – The version string.
- **length** (*int*) – The maximum number of version levels. Default: 4.

Returns The version info in digits (integers).

Return type tuple[int]

51.4.2 mmengine.utils.get_git_hash

mmengine.utils.**get_git_hash**(*fallback='unknown'*, *digits=None*)
Get the git hash of the current repo.

Parameters

- **fallback** (*str, optional*) – The fallback string when git hash is unavailable. Defaults to 'unknown'.
- **digits** (*int, optional*) – kept digits of the hash. Defaults to None, meaning all digits are kept.

Returns Git commit hash.

Return type str

51.5 Progress Bar

ProgressBar	A progress bar which can print the progress.

51.5.1 ProgressBar

class mmengine.utils.**ProgressBar**(*task_num=0*, *bar_width=50*, *start=True*, *file=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)

A progress bar which can print the progress.

track_iter_progress	Track the progress of tasks iteration or enumeration with a progress bar.
track_parallel_progress	Track the progress of parallel task execution with a progress bar.
track_progress	Track the progress of tasks execution with a progress bar.

51.5.2 mmengine.utils.track_iter_progress

mmengine.utils.**track_iter_progress**(*tasks*, *bar_width=50*, *file=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)

Track the progress of tasks iteration or enumeration with a progress bar.

Tasks are yielded with a simple for-loop.

Parameters

- **tasks** (*list or tuple[Iterable, int]*) – A list of tasks or (tasks, total num).
- **bar_width** (*int*) – Width of progress bar.

Yields *list* – The task results.

51.5.3 mmengine.utils.track_parallel_progress

mmengine.utils.**track_parallel_progress**(*func*, *tasks*, *nproc*, *initializer=None*, *initargs=None*, *bar_width=50*, *chunksize=1*, *skip_first=False*, *keep_order=True*, *file=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)

Track the progress of parallel task execution with a progress bar.

The built-in `multiprocessing` module is used for process pools and tasks are done with `Pool.map()` or `Pool.imap_unordered()`.

Parameters

- **func** (*callable*) – The function to be applied to each task.
- **tasks** (*list or tuple[Iterable, int]*) – A list of tasks or (tasks, total num).
- **nproc** (*int*) – Process (worker) number.
- **initializer** (*None or callable*) – Refer to `multiprocessing.Pool` for details.
- **initargs** (*None or tuple*) – Refer to `multiprocessing.Pool` for details.

- **chunksize** (*int*) – Refer to `multiprocessing.Pool` for details.
- **bar_width** (*int*) – Width of progress bar.
- **skip_first** (*bool*) – Whether to skip the first sample for each worker when estimating fps, since the initialization step may takes longer.
- **keep_order** (*bool*) – If True, `Pool.imap()` is used, otherwise `Pool.imap_unordered()` is used.

Returns The task results.

Return type list

51.5.4 mmengine.utils.track_progress

mmengine.utils.**track_progress**(*func*, *tasks*, *bar_width=50*, *file=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, ***kwargs*)

Track the progress of tasks execution with a progress bar.

Tasks are done with a simple for-loop.

Parameters

- **func** (*callable*) – The function to be applied to each task.
- **tasks** (*list or tuple[Iterable, int]*) – A list of tasks or (tasks, total num).
- **bar_width** (*int*) – Width of progress bar.

Returns The task results.

Return type list

51.6 Miscellaneous

Timer	A flexible Timer class.
TimerError	

51.6.1 Timer

class mmengine.utils.**Timer**(*start=True*, *print_tmpl=None*)

A flexible Timer class.

Examples

```
>>> import time
>>> import mmcv
>>> with mmcv.Timer():
>>>     # simulate a code block that will run for 1s
>>>     time.sleep(1)
1.000
>>> with mmcv.Timer(print_tmpl='it takes {:.1f} seconds'):
>>>     # simulate a code block that will run for 1s
>>>     time.sleep(1)
it takes 1.0 seconds
>>> timer = mmcv.Timer()
>>> time.sleep(0.5)
>>> print(timer.since_start())
0.500
>>> time.sleep(0.5)
>>> print(timer.since_last_check())
0.500
>>> print(timer.since_start())
1.000
```

property is_running
indicate whether the timer is running

Type bool

since_last_check()
Time since the last checking.

Either *since_start()* or *since_last_check()* is a checking operation.

Returns Time in seconds.

Return type float

since_start()
Total time since the timer is started.

Returns Time in seconds.

Return type float

start()
Start the timer.

51.6.2 TimerError

class mmengine.utils.**TimerError**(*message*)

is_list_of	Check whether it is a list of some type.
is_tuple_of	Check whether it is a tuple of some type.
is_seq_of	Check whether it is a sequence of some type.
is_str	Whether the input is an string instance.
iter_cast	Cast elements of an iterable object into some type.

continues on next page

Table 8 – continued from previous page

list_cast	Cast elements of an iterable object into a list of some type.
tuple_cast	Cast elements of an iterable object into a tuple of some type.
concat_list	Concatenate a list of list into a single list.
slice_list	Slice a list into several sub lists by a list of given length.
to_1tuple	
to_2tuple	
to_3tuple	
to_4tuple	
to_ntuple	
check_prerequisites	A decorator factory to check if prerequisites are satisfied.
deprecated_api_warning	A decorator to check if some arguments are deprecate and try to replace deprecate src_arg_name to dst_arg_name.
deprecated_function	Marks functions as deprecated.
has_method	Check whether the object has a method.
is_method_overridden	Check if a method of base class is overridden in derived class.
import_modules_from_strings	Import modules from the given list of strings.
requires_executable	A decorator to check if some executable files are installed.
requires_package	A decorator to check if some python packages are installed.
check_time	Add check points in a single line.

51.6.3 mmengine.utils.is_list_of

mmengine.utils.**is_list_of**(*seq*, *expected_type*)

Check whether it is a list of some type.

A partial method of *is_seq_of()*.

51.6.4 mmengine.utils.is_tuple_of

mmengine.utils.**is_tuple_of**(*seq*, *expected_type*)

Check whether it is a tuple of some type.

A partial method of *is_seq_of()*.

51.6.5 mmengine.utils.is_seq_of

mmengine.utils.**is_seq_of**(*seq*, *expected_type*, *seq_type=None*)

Check whether it is a sequence of some type.

Parameters

- **seq** (*Sequence*) – The sequence to be checked.
- **expected_type** (*type or tuple*) – Expected type of sequence items.
- **seq_type** (*type, optional*) – Expected sequence type. Defaults to None.

Returns Return True if `seq` is valid else False.

Return type bool

Examples

```
>>> from mmengine.utils import is_seq_of
>>> seq = ['a', 'b', 'c']
>>> is_seq_of(seq, str)
True
>>> is_seq_of(seq, int)
False
```

51.6.6 mmengine.utils.is_str

mmengine.utils.**is_str**(*x*)

Whether the input is an string instance.

Note: This method is deprecated since python 2 is no longer supported.

51.6.7 mmengine.utils.iter_cast

mmengine.utils.**iter_cast**(*inputs*, *dst_type*, *return_type=None*)

Cast elements of an iterable object into some type.

Parameters

- **inputs** (*Iterable*) – The input object.
- **dst_type** (*type*) – Destination type.
- **return_type** (*type, optional*) – If specified, the output object will be converted to this type, otherwise an iterator.

Returns The converted object.

Return type iterator or specified type

51.6.8 mmengine.utils.list_cast

mmengine.utils.**list_cast**(*inputs*, *dst_type*)
Cast elements of an iterable object into a list of some type.

A partial method of *iter_cast()*.

51.6.9 mmengine.utils.tuple_cast

mmengine.utils.**tuple_cast**(*inputs*, *dst_type*)
Cast elements of an iterable object into a tuple of some type.

A partial method of *iter_cast()*.

51.6.10 mmengine.utils.concat_list

mmengine.utils.**concat_list**(*in_list*)
Concatenate a list of list into a single list.

Parameters **in_list** (*list*) – The list of list to be merged.

Returns The concatenated flat list.

Return type list

51.6.11 mmengine.utils.slice_list

mmengine.utils.**slice_list**(*in_list*, *lens*)
Slice a list into several sub lists by a list of given length.

Parameters

- **in_list** (*list*) – The list to be sliced.
- **lens** (*int or list*) – The expected length of each out list.

Returns A list of sliced list.

Return type list

51.6.12 mmengine.utils.to_1tuple

mmengine.utils.**to_1tuple**(*x*)

51.6.13 mmengine.utils.to_2tuple

mmengine.utils.**to_2tuple**(*x*)

51.6.14 mmengine.utils.to_3tuple

mmengine.utils.**to_3tuple**(*x*)

51.6.15 mmengine.utils.to_4tuple

mmengine.utils.**to_4tuple**(*x*)

51.6.16 mmengine.utils.to_ntuple

mmengine.utils.**to_ntuple**(*n*)

51.6.17 mmengine.utils.check_prerequisites

mmengine.utils.**check_prerequisites**(*prerequisites*, *checker*, *msg_tmpl='Prerequisites "{}" are required in method "{}" but not found, please install them first.'*)

A decorator factory to check if prerequisites are satisfied.

Parameters

- **prerequisites** (*str of list[str]*) – Prerequisites to be checked.
- **checker** (*callable*) – The checker method that returns True if a prerequisite is meet, False otherwise.
- **msg_tmpl** (*str*) – The message template with two variables.

Returns A specific decorator.

Return type decorator

51.6.18 mmengine.utils.deprecated_api_warning

mmengine.utils.**deprecated_api_warning**(*name_dict*, *cls_name=None*)

A decorator to check if some arguments are deprecate and try to replace deprecate src_arg_name to dst_arg_name.

Parameters

- **name_dict** (*dict*) – key (str): Deprecate argument names. val (str): Expected argument names.
- **cls_name** (*Optional[str]*) –

Returns New function.

