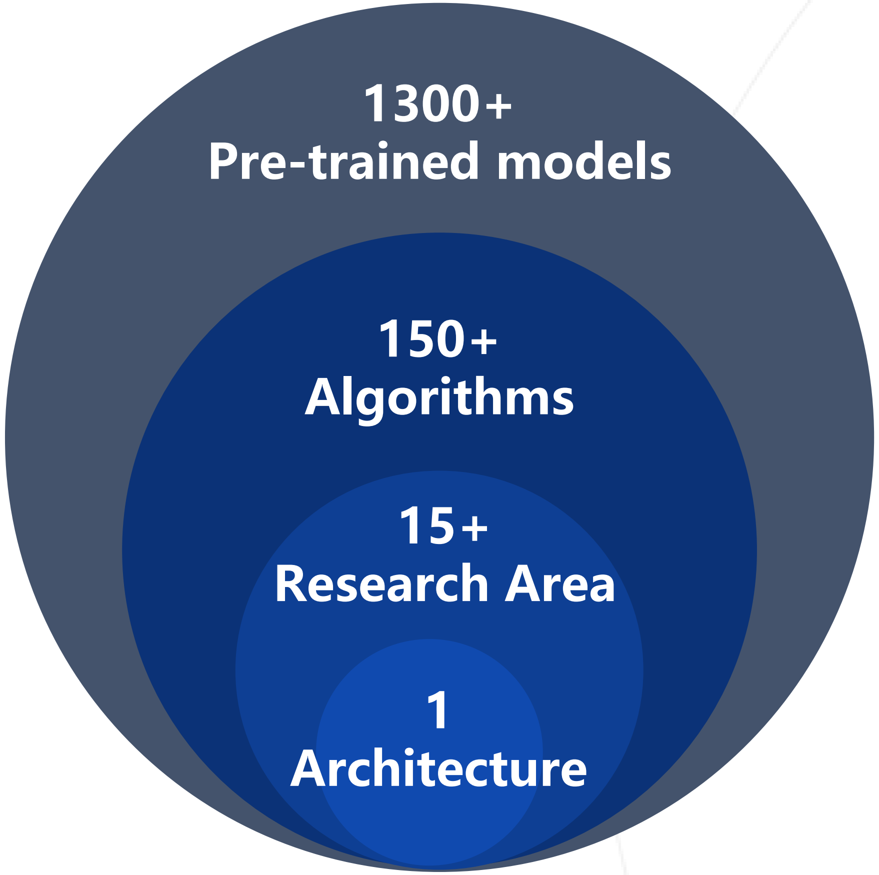
OpenMMLab

Introduction to OpenMMLab:
An Open-source Algorithm Platform for Computer Vision

Kai Chen

Overview



1 Architecture

- A unified architecture for all codebases

15+ Research Area

- Cover various areas for computer vision research