Return type func

51.6.19 mmengine.utils.deprecated_function

mmengine.utils.**deprecated_function**(*since*, *removed_in*, *instructions*)

Marks functions as deprecated.

Throw a warning when a deprecated function is called, and add a note in the docstring. Modified from https://github.com/pytorch/pytorch/blob/master/torch/onnx/_deprecation.py

Parameters

- **since** (*str*) – The version when the function was first deprecated.
- **removed_in** (*str*) – The version when the function will be removed.
- **instructions** (*str*) – The action users should take.

Returns A new function, which will be deprecated soon.

Return type Callable

51.6.20 mmengine.utils.has_method

mmengine.utils.**has_method**(*obj*, *method*)

Check whether the object has a method.

Parameters

- **method** (*str*) – The method name to check.
- **obj** (*object*) – The object to check.

Returns True if the object has the method else False.

Return type bool

51.6.21 mmengine.utils.is_method_overridden

mmengine.utils.**is_method_overridden**(*method*, *base_class*, *derived_class*)

Check if a method of base class is overridden in derived class.

Parameters

- **method** (*str*) – the method name to check.
- **base_class** (*type*) – the class of the base class.
- **derived_class** (*type | Any*) – the class or instance of the derived class.

Return type bool

51.6.22 mmengine.utils.import_modules_from_strings

mmengine.utils.**import_modules_from_strings**(*imports*, *allow_failed_imports=False*)

Import modules from the given list of strings.

Parameters

- **imports** (*list | str | None*) – The given module names to be imported.
- **allow_failed_imports** (*bool*) – If True, the failed imports will return None. Otherwise, an ImportError is raise. Default: False.

Returns The imported modules.

Return type list[module] | module | None

Examples

```
>>> osp, sys = import_modules_from_strings(
...     ['os.path', 'sys'])
>>> import os.path as osp_
>>> import sys as sys_
>>> assert osp == osp_
>>> assert sys == sys_
```

51.6.23 mmengine.utils.requires_executable

mmengine.utils.**requires_executable**(*prerequisites*)

A decorator to check if some executable files are installed.

Example

```
>>> @requires_executable('ffmpeg')
>>> func(arg1, args):
>>>     print(1)
1
```

51.6.24 mmengine.utils.requires_package

mmengine.utils.**requires_package**(*prerequisites*)

A decorator to check if some python packages are installed.

Example

```
>>> @requires_package('numpy')
>>> func(arg1, args):
>>>     return numpy.zeros(1)
array([0.])
>>> @requires_package(['numpy', 'non_package'])
>>> func(arg1, args):
>>>     return numpy.zeros(1)
ImportError
```

51.6.25 mmengine.utils.check_time

mmengine.utils.**check_time**(*timer_id*)

Add check points in a single line.

This method is suitable for running a task on a list of items. A timer will be registered when the method is called for the first time.

Examples

```
>>> import time
>>> import mmcv
>>> for i in range(1, 6):
>>>     # simulate a code block
>>>     time.sleep(i)
>>>     mmcv.check_time('task1')
2.000
3.000
4.000
5.000
```

Parameters **str** – Timer identifier.

CHAPTER

FIFTYTWO

MMENGINE.UTILS.DL_UTILS

TimeCounter	A tool that counts the average running time of a function or a method.

52.1 TimeCounter

class mmengine.utils.dl_utils.**TimeCounter**(*log_interval=1*, *warmup_interval=1*, *with_sync=True*, *tag=None*, *logger=None*)

A tool that counts the average running time of a function or a method. Users can use it as a decorator or context manager to calculate the average running time of code blocks.

Parameters

- **log_interval** (*int*) – The interval of logging. Defaults to 1.
- **warmup_interval** (*int*) – The interval of warmup. Defaults to 1.
- **with_sync** (*bool*) – Whether to synchronize cuda. Defaults to True.
- **tag** (*str, optional*) – Function tag. Used to distinguish between different functions or methods being called. Defaults to None.
- **logger** (*MMLogger, optional*) – Formatted logger used to record messages. Defaults to None.

Examples

```
>>> import time
>>> from mmengine.utils.dl_utils import TimeCounter
>>> @TimeCounter()
... def fun1():
...     time.sleep(0.1)
... fun1()
[fun1]-time per run averaged in the past 1 runs: 100.0 ms
```

```
>>> @@TimeCounter(log_interval=2, tag='fun')
... def fun2():
...    time.sleep(0.2)
>>> for _ in range(3):
...    fun2()
[fun]-time per run averaged in the past 2 runs: 200.0 ms
```

```
>>> with TimeCounter(tag='fun3'):
...       time.sleep(0.3)
[fun3]-time per run averaged in the past 1 runs: 300.0 ms
```

print_time(*elapsed*)
print times per count.

Parameters **elapsed** (*Union[int, float]*) –

Return type None

collect_env	Collect the information of the running environments.
load_url	Loads the Torch serialized object at the given URL.
has_batch_norm	Detect whether model has a BatchNormalization layer.
is_norm	Check if a layer is a normalization layer.
mmcv_full_available	Check whether mmcv-full is installed.
tensor2imgs	Convert tensor to 3-channel images or 1-channel gray images.
TORCH_VERSION	A string with magic powers to compare to both Version and iterables! Prior to 1.10.0 torch.__version__ was stored as a str and so many did comparisons against torch.__version__ as if it were a str.
set_multi_processing	Set multi-processing related environment.
torch_meshgrid	A wrapper of torch.meshgrid to compat different PyTorch versions.
is_jit_tracing	

52.2 mmengine.utils.dl_utils.collect_env

mmengine.utils.dl_utils.**collect_env**()
Collect the information of the running environments.

Returns

The environment information. The following fields are contained.

- sys.platform: The variable of `sys.platform`.
- Python: Python version.
- CUDA available: Bool, indicating if CUDA is available.
- GPU devices: Device type of each GPU.
- CUDA_HOME (optional): The env var `CUDA_HOME`.
- NVCC (optional): NVCC version.
- GCC: GCC version, "n/a" if GCC is not installed.
- MSVC: Microsoft Virtual C++ Compiler version, Windows only.
- PyTorch: PyTorch version.
- PyTorch compiling details: The output of `torch.__config__.show()`.
- TorchVision (optional): TorchVision version.

- OpenCV (optional): OpenCV version.
- MMENGINE: MMENGINE version.

Return type dict

52.3 mmengine.utils.dl_utils.load_url

mmengine.utils.dl_utils.**load_url**(*url*, *model_dir=None*, *map_location=None*, *progress=True*, *check_hash=False*, *file_name=None*)

Loads the Torch serialized object at the given URL.

If downloaded file is a zip file, it will be automatically decompressed.

If the object is already present in *model_dir*, it's deserialized and returned. The default value of `model_dir` is `<hub_dir>/checkpoints` where `hub_dir` is the directory returned by `get_dir()`.

Parameters

- **url** (*str*) – URL of the object to download
- **model_dir** (*str, optional*) – directory in which to save the object
- **map_location** (*optional*) – a function or a dict specifying how to remap storage locations (see torch.load)
- **progress** (*bool, optional*) – whether or not to display a progress bar to stderr. Default: True
- **check_hash** (*bool, optional*) – If True, the filename part of the URL should follow the naming convention `filename-<sha256>.ext` where `<sha256>` is the first eight or more digits of the SHA256 hash of the contents of the file. The hash is used to ensure unique names and to verify the contents of the file. Default: False
- **file_name** (*str, optional*) – name for the downloaded file. Filename from `url` will be used if not set.

Return type Dict[str, Any]

Example

```
>>> state_dict = torch.hub.load_state_dict_from_url('https://s3.amazonaws.com/
↪pytorch/models/resnet18-5c106cde.pth')
```

52.4 mmengine.utils.dl_utils.has_batch_norm

mmengine.utils.dl_utils.**has_batch_norm**(*model*)

Detect whether model has a BatchNormalization layer.

Parameters **model** (*nn.Module*) – training model.

Returns whether model has a BatchNormalization layer

Return type bool

52.5 mmengine.utils.dl_utils.is_norm

mmengine.utils.dl_utils.**is_norm**(*layer*, *exclude=None*)
Check if a layer is a normalization layer.

Parameters

- **layer** (*nn.Module*) – The layer to be checked.
- **exclude** (*type, tuple[type], optional*) – Types to be excluded.

Returns Whether the layer is a norm layer.

Return type bool

52.6 mmengine.utils.dl_utils.mmcv_full_available

mmengine.utils.dl_utils.**mmcv_full_available**()
Check whether mmcv-full is installed.

Returns True if mmcv-full is installed else False.

Return type bool

52.7 mmengine.utils.dl_utils.tensor2imgs

mmengine.utils.dl_utils.**tensor2imgs**(*tensor*, *mean=None*, *std=None*, *to_bgr=True*)
Convert tensor to 3-channel images or 1-channel gray images.

Parameters

- **tensor** (*torch.Tensor*) – Tensor that contains multiple images, shape (N, C, H, W). C can be either 3 or 1. If C is 3, the format should be RGB.
- **mean** (*tuple[float], optional*) – Mean of images. If None, (0, 0, 0) will be used for tensor with 3-channel, while (0,) for tensor with 1-channel. Defaults to None.
- **std** (*tuple[float], optional*) – Standard deviation of images. If None, (1, 1, 1) will be used for tensor with 3-channel, while (1,) for tensor with 1-channel. Defaults to None.
- **to_bgr** (*bool*) – For the tensor with 3 channel, convert its format to BGR. For the tensor with 1 channel, it must be False. Defaults to True.

Returns A list that contains multiple images.

Return type list[np.ndarray]

52.8 mmengine.utils.dl_utils.TORCH_VERSION

mmengine.utils.dl_utils.**TORCH_VERSION = '1.13.1+cu117'**

A string with magic powers to compare to both Version and iterables! Prior to 1.10.0 torch.__version__ was stored as a str and so many did comparisons against torch.__version__ as if it were a str. In order to not break them we have TorchVersion which masquerades as a str while also having the ability to compare against both packaging.version.Version as well as tuples of values, eg. (1, 2, 1) .. rubric:: Examples

Comparing a TorchVersion object to a Version object TorchVersion('1.10.0a') > Version('1.10.0a')

Comparing a TorchVersion object to a Tuple object TorchVersion('1.10.0a') > (1, 2) # 1.2 TorchVersion('1.10.0a') > (1, 2, 1) # 1.2.1

Comparing a TorchVersion object against a string TorchVersion('1.10.0a') > '1.2' TorchVersion('1.10.0a') > '1.2.1'

52.9 mmengine.utils.dl_utils.set_multi_processing

mmengine.utils.dl_utils.**set_multi_processing**(*mp_start_method='fork'*, *opencv_num_threads=0*, *distributed=False*)

Set multi-processing related environment.

Parameters

- **mp_start_method** (*str*) – Set the method which should be used to start child processes. Defaults to 'fork'.
- **opencv_num_threads** (*int*) – Number of threads for opencv. Defaults to 0.
- **distributed** (*bool*) – True if distributed environment. Defaults to False.

Return type None

52.10 mmengine.utils.dl_utils.torch_meshgrid

mmengine.utils.dl_utils.**torch_meshgrid**(**tensors*)

A wrapper of torch.meshgrid to compat different PyTorch versions.

Since PyTorch 1.10.0a0, torch.meshgrid supports the arguments `indexing`. So we implement a wrapper here to avoid warning when using high-version PyTorch and avoid compatibility issues when using previous versions of PyTorch.

Parameters **tensors** (*List[Tensor]*) – List of scalars or 1 dimensional tensors.

Returns Sequence of meshgrid tensors.

Return type Sequence[Tensor]

52.11 mmengine.utils.dl_utils.is_jit_tracing

mmengine.utils.dl_utils.**is_jit_tracing**()

Return type bool

CHAPTER

FIFTYTHREE

CHANGELOG OF V0.X

53.1 v0.3.0 (11/02/2022)

53.1.1 New Features & Enhancements

- Support running on Ascend chip by @wangjiangben-hw in https://github.com/open-mmlab/mmengine/pull/572
- Support torch `ZeroRedundancyOptimizer` by @nijkah in https://github.com/open-mmlab/mmengine/pull/551
- Add non-blocking feature to `BaseDataPreprocessor` by @shenmishajing in https://github.com/open-mmlab/mmengine/pull/618
- Add documents for `clip_grad`, and support clip grad by value. by @HAOCHENYE in https://github.com/open-mmlab/mmengine/pull/513
- Add ROCm info when collecting env by @zhouzaida in https://github.com/open-mmlab/mmengine/pull/633
- Add a function to mark the deprecated function. by @HAOCHENYE in https://github.com/open-mmlab/mmengine/pull/609
- Call `register_all_modules` in `Registry.get()` by @HAOCHENYE in https://github.com/open-mmlab/mmengine/pull/541
- Deprecate `_save_to_state_dict` implemented in mmengine by @HAOCHENYE in https://github.com/open-mmlab/mmengine/pull/610
- Add `ignore_keys` in ConcatDataset by @BIGWangYuDong in https://github.com/open-mmlab/mmengine/pull/556

53.1.2 Docs

- Fix cannot show `changelog.md` in chinese documents. by @HAOCHENYE in https://github.com/open-mmlab/mmengine/pull/606
- Fix Chinese docs whitespaces by @C1rN09 in https://github.com/open-mmlab/mmengine/pull/521
- Translate installation and 15_min by @xin-li-67 in https://github.com/open-mmlab/mmengine/pull/629
- Refine chinese doc by @Tau-J in https://github.com/open-mmlab/mmengine/pull/516
- Add MMYOLO link in README by @Xiangxu-0103 in https://github.com/open-mmlab/mmengine/pull/634
- Add MMEngine logo in docs by @zhouzaida in https://github.com/open-mmlab/mmengine/pull/641
- Fix docstring of `BaseDataset` by @HAOCHENYE in https://github.com/open-mmlab/mmengine/pull/656

- Fix docstring and documentation used for `hub.get_model` by @zengyh1900 in https://github.com/open-mmlab/mmengine/pull/659
- Fix typo in `docs/zh_cn/advanced_tutorials/visualization.md` by @MambaWong in https://github.com/open-mmlab/mmengine/pull/616
- Fix typo docstring of `DefaultOptimWrapperConstructor` by @triple-Mu in https://github.com/open-mmlab/mmengine/pull/644
- Fix typo in advanced tutorial by @cxiang26 in https://github.com/open-mmlab/mmengine/pull/650
- Fix typo in `Config` docstring by @sanbuphy in https://github.com/open-mmlab/mmengine/pull/654
- Fix typo in `docs/zh_cn/tutorials/config.md` by @Xiangxu-0103 in https://github.com/open-mmlab/mmengine/pull/596
- Fix typo in `docs/zh_cn/tutorials/model.md` by @C1rN09 in https://github.com/open-mmlab/mmengine/pull/598

53.1.3 Bug Fixes

- Fix error calculation of `eta_min` in `CosineRestartParamScheduler` by @Z-Fran in https://github.com/open-mmlab/mmengine/pull/639
- Fix `BaseDataPreprocessor.cast_data` could not handle string data by @HAOCHENYE in https://github.com/open-mmlab/mmengine/pull/602
- Make `autocast` compatible with mps by @HAOCHENYE in https://github.com/open-mmlab/mmengine/pull/587
- Fix error format of log message by @HAOCHENYE in https://github.com/open-mmlab/mmengine/pull/508
- Fix error implementation of `is_model_wrapper` by @HAOCHENYE in https://github.com/open-mmlab/mmengine/pull/640
- Fix `VisBackend.add_config` is not called by @shenmishajing in https://github.com/open-mmlab/mmengine/pull/613
- Change `strict_load` of EMAHook to False by default by @HAOCHENYE in https://github.com/open-mmlab/mmengine/pull/642
- Fix `open` encoding problem of Config in Windows by @sanbuphy in https://github.com/open-mmlab/mmengine/pull/648
- Fix the total number of iterations in log is a float number. by @jbwang1997 in https://github.com/open-mmlab/mmengine/pull/604
- Fix `pip upgrade` CI by @HAOCHENYE in https://github.com/open-mmlab/mmengine/pull/622

53.1.4 New Contributors

- @shenmishajing made their first contribution in https://github.com/open-mmlab/mmengine/pull/618
- @Xiangxu-0103 made their first contribution in https://github.com/open-mmlab/mmengine/pull/596
- @Tau-J made their first contribution in https://github.com/open-mmlab/mmengine/pull/516
- @wangjiangben-hw made their first contribution in https://github.com/open-mmlab/mmengine/pull/572
- @triple-Mu made their first contribution in https://github.com/open-mmlab/mmengine/pull/644
- @sanbuphy made their first contribution in https://github.com/open-mmlab/mmengine/pull/648

- @Z-Fran made their first contribution in https://github.com/open-mmlab/mmengine/pull/639
- @BIGWangYuDong made their first contribution in https://github.com/open-mmlab/mmengine/pull/556
- @zengyh1900 made their first contribution in https://github.com/open-mmlab/mmengine/pull/659

53.2 v0.2.0 (11/10/2022)

53.2.1 New Features & Enhancements

- Add SMDDP backend and support running on AWS by @austinmw in https://github.com/open-mmlab/mmengine/pull/579
- Refactor `FileIO` but without breaking bc by @zhouzaida in https://github.com/open-mmlab/mmengine/pull/533
- Add test time augmentation base model by @HAOCHENYE in https://github.com/open-mmlab/mmengine/pull/538
- Use `torch.lerp\_()` to speed up EMA by @RangiLyu in https://github.com/open-mmlab/mmengine/pull/519
- Support converting `BN` to `SyncBN` by config by @HAOCHENYE in https://github.com/open-mmlab/mmengine/pull/506
- Support defining metric name in wandb backend by @okotaku in https://github.com/open-mmlab/mmengine/pull/509
- Add dockerfile by @zhouzaida in https://github.com/open-mmlab/mmengine/pull/347

53.2.2 Docs

- Fix API files of English documentation by @zhouzaida in https://github.com/open-mmlab/mmengine/pull/525
- Fix typo in `instance_data.py` by @Dai-Wenxun in https://github.com/open-mmlab/mmengine/pull/530
- Fix the docstring of the model sub-package by @zhouzaida in https://github.com/open-mmlab/mmengine/pull/573
- Fix a spelling error in docs/zh_cn by @cxiang26 in https://github.com/open-mmlab/mmengine/pull/548
- Fix typo in docstring by @MengzhangLI in https://github.com/open-mmlab/mmengine/pull/527
- Update `config.md` by @Zhengfei-0311 in https://github.com/open-mmlab/mmengine/pull/562

53.2.3 Bug Fixes

- Fix `LogProcessor` does not smooth loss if the name of loss doesn't start with `loss` by @liuyanyi in https://github.com/open-mmlab/mmengine/pull/539
- Fix failed to enable `detect_anomalous_params` in `MMSeparateDistributedDataParallel` by @HAOCHENYE in https://github.com/open-mmlab/mmengine/pull/588
- Fix CheckpointHook behavior unexpected if given `filename_tmpl` argument by @C1rN09 in https://github.com/open-mmlab/mmengine/pull/518
- Fix error argument sequence in `FSDP` by @HAOCHENYE in https://github.com/open-mmlab/mmengine/pull/520
- Fix uploading image in wandb backend @okotaku in https://github.com/open-mmlab/mmengine/pull/510

- Fix loading state dictionary in `EMAHook` by @okotaku in https://github.com/open-mmlab/mmengine/pull/507
- Fix circle import in `EMAHook` by @HAOCHENYE in https://github.com/open-mmlab/mmengine/pull/523
- Fix unit test could fail caused by `MultiProcessTestCase` by @HAOCHENYE in https://github.com/open-mmlab/mmengine/pull/535
- Remove unnecessary "if statement" in `Registry` by @MambaWong in https://github.com/open-mmlab/mmengine/pull/536
- Fix `_save_to_state_dict` by @HAOCHENYE in https://github.com/open-mmlab/mmengine/pull/542
- Support comparing NumPy array dataset meta in `Runner.resume` by @HAOCHENYE in https://github.com/open-mmlab/mmengine/pull/511
- Use `get` instead of `pop` to dump `runner_type` in `build_runner_from_cfg` by @nijkah in https://github.com/open-mmlab/mmengine/pull/549
- Upgrade pre-commit hooks by @zhouzaida in https://github.com/open-mmlab/mmengine/pull/576
- Delete the error comment in `registry.md` by @vansin in https://github.com/open-mmlab/mmengine/pull/514
- Fix Some out-of-date unit tests by @C1rN09 in https://github.com/open-mmlab/mmengine/pull/586
- Fix typo in `MMFullyShardedDataParallel` by @yhna940 in https://github.com/open-mmlab/mmengine/pull/569
- Update Github Action CI and CircleCI by @zhouzaida in https://github.com/open-mmlab/mmengine/pull/512
- Fix unit test in windows by @HAOCHENYE in https://github.com/open-mmlab/mmengine/pull/515
- Fix merge ci & multiprocessing unit test by @HAOCHENYE in https://github.com/open-mmlab/mmengine/pull/529

53.2.4 New Contributors

- @okotaku made their first contribution in https://github.com/open-mmlab/mmengine/pull/510
- @MengzhangLI made their first contribution in https://github.com/open-mmlab/mmengine/pull/527
- @MambaWong made their first contribution in https://github.com/open-mmlab/mmengine/pull/536
- @cxiang26 made their first contribution in https://github.com/open-mmlab/mmengine/pull/548
- @nijkah made their first contribution in https://github.com/open-mmlab/mmengine/pull/549
- @Zhengfei-0311 made their first contribution in https://github.com/open-mmlab/mmengine/pull/562
- @austinmw made their first contribution in https://github.com/open-mmlab/mmengine/pull/579
- @yhna940 made their first contribution in https://github.com/open-mmlab/mmengine/pull/569
- @liuyanyi made their first contribution in https://github.com/open-mmlab/mmengine/pull/539

CHAPTER

FIFTYFOUR

ENGLISH

CHAPTER

FIFTYFIVE

CHAPTER

FIFTYSIX

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

_ParamScheduler (*class in mmengine.optim*), 173

A

add_config() (*mmengine.visualization.BaseVisBackend method*), 241
add_config() (*mmengine.visualization.LocalVisBackend method*), 242
add_config() (*mmengine.visualization.TensorboardVisBackend method*), 244
add_config() (*mmengine.visualization.Visualizer method*), 234
add_config() (*mmengine.visualization.WandbVisBackend method*), 245
add_datasample() (*mmengine.visualization.Visualizer method*), 234
add_graph() (*mmengine.visualization.BaseVisBackend method*), 241
add_graph() (*mmengine.visualization.Visualizer method*), 234
add_image() (*mmengine.visualization.BaseVisBackend method*), 241
add_image() (*mmengine.visualization.LocalVisBackend method*), 243
add_image() (*mmengine.visualization.TensorboardVisBackend method*), 244
add_image() (*mmengine.visualization.Visualizer method*), 234
add_image() (*mmengine.visualization.WandbVisBackend method*), 245
add_params() (*mmengine.optim.DefaultOptimWrapperConstructor method*), 170
add_scalar() (*mmengine.visualization.BaseVisBackend method*), 241
add_scalar() (*mmengine.visualization.LocalVisBackend method*), 243
add_scalar() (*mmengine.visualization.TensorboardVisBackend method*), 244
add_scalar() (*mmengine.visualization.Visualizer method*), 234
add_scalar() (*mmengine.visualization.WandbVisBackend method*), 246
add_scalars() (*mmengine.visualization.BaseVisBackend method*), 241
add_scalars() (*mmengine.visualization.LocalVisBackend method*), 243
add_scalars() (*mmengine.visualization.TensorboardVisBackend method*), 244
add_scalars() (*mmengine.visualization.Visualizer method*), 235
add_scalars() (*mmengine.visualization.WandbVisBackend method*), 246
after_load_checkpoint() (*mmengine.hooks.EMAHook method*), 126
after_load_checkpoint() (*mmengine.hooks.Hook method*), 119
after_run() (*mmengine.hooks.Hook method*), 119
after_run() (*mmengine.hooks.LoggerHook method*), 129
after_test() (*mmengine.hooks.Hook method*), 120
after_test_epoch() (*mmengine.hooks.EMAHook method*), 126
after_test_epoch() (*mmengine.hooks.Hook method*), 120
after_test_epoch() (*mmengine.hooks.LoggerHook method*), 129
after_test_epoch() (*mmengine.hooks.RuntimeInfoHook method*), 131
after_test_iter() (*mmengine.hooks.Hook method*), 120
after_test_iter() (*mmengine.hooks.LoggerHook method*), 129
after_test_iter() (*mmengine.hooks.NaiveVisualizationHook method*), 130
after_train() (*mmengine.hooks.Hook method*), 120
after_train_epoch() (*mmengine.hooks.CheckpointHook method*), 125
after_train_epoch() (*mmengine.hooks.Hook method*), 120
after_train_epoch() (*mmengine.hooks.ParamSchedulerHook method*), 131
after_train_epoch()

(*mmengine.hooks.SyncBuffersHook method*), 133
after_train_iter() (*mmengine.hooks.CheckpointHook method*), 125
after_train_iter() (*mmengine.hooks.EMAHook method*), 126
after_train_iter() (*mmengine.hooks.Hook method*), 120
after_train_iter() (*mmengine.hooks.LoggerHook method*), 129
after_train_iter() (*mmengine.hooks.ParamSchedulerHook method*), 131
after_train_iter() (*mmengine.hooks.RuntimeInfoHook method*), 131
after_val() (*mmengine.hooks.Hook method*), 120
after_val_epoch() (*mmengine.hooks.CheckpointHook method*), 125
after_val_epoch() (*mmengine.hooks.EMAHook method*), 127
after_val_epoch() (*mmengine.hooks.Hook method*), 121
after_val_epoch() (*mmengine.hooks.LoggerHook method*), 129
after_val_epoch() (*mmengine.hooks.RuntimeInfoHook method*), 131
after_val_iter() (*mmengine.hooks.Hook method*), 121
after_val_iter() (*mmengine.hooks.LoggerHook method*), 129
all_gather() (*in module mmengine.dist*), 294
all_gather_object() (*in module mmengine.dist*), 295
all_items() (*mmengine.structures.BaseDataElement method*), 196
all_keys() (*mmengine.structures.BaseDataElement method*), 196
all_reduce() (*in module mmengine.dist*), 296
all_reduce_dict() (*in module mmengine.dist*), 297
all_reduce_params() (*in module mmengine.dist*), 297
all_values() (*mmengine.structures.BaseDataElement method*), 196
AmpOptimWrapper (*class in mmengine.optim*), 161
auto_argparser() (*mmengine.config.Config static method*), 88
autocast() (*in module mmengine.runner*), 114
avg_func() (*mmengine.model.BaseAveragedModel method*), 144
avg_func() (*mmengine.model.ExponentialMovingAverage method*), 145
avg_func() (*mmengine.model.MomentumAnnealingEMA method*), 145
avg_func() (*mmengine.model.StochasticWeightAverage method*), 146

B

backward() (*mmengine.optim.AmpOptimWrapper method*), 162
backward() (*mmengine.optim.OptimWrapper method*), 164
backward() (*mmengine.optim.OptimWrapperDict method*), 167
barrier() (*in module mmengine.dist*), 306
BaseAveragedModel (*class in mmengine.model*), 143
BaseDataElement (*class in mmengine.structures*), 193
BaseDataPreprocessor (*class in mmengine.model*), 140
BaseDataset (*class in mmengine.dataset*), 205
BaseFileHandler (*class in mmengine.fileio*), 274
BaseInit (*class in mmengine.model*), 153
BaseLoop (*class in mmengine.runner*), 107
BaseMetric (*class in mmengine.evaluator*), 190
BaseModel (*class in mmengine.model*), 137
BaseModule (*class in mmengine.model*), 135
BaseStorageBackend (*class in mmengine.fileio*), 247
BaseTTAModel (*class in mmengine.model*), 142
BaseVisBackend (*class in mmengine.visualization*), 241
before_run() (*mmengine.hooks.EMAHook method*), 127
before_run() (*mmengine.hooks.Hook method*), 121
before_run() (*mmengine.hooks.LoggerHook method*), 130
before_run() (*mmengine.hooks.RuntimeInfoHook method*), 132
before_save_checkpoint() (*mmengine.hooks.EMAHook method*), 127
before_save_checkpoint() (*mmengine.hooks.Hook method*), 121
before_test() (*mmengine.hooks.Hook method*), 121
before_test_epoch() (*mmengine.hooks.EMAHook method*), 127
before_test_epoch() (*mmengine.hooks.Hook method*), 121
before_test_iter() (*mmengine.hooks.Hook method*), 121
before_train() (*mmengine.hooks.CheckpointHook method*), 125
before_train() (*mmengine.hooks.EMAHook method*), 127
before_train() (*mmengine.hooks.Hook method*), 122
before_train() (*mmengine.hooks.IterTimerHook method*), 133
before_train() (*mmengine.hooks.RuntimeInfoHook method*), 132
before_train_epoch() (*mmengine.hooks.DistSamplerSeedHook method*), 132
before_train_epoch() (*mmengine.hooks.Hook method*), 122

before_train_epoch() (*mmengine.hooks.RuntimeInfoHook method*), 132
before_train_iter() (*mmengine.hooks.Hook method*), 122
before_train_iter() (*mmengine.hooks.RuntimeInfoHook method*), 132
before_val() (*mmengine.hooks.Hook method*), 122
before_val_epoch() (*mmengine.hooks.EMAHook method*), 127
before_val_epoch() (*mmengine.hooks.Hook method*), 122
before_val_iter() (*mmengine.hooks.Hook method*), 122
bias_init_with_prob() (*in module mmengine.model*), 156
broadcast() (*in module mmengine.dist*), 298
broadcast_object_list() (*in module mmengine.dist*), 300
build() (*mmengine.registry.Registry method*), 78
build_dataloader() (*mmengine.runner.Runner static method*), 94
build_evaluator() (*mmengine.runner.Runner method*), 95
build_from_cfg() (*in module mmengine.registry*), 83
build_iter_from_epoch() (*mmengine.optim.ConstantParamScheduler class method*), 175
build_iter_from_epoch() (*mmengine.optim.CosineAnnealingParamScheduler class method*), 177
build_iter_from_epoch() (*mmengine.optim.ExponentialParamScheduler class method*), 178
build_iter_from_epoch() (*mmengine.optim.LinearParamScheduler class method*), 180
build_iter_from_epoch() (*mmengine.optim.MultiStepParamScheduler class method*), 182
build_iter_from_epoch() (*mmengine.optim.OneCycleParamScheduler class method*), 184
build_iter_from_epoch() (*mmengine.optim.PolyParamScheduler class method*), 185
build_iter_from_epoch() (*mmengine.optim.StepParamScheduler class method*), 187
build_log_processor() (*mmengine.runner.Runner method*), 95
build_logger() (*mmengine.runner.Runner method*), 95
build_message_hub() (*mmengine.runner.Runner method*), 96
build_model() (*mmengine.runner.Runner method*), 96
build_model_from_cfg() (*in module mmengine.registry*), 84
build_optim_wrapper() (*in module mmengine.optim*), 171
build_optim_wrapper() (*mmengine.runner.Runner method*), 96
build_param_scheduler() (*mmengine.runner.Runner method*), 98
build_runner_from_cfg() (*in module mmengine.registry*), 84
build_scheduler_from_cfg() (*in module mmengine.registry*), 84
build_test_loop() (*mmengine.runner.Runner method*), 99
build_train_loop() (*mmengine.runner.Runner method*), 100
build_val_loop() (*mmengine.runner.Runner method*), 100
build_visualizer() (*mmengine.runner.Runner method*), 100

C

caffe2_xavier_init() (*in module mmengine.model*), 156
Caffe2XavierInit (*class in mmengine.model*), 153
call_command() (*in module mmengine.utils*), 313
call_hook() (*mmengine.runner.Runner method*), 100
callHandlers() (*mmengine.logging.MMLogger method*), 222
cast_data() (*mmengine.model.BaseDataPreprocessor method*), 140
cast_data_device() (*in module mmengine.dist*), 308
cat() (*mmengine.structures.InstanceData static method*), 201
check_file_exist() (*in module mmengine.utils*), 312
check_instance_created() (*mmengine.utils.ManagerMixin class method*), 310
check_prerequisites() (*in module mmengine.utils*), 321
check_time() (*in module mmengine.utils*), 324
CheckpointHook (*class in mmengine.hooks*), 124
CheckpointLoader (*class in mmengine.runner*), 111
ClassBalancedDataset (*class in mmengine.dataset*), 210
client (*mmengine.fileio.FileClient attribute*), 248
client_cfg (*mmengine.fileio.MemcachedBackend attribute*), 263
clone() (*mmengine.structures.BaseDataElement method*), 197
close() (*mmengine.visualization.BaseVisBackend method*), 242

close() (*mmengine.visualization.TensorboardVisBackend method*), 244
close() (*mmengine.visualization.Visualizer method*), 235
close() (*mmengine.visualization.WandbVisBackend method*), 246
collect_env() (*in module mmengine.utils.dl_utils*), 326
collect_results() (*in module mmengine.dist*), 301
collect_results_cpu() (*in module mmengine.dist*), 302
collect_results_gpu() (*in module mmengine.dist*), 302
Compose (*class in mmengine.dataset*), 210
compute_metrics() (*mmengine.evaluator.BaseMetric method*), 190
compute_metrics() (*mmengine.evaluator.DumpResults method*), 191
concat_list() (*in module mmengine.utils*), 320
ConcatDataset (*class in mmengine.dataset*), 211
Config (*class in mmengine.config*), 87
ConfigDict (*class in mmengine.config*), 89
constant_init() (*in module mmengine.model*), 156
ConstantInit (*class in mmengine.model*), 153
ConstantLR (*class in mmengine.optim*), 174
ConstantMomentum (*class in mmengine.optim*), 174
ConstantParamScheduler (*class in mmengine.optim*), 175
convert_sync_batchnorm() (*in module mmengine.model*), 160
copy_if_symlink_fails() (*in module mmengine.fileio*), 276
copy_if_symlink_fails() (*mmengine.fileio.LocalBackend method*), 253
copy_if_symlink_fails() (*mmengine.fileio.PetrelBackend method*), 264
copyfile() (*in module mmengine.fileio*), 277
copyfile() (*mmengine.fileio.LocalBackend method*), 253
copyfile() (*mmengine.fileio.PetrelBackend method*), 264
copyfile_from_local() (*in module mmengine.fileio*), 278
copyfile_from_local() (*mmengine.fileio.LocalBackend method*), 254
copyfile_from_local() (*mmengine.fileio.PetrelBackend method*), 265
copyfile_to_local() (*in module mmengine.fileio*), 278
copyfile_to_local() (*mmengine.fileio.LocalBackend method*), 254
copyfile_to_local() (*mmengine.fileio.PetrelBackend method*), 266
copytree() (*in module mmengine.fileio*), 279
copytree() (*mmengine.fileio.LocalBackend method*), 255
copytree() (*mmengine.fileio.PetrelBackend method*), 266
copytree_from_local() (*in module mmengine.fileio*), 280
copytree_from_local() (*mmengine.fileio.LocalBackend method*), 255
copytree_from_local() (*mmengine.fileio.PetrelBackend method*), 267
copytree_to_local() (*in module mmengine.fileio*), 280
copytree_to_local() (*mmengine.fileio.LocalBackend method*), 256
copytree_to_local() (*mmengine.fileio.PetrelBackend method*), 267
CosineAnnealingLR (*class in mmengine.optim*), 175
CosineAnnealingMomentum (*class in mmengine.optim*), 176
CosineAnnealingParamScheduler (*class in mmengine.optim*), 177
count_registered_modules() (*in module mmengine.registry*), 85
cpu() (*mmengine.model.BaseDataPreprocessor method*), 140
cpu() (*mmengine.model.BaseModel method*), 138
cpu() (*mmengine.structures.BaseDataElement method*), 197
cuda() (*mmengine.model.BaseDataPreprocessor method*), 140
cuda() (*mmengine.model.BaseModel method*), 138
cuda() (*mmengine.structures.BaseDataElement method*), 197
current() (*mmengine.logging.HistoryBuffer method*), 227

D

data (*mmengine.logging.HistoryBuffer property*), 227
data_preprocessor (*mmengine.model.BaseModel attribute*), 137
dataset_meta (*mmengine.evaluator.BaseMetric property*), 190
dataset_meta (*mmengine.evaluator.Evaluator property*), 189
dataset_meta (*mmengine.visualization.Visualizer property*), 235
db_path (*mmengine.fileio.LmdbBackend attribute*), 262
default_collate() (*in module mmengine.dataset*), 215

DefaultOptimWrapperConstructor (*class in mmengine.optim*), 169
defaults (*mmengine.optim.OptimWrapper property*), 164
DefaultSampler (*class in mmengine.dataset*), 213
DefaultScope (*class in mmengine.registry*), 81
deprecated_api_warning() (*in module mmengine.utils*), 321
deprecated_function() (*in module mmengine.utils*), 322
detach() (*mmengine.structures.BaseDataElement method*), 197
detect_anomalous_params() (*in module mmengine.model*), 159
deterministic (*mmengine.runner.Runner property*), 101
dict_from_file() (*in module mmengine.fileio*), 288
DictAction (*class in mmengine.config*), 89
digit_version() (*in module mmengine.utils*), 314
distributed (*mmengine.runner.Runner property*), 101
DistSamplerSeedHook (*class in mmengine.hooks*), 132
draw_bboxes() (*mmengine.visualization.Visualizer method*), 235
draw_binary_masks() (*mmengine.visualization.Visualizer method*), 235
draw_circles() (*mmengine.visualization.Visualizer method*), 236
draw_featmap() (*mmengine.visualization.Visualizer static method*), 236
draw_lines() (*mmengine.visualization.Visualizer method*), 237
draw_points() (*mmengine.visualization.Visualizer method*), 237
draw_polygons() (*mmengine.visualization.Visualizer method*), 238
draw_texts() (*mmengine.visualization.Visualizer method*), 239
dump() (*in module mmengine.fileio*), 275
dump() (*mmengine.config.Config method*), 88
dump_config() (*mmengine.runner.Runner method*), 101
DumpResults (*class in mmengine.evaluator*), 191

E

EMAHook (*class in mmengine.hooks*), 126
EmptyCacheHook (*class in mmengine.hooks*), 133
end_of_epoch() (*mmengine.hooks.Hook method*), 122
epoch (*mmengine.runner.EpochBasedTrainLoop property*), 108
epoch (*mmengine.runner.IterBasedTrainLoop property*), 109
epoch (*mmengine.runner.Runner property*), 101
EpochBasedTrainLoop (*class in mmengine.runner*), 108
evaluate() (*mmengine.evaluator.BaseMetric method*), 191
evaluate() (*mmengine.evaluator.Evaluator method*), 189
Evaluator (*class in mmengine.evaluator*), 189
every_n_epochs() (*mmengine.hooks.Hook method*), 123
every_n_inner_iters() (*mmengine.hooks.Hook method*), 123
every_n_train_iters() (*mmengine.hooks.Hook method*), 123
exists() (*in module mmengine.fileio*), 281
exists() (*mmengine.fileio.FileClient method*), 248
exists() (*mmengine.fileio.LocalBackend method*), 256
exists() (*mmengine.fileio.PetrelBackend method*), 268
experiment (*mmengine.visualization.BaseVisBackend property*), 242
experiment (*mmengine.visualization.LocalVisBackend property*), 243
experiment (*mmengine.visualization.TensorboardVisBackend property*), 245
experiment (*mmengine.visualization.WandbVisBackend property*), 246
experiment_name (*mmengine.runner.Runner property*), 101
ExponentialLR (*class in mmengine.optim*), 177
ExponentialMomentum (*class in mmengine.optim*), 178
ExponentialMovingAverage (*class in mmengine.model*), 144
ExponentialParamScheduler (*class in mmengine.optim*), 178

F

FileClient (*class in mmengine.fileio*), 248
filename (*mmengine.config.Config property*), 88
filter_data() (*mmengine.dataset.BaseDataset method*), 207
find_latest_checkpoint() (*in module mmengine.runner*), 112
fopen() (*in module mmengine.utils*), 312
forward() (*mmengine.model.BaseAveragedModel method*), 144
forward() (*mmengine.model.BaseDataPreprocessor method*), 140
forward() (*mmengine.model.BaseModel method*), 138
forward() (*mmengine.model.ImgDataPreprocessor method*), 142
from_cfg() (*mmengine.runner.Runner class method*), 101
fromfile() (*mmengine.config.Config static method*), 88
fromstring() (*mmengine.config.Config static method*), 88
full_init() (*mmengine.dataset.BaseDataset method*), 207

full_init() (*mmengine.dataset.ClassBalancedDataset method*), 210
full_init() (*mmengine.dataset.ConcatDataset method*), 212
full_init() (*mmengine.dataset.RepeatDataset method*), 212

G

gather() (*in module mmengine.dist*), 291
gather_object() (*in module mmengine.dist*), 293
generate_presigned_url() (*in module mmengine.fileio*), 281
generate_presigned_url() (*mmengine.fileio.PetrelBackend method*), 268
get() (*in module mmengine.fileio*), 282
get() (*mmengine.fileio.FileClient method*), 248
get() (*mmengine.fileio.HTTPBackend method*), 261
get() (*mmengine.fileio.LmdbBackend method*), 262
get() (*mmengine.fileio.LocalBackend method*), 257
get() (*mmengine.fileio.MemcachedBackend method*), 263
get() (*mmengine.fileio.PetrelBackend method*), 268
get() (*mmengine.registry.Registry method*), 78
get() (*mmengine.structures.BaseDataElement method*), 197
get_backend() (*in module mmengine.dist*), 304
get_backend() (*mmengine.visualization.Visualizer method*), 239
get_cat_ids() (*mmengine.dataset.BaseDataset method*), 207
get_cat_ids() (*mmengine.dataset.ClassBalancedDataset method*), 211
get_comm_device() (*in module mmengine.dist*), 308
get_config() (*in module mmengine.hub*), 219
get_current_instance() (*mmengine.logging.MessageHub class method*), 223
get_current_instance() (*mmengine.logging.MMLogger class method*), 222
get_current_instance() (*mmengine.registry.DefaultScope class method*), 82
get_current_instance() (*mmengine.utils.ManagerMixin class method*), 310
get_data_device() (*in module mmengine.dist*), 307
get_data_info() (*mmengine.dataset.BaseDataset method*), 207
get_data_info() (*mmengine.dataset.ClassBalancedDataset method*), 211
get_data_info() (*mmengine.dataset.ConcatDataset method*), 212
get_data_info() (*mmengine.dataset.RepeatDataset method*), 212
get_default_group() (*in module mmengine.dist*), 307
get_deprecated_model_names() (*in module mmengine.runner*), 112
get_device() (*in module mmengine.device*), 217
get_dist_info() (*in module mmengine.dist*), 303
get_external_models() (*in module mmengine.runner*), 112
get_file_backend() (*in module mmengine.fileio*), 282
get_git_hash() (*in module mmengine.utils*), 314
get_image() (*mmengine.visualization.Visualizer method*), 240
get_info() (*mmengine.logging.MessageHub method*), 223
get_installed_path() (*in module mmengine.utils*), 313
get_instance() (*mmengine.utils.ManagerMixin class method*), 311
get_instance() (*mmengine.visualization.Visualizer class method*), 240
get_last_value() (*mmengine.optim._ParamScheduler method*), 173
get_local_group() (*in module mmengine.dist*), 307
get_local_path() (*in module mmengine.fileio*), 283
get_local_path() (*mmengine.fileio.FileClient method*), 249
get_local_path() (*mmengine.fileio.HTTPBackend method*), 261
get_local_path() (*mmengine.fileio.LocalBackend method*), 257
get_local_path() (*mmengine.fileio.PetrelBackend method*), 269
get_local_rank() (*in module mmengine.dist*), 306
get_local_size() (*in module mmengine.dist*), 305
get_log_after_epoch() (*mmengine.runner.LogProcessor method*), 117
get_log_after_iter() (*mmengine.runner.LogProcessor method*), 117
get_lr() (*mmengine.optim.OptimWrapper method*), 164
get_lr() (*mmengine.optim.OptimWrapperDict method*), 167
get_max_cuda_memory() (*in module mmengine.device*), 217
get_metric_value() (*in module mmengine.evaluator*), 192
get_mmcls_models() (*in module mmengine.runner*), 112
get_model() (*in module mmengine.hub*), 220
get_momentum() (*mmengine.optim.OptimWrapper method*), 165
get_momentum() (*mmengine.optim.OptimWrapperDict*

method), 168
get_priority() (*in module mmengine.runner*), 118
get_rank() (*in module mmengine.dist*), 305
get_scalar() (*mmengine.logging.MessageHub method*), 224
get_state_dict() (*in module mmengine.runner*), 112
get_subset() (*mmengine.dataset.BaseDataset method*), 208
get_subset() (*mmengine.dataset.ClassBalancedDataset method*), 211
get_subset() (*mmengine.dataset.ConcatDataset method*), 212
get_subset() (*mmengine.dataset.RepeatDataset method*), 213
get_subset_() (*mmengine.dataset.BaseDataset method*), 208
get_subset_() (*mmengine.dataset.ClassBalancedDataset method*), 211
get_subset_() (*mmengine.dataset.ConcatDataset method*), 212
get_subset_() (*mmengine.dataset.RepeatDataset method*), 213
get_text() (*in module mmengine.fileio*), 283
get_text() (*mmengine.fileio.FileClient method*), 249
get_text() (*mmengine.fileio.HTTPBackend method*), 262
get_text() (*mmengine.fileio.LocalBackend method*), 257
get_text() (*mmengine.fileio.PetrelBackend method*), 269
get_torchvision_models() (*in module mmengine.runner*), 113
get_world_size() (*in module mmengine.dist*), 305

H

HardDiskBackend (*class in mmengine.fileio*), 253
has_batch_norm() (*in module mmengine.utils.dl_utils*), 327
has_method() (*in module mmengine.utils*), 322
HistoryBuffer (*class in mmengine.logging*), 227
Hook (*class in mmengine.hooks*), 119
hooks (*mmengine.runner.Runner property*), 101
HTTPBackend (*class in mmengine.fileio*), 261

I

ImgDataPreprocessor (*class in mmengine.model*), 141
import_modules_from_strings() (*in module mmengine.utils*), 323
infer_client() (*mmengine.fileio.FileClient class method*), 249
infer_scope() (*mmengine.registry.Registry static method*), 79
InfiniteSampler (*class in mmengine.dataset*), 214
init_cfg (*mmengine.model.BaseModel attribute*), 137
init_dist() (*in module mmengine.dist*), 304
init_local_group() (*in module mmengine.dist*), 304
init_weights() (*mmengine.model.BaseModule method*), 135
initialize() (*in module mmengine.model*), 156
initialize_count_status() (*mmengine.optim.OptimWrapper method*), 165
initialize_count_status() (*mmengine.optim.OptimWrapperDict method*), 168
inner_count (*mmengine.optim.OptimWrapper property*), 165
install_package() (*in module mmengine.utils*), 313
instance_name (*mmengine.utils.ManagerMixin property*), 311
InstanceData (*class in mmengine.structures*), 199
is_abs() (*in module mmengine.utils*), 312
is_cuda_available() (*in module mmengine.device*), 218
is_distributed() (*in module mmengine.dist*), 307
is_filepath() (*in module mmengine.utils*), 312
is_installed() (*in module mmengine.utils*), 314
is_jit_tracing() (*in module mmengine.utils.dl_utils*), 330
is_last_train_epoch() (*mmengine.hooks.Hook method*), 123
is_last_train_iter() (*mmengine.hooks.Hook method*), 123
is_list_of() (*in module mmengine.utils*), 318
is_main_process() (*in module mmengine.dist*), 306
is_method_overridden() (*in module mmengine.utils*), 322
is_mlu_available() (*in module mmengine.device*), 218
is_model_wrapper (*class in mmengine.model*), 152
is_mps_available() (*in module mmengine.device*), 218
is_norm() (*in module mmengine.utils.dl_utils*), 328
is_npu_available() (*in module mmengine.device*), 218
is_running (*mmengine.utils.Timer property*), 317
is_seq_of() (*in module mmengine.utils*), 319
is_str() (*in module mmengine.utils*), 319
is_tuple_of() (*in module mmengine.utils*), 318
isdir() (*in module mmengine.fileio*), 284
isdir() (*mmengine.fileio.FileClient method*), 250
isdir() (*mmengine.fileio.LocalBackend method*), 258
isdir() (*mmengine.fileio.PetrelBackend method*), 269
isfile() (*in module mmengine.fileio*), 284
isfile() (*mmengine.fileio.FileClient method*), 250
isfile() (*mmengine.fileio.LocalBackend method*), 258
isfile() (*mmengine.fileio.PetrelBackend method*), 270
items() (*mmengine.optim.OptimWrapperDict method*),

168
items() (*mmengine.structures.BaseDataElement method*), 197
iter (*mmengine.runner.EpochBasedTrainLoop property*), 108
iter (*mmengine.runner.IterBasedTrainLoop property*), 109
iter (*mmengine.runner.Runner property*), 101
iter_cast() (*in module mmengine.utils*), 319
IterBasedTrainLoop (*class in mmengine.runner*), 109
IterTimerHook (*class in mmengine.hooks*), 133

J

join_path() (*in module mmengine.fileio*), 284
join_path() (*mmengine.fileio.FileClient method*), 250
join_path() (*mmengine.fileio.LocalBackend method*), 258
join_path() (*mmengine.fileio.PetrelBackend method*), 270
JsonHandler (*class in mmengine.fileio*), 274

K

kaiming_init() (*in module mmengine.model*), 157
KaimingInit (*class in mmengine.model*), 154
keys() (*mmengine.optim.OptimWrapperDict method*), 168
keys() (*mmengine.structures.BaseDataElement method*), 197

L

label_to_onehot() (*mmengine.structures.LabelData static method*), 202
LabelData (*class in mmengine.structures*), 202
launcher (*mmengine.runner.Runner property*), 101
LinearLR (*class in mmengine.optim*), 179
LinearMomentum (*class in mmengine.optim*), 179
LinearParamScheduler (*class in mmengine.optim*), 180
list_cast() (*in module mmengine.utils*), 320
list_dir_or_file() (*in module mmengine.fileio*), 285
list_dir_or_file() (*mmengine.fileio.FileClient method*), 250
list_dir_or_file() (*mmengine.fileio.LocalBackend method*), 259
list_dir_or_file() (*mmengine.fileio.PetrelBackend method*), 270
list_from_file() (*in module mmengine.fileio*), 289
LmdbBackend (*class in mmengine.fileio*), 262
load() (*in module mmengine.fileio*), 276
load_checkpoint() (*in module mmengine.runner*), 113
load_checkpoint() (*mmengine.runner.CheckpointLoader class method*), 111
load_checkpoint() (*mmengine.runner.Runner method*), 101
load_data_list() (*mmengine.dataset.BaseDataset method*), 209
load_or_resume() (*mmengine.runner.Runner method*), 102
load_state_dict() (*in module mmengine.runner*), 113
load_state_dict() (*mmengine.logging.MessageHub method*), 224
load_state_dict() (*mmengine.optim._ParamScheduler method*), 173
load_state_dict() (*mmengine.optim.AmpOptimWrapper method*), 162
load_state_dict() (*mmengine.optim.OptimWrapper method*), 165
load_state_dict() (*mmengine.optim.OptimWrapperDict method*), 168
load_url() (*in module mmengine.utils.dl_utils*), 327
LocalBackend (*class in mmengine.fileio*), 253
LocalVisBackend (*class in mmengine.visualization*), 242
log_scalars (*mmengine.logging.MessageHub property*), 224
LoggerHook (*class in mmengine.hooks*), 128
LogProcessor (*class in mmengine.runner*), 115

M

ManagerMeta (*class in mmengine.utils*), 309
ManagerMixin (*class in mmengine.utils*), 310
master_only() (*in module mmengine.dist*), 306
max() (*mmengine.logging.HistoryBuffer method*), 227
max_epochs (*mmengine.runner.EpochBasedTrainLoop property*), 108
max_epochs (*mmengine.runner.IterBasedTrainLoop property*), 109
max_epochs (*mmengine.runner.Runner property*), 102
max_iters (*mmengine.runner.EpochBasedTrainLoop property*), 108
max_iters (*mmengine.runner.IterBasedTrainLoop property*), 109
max_iters (*mmengine.runner.Runner property*), 102
mean() (*mmengine.logging.HistoryBuffer method*), 228
MemcachedBackend (*class in mmengine.fileio*), 263
merge_dict() (*in module mmengine.model*), 159
merge_from_dict() (*mmengine.config.Config method*), 88
merge_preds() (*mmengine.model.BaseTTAModel method*), 143
MessageHub (*class in mmengine.logging*), 223
metainfo (*mmengine.dataset.BaseDataset property*), 209
metainfo (*mmengine.dataset.ClassBalancedDataset property*), 211
metainfo (*mmengine.dataset.ConcatDataset property*), 212

metainfo (*mmengine.dataset.RepeatDataset property*), 213
metainfo (*mmengine.structures.BaseDataElement property*), 197
metainfo_items() (*mmengine.structures.BaseDataElement method*), 197
metainfo_keys() (*mmengine.structures.BaseDataElement method*), 197
metainfo_values() (*mmengine.structures.BaseDataElement method*), 197
min() (*mmengine.logging.HistoryBuffer method*), 228
mkdir_or_exist() (*in module mmengine.utils*), 312
mmcv_full_available() (*in module mmengine.utils.dl_utils*), 328
MMDistributedDataParallel (*class in mmengine.model*), 147
MMFullyShardedDataParallel (*class in mmengine.model*), 150
MMLogger (*class in mmengine.logging*), 221
MMSeparateDistributedDataParallel (*class in mmengine.model*), 148
model_name (*mmengine.runner.Runner property*), 102
ModuleDict (*class in mmengine.model*), 136
ModuleList (*class in mmengine.model*), 136
MomentumAnnealingEMA (*class in mmengine.model*), 145
MultiStepLR (*class in mmengine.optim*), 180
MultiStepMomentum (*class in mmengine.optim*), 181
MultiStepParamScheduler (*class in mmengine.optim*), 181

N

NaiveVisualizationHook (*class in mmengine.hooks*), 130
new() (*mmengine.structures.BaseDataElement method*), 198
no_sync() (*mmengine.model.MMSeparateDistributedDataParallel method*), 149
normal_init() (*in module mmengine.model*), 158
NormalInit (*class in mmengine.model*), 154
npu() (*mmengine.model.BaseModel method*), 138
numpy() (*mmengine.structures.BaseDataElement method*), 198

O

offline_evaluate() (*mmengine.evaluator.Evaluator method*), 189
OneCycleLR (*class in mmengine.optim*), 182
OneCycleParamScheduler (*class in mmengine.optim*), 183
onehot_to_label() (*mmengine.structures.LabelData static method*), 202
optim_context() (*mmengine.optim.AmpOptimWrapper method*), 162
optim_context() (*mmengine.optim.OptimWrapper method*), 165
optim_context() (*mmengine.optim.OptimWrapperDict method*), 168
OptimWrapper (*class in mmengine.optim*), 162
OptimWrapperDict (*class in mmengine.optim*), 167
overwrite_default_scope() (*mmengine.registry.DefaultScope class method*), 82

P

param_groups (*mmengine.optim.OptimWrapper property*), 165
param_groups (*mmengine.optim.OptimWrapperDict property*), 168
ParamSchedulerHook (*class in mmengine.hooks*), 131
parse_data_info() (*mmengine.dataset.BaseDataset method*), 209
parse_losses() (*mmengine.model.BaseModel method*), 139
parse_uri_prefix() (*mmengine.fileio.FileClient static method*), 251
PetrelBackend (*class in mmengine.fileio*), 264
PickleHandler (*class in mmengine.fileio*), 274
PixelData (*class in mmengine.structures*), 202
PolyLR (*class in mmengine.optim*), 184
PolyMomentum (*class in mmengine.optim*), 184
PolyParamScheduler (*class in mmengine.optim*), 185
pop() (*mmengine.structures.BaseDataElement method*), 198
prepare_data() (*mmengine.dataset.BaseDataset method*), 209
PretrainedInit (*class in mmengine.model*), 154
pretty_text (*mmengine.config.Config property*), 89
print_log() (*in module mmengine.logging*), 229
print_time() (*mmengine.utils.dl_utils.TimeCounter method*), 326
print_value() (*mmengine.optim._ParamScheduler method*), 173
Priority (*class in mmengine.runner*), 117
process() (*mmengine.evaluator.BaseMetric method*), 191
process() (*mmengine.evaluator.DumpResults method*), 191
process() (*mmengine.evaluator.Evaluator method*), 190
ProgressBar (*class in mmengine.utils*), 315
pseudo_collate() (*in module mmengine.dataset*), 215
put() (*in module mmengine.fileio*), 286
put() (*mmengine.fileio.FileClient method*), 251
put() (*mmengine.fileio.LocalBackend method*), 259
put() (*mmengine.fileio.PetrelBackend method*), 271
put_text() (*in module mmengine.fileio*), 287
put_text() (*mmengine.fileio.FileClient method*), 251

put_text() (*mmengine.fileio.LocalBackend method*), 260
put_text() (*mmengine.fileio.PetrelBackend method*), 272

R

rank (*mmengine.runner.Runner property*), 102
register_backend() (*in module mmengine.fileio*), 273
register_backend() (*mmengine.fileio.FileClient class method*), 252
register_custom_hooks() (*mmengine.runner.Runner method*), 102
register_default_hooks() (*mmengine.runner.Runner method*), 102
register_handler() (*in module mmengine.fileio*), 274
register_hook() (*mmengine.runner.Runner method*), 103
register_hooks() (*mmengine.runner.Runner method*), 103
register_module() (*mmengine.registry.Registry method*), 79
register_scheme() (*mmengine.runner.CheckpointLoader class method*), 111
register_statistics() (*mmengine.logging.HistoryBuffer class method*), 228
Registry (*class in mmengine.registry*), 77
remove() (*in module mmengine.fileio*), 287
remove() (*mmengine.fileio.FileClient method*), 252
remove() (*mmengine.fileio.LocalBackend method*), 260
remove() (*mmengine.fileio.PetrelBackend method*), 272
RepeatDataset (*class in mmengine.dataset*), 212
requires_executable() (*in module mmengine.utils*), 323
requires_package() (*in module mmengine.utils*), 323
resume() (*mmengine.runner.Runner method*), 103
revert_sync_batchnorm() (*in module mmengine.model*), 160
rmtree() (*in module mmengine.fileio*), 288
rmtree() (*mmengine.fileio.LocalBackend method*), 261
rmtree() (*mmengine.fileio.PetrelBackend method*), 272
run() (*mmengine.runner.BaseLoop method*), 107
run() (*mmengine.runner.EpochBasedTrainLoop method*), 108
run() (*mmengine.runner.IterBasedTrainLoop method*), 109
run() (*mmengine.runner.TestLoop method*), 110
run() (*mmengine.runner.ValLoop method*), 110
run_epoch() (*mmengine.runner.EpochBasedTrainLoop method*), 108
run_iter() (*mmengine.runner.EpochBasedTrainLoop method*), 108
run_iter() (*mmengine.runner.IterBasedTrainLoop method*), 109
run_iter() (*mmengine.runner.TestLoop method*), 110
run_iter() (*mmengine.runner.ValLoop method*), 110
Runner (*class in mmengine.runner*), 91
runtime_info (*mmengine.logging.MessageHub property*), 224
RuntimeInfoHook (*class in mmengine.hooks*), 131

S

save_checkpoint() (*in module mmengine.runner*), 114
save_checkpoint() (*mmengine.runner.Runner method*), 104
scale_loss() (*mmengine.optim.OptimWrapper method*), 166
scale_lr() (*mmengine.runner.Runner method*), 104
scandir() (*in module mmengine.utils*), 312
scope_name (*mmengine.registry.DefaultScope property*), 82
seed (*mmengine.runner.Runner property*), 105
Sequential (*class in mmengine.model*), 136
server_list_cfg (*mmengine.fileio.MemcachedBackend attribute*), 263
set_data() (*mmengine.structures.BaseDataElement method*), 198
set_epoch() (*mmengine.dataset.DefaultSampler method*), 214
set_epoch() (*mmengine.dataset.InfiniteSampler method*), 214
set_field() (*mmengine.structures.BaseDataElement method*), 198
set_image() (*mmengine.visualization.Visualizer method*), 240
set_metainfo() (*mmengine.structures.BaseDataElement method*), 198
set_multi_processing() (*in module mmengine.utils.dl_utils*), 329
set_randomness() (*mmengine.runner.Runner method*), 105
setLevel() (*mmengine.logging.MMLogger method*), 222
setup_env() (*mmengine.runner.Runner method*), 105
shape (*mmengine.structures.PixelData property*), 203
should_sync() (*mmengine.optim.OptimWrapper method*), 166
should_update() (*mmengine.optim.OptimWrapper method*), 166
show() (*mmengine.visualization.Visualizer method*), 240
since_last_check() (*mmengine.utils.Timer method*), 317
since_start() (*mmengine.utils.Timer method*), 317
slice_list() (*in module mmengine.utils*), 320
split_scope_key() (*mmengine.registry.Registry static method*), 80
stack_batch() (*in module mmengine.model*), 160
start() (*mmengine.utils.Timer method*), 317

state_dict() (*mmengine.logging.MessageHub method*), 224
state_dict() (*mmengine.optim._ParamScheduler method*), 173
state_dict() (*mmengine.optim.AmpOptimWrapper method*), 162
state_dict() (*mmengine.optim.OptimWrapper method*), 166
state_dict() (*mmengine.optim.OptimWrapperDict method*), 168
statistics() (*mmengine.logging.HistoryBuffer method*), 228
step() (*mmengine.optim._ParamScheduler method*), 173
step() (*mmengine.optim.AmpOptimWrapper method*), 162
step() (*mmengine.optim.OptimWrapper method*), 166
step() (*mmengine.optim.OptimWrapperDict method*), 168
StepLR (*class in mmengine.optim*), 185
StepMomentum (*class in mmengine.optim*), 186
StepParamScheduler (*class in mmengine.optim*), 186
StochasticWeightAverage (*class in mmengine.model*), 146
switch_scope_and_registry() (*mmengine.registry.Registry method*), 80
symlink() (*in module mmengine.utils*), 313
sync_random_seed() (*in module mmengine.dist*), 299
SyncBuffersHook (*class in mmengine.hooks*), 133
sys_path (*mmengine.fileio.MemcachedBackend attribute*), 263

T

tensor2imgs() (*in module mmengine.utils.dl_utils*), 328
TensorboardVisBackend (*class in mmengine.visualization*), 243
test() (*mmengine.runner.Runner method*), 105
test_dataloader (*mmengine.runner.Runner property*), 105
test_evaluator (*mmengine.runner.Runner property*), 105
test_loop (*mmengine.runner.Runner property*), 105
test_step() (*mmengine.model.BaseModel method*), 139
test_step() (*mmengine.model.BaseTTAModel method*), 143
test_step() (*mmengine.model.MMDistributedDataParallel method*), 148
test_step() (*mmengine.model.MMFullyShardedDataParallel method*), 151
test_step() (*mmengine.model.MMSeparateDistributedDataParallel method*), 149
TestLoop (*class in mmengine.runner*), 110
text (*mmengine.config.Config property*), 89
TimeCounter (*class in mmengine.utils.dl_utils*), 325
Timer (*class in mmengine.utils*), 316
TimerError (*class in mmengine.utils*), 317
timestamp (*mmengine.runner.Runner property*), 106
to() (*mmengine.model.BaseDataPreprocessor method*), 140
to() (*mmengine.model.BaseModel method*), 139
to() (*mmengine.structures.BaseDataElement method*), 198
to_1tuple() (*in module mmengine.utils*), 320
to_2tuple() (*in module mmengine.utils*), 320
to_3tuple() (*in module mmengine.utils*), 321
to_4tuple() (*in module mmengine.utils*), 321
to_dict() (*mmengine.structures.BaseDataElement method*), 198
to_ntuple() (*in module mmengine.utils*), 321
to_tensor() (*mmengine.structures.BaseDataElement method*), 199
torch_meshgrid() (*in module mmengine.utils.dl_utils*), 329
TORCH_VERSION (*in module mmengine.utils.dl_utils*), 329
track_iter_progress() (*in module mmengine.utils*), 315
track_parallel_progress() (*in module mmengine.utils*), 315
track_progress() (*in module mmengine.utils*), 316
train() (*mmengine.model.MMSeparateDistributedDataParallel method*), 149
train() (*mmengine.runner.Runner method*), 106
train_dataloader (*mmengine.runner.Runner property*), 106
train_loop (*mmengine.runner.Runner property*), 106
train_step() (*mmengine.model.BaseModel method*), 139
train_step() (*mmengine.model.MMDistributedDataParallel method*), 148
train_step() (*mmengine.model.MMFullyShardedDataParallel method*), 151
train_step() (*mmengine.model.MMSeparateDistributedDataParallel method*), 150
traverse_registry_tree() (*in module mmengine.registry*), 85
trunc_normal_init() (*in module mmengine.model*), 158
TruncNormalInit (*class in mmengine.model*), 155
tuple_cast() (*in module mmengine.utils*), 320

U

uniform_init() (*in module mmengine.model*), 158
UniformInit (*class in mmengine.model*), 155
update() (*mmengine.logging.HistoryBuffer method*), 229
update() (*mmengine.structures.BaseDataElement method*), 199

update_info() (*mmengine.logging.MessageHub method*), 224
update_info_dict() (*mmengine.logging.MessageHub method*), 225
update_init_info() (*in module mmengine.model*), 158
update_parameters() (*mmengine.model.BaseAveragedModel method*), 144
update_params() (*mmengine.optim.OptimWrapper method*), 166
update_params() (*mmengine.optim.OptimWrapperDict method*), 168
update_scalar() (*mmengine.logging.MessageHub method*), 225
update_scalars() (*mmengine.logging.MessageHub method*), 226

V

val() (*mmengine.runner.Runner method*), 106
val_begin (*mmengine.runner.Runner property*), 106
val_dataloader (*mmengine.runner.Runner property*), 106
val_evaluator (*mmengine.runner.Runner property*), 106
val_interval (*mmengine.runner.Runner property*), 106
val_loop (*mmengine.runner.Runner property*), 106
val_step() (*mmengine.model.BaseModel method*), 139
val_step() (*mmengine.model.MMDistributedDataParallel method*), 148
val_step() (*mmengine.model.MMFullyShardedDataParallel method*), 152
val_step() (*mmengine.model.MMSeparateDistributedDataParallel method*), 150
ValLoop (*class in mmengine.runner*), 110
values() (*mmengine.optim.OptimWrapperDict method*), 169
values() (*mmengine.structures.BaseDataElement method*), 199
Visualizer (*class in mmengine.visualization*), 231

W

WandbVisBackend (*class in mmengine.visualization*), 245
weights_to_cpu() (*in module mmengine.runner*), 114
work_dir (*mmengine.runner.Runner property*), 106
worker_init_fn() (*in module mmengine.dataset*), 215
world_size (*mmengine.runner.Runner property*), 106
wrap_model() (*mmengine.runner.Runner method*), 106

X

xavier_init() (*in module mmengine.model*), 158
XavierInit (*class in mmengine.model*), 155

Y

YamlHandler (*class in mmengine.fileio*), 274

Z

zero_grad() (*mmengine.optim.OptimWrapper method*), 166
zero_grad() (*mmengine.optim.OptimWrapperDict method*), 169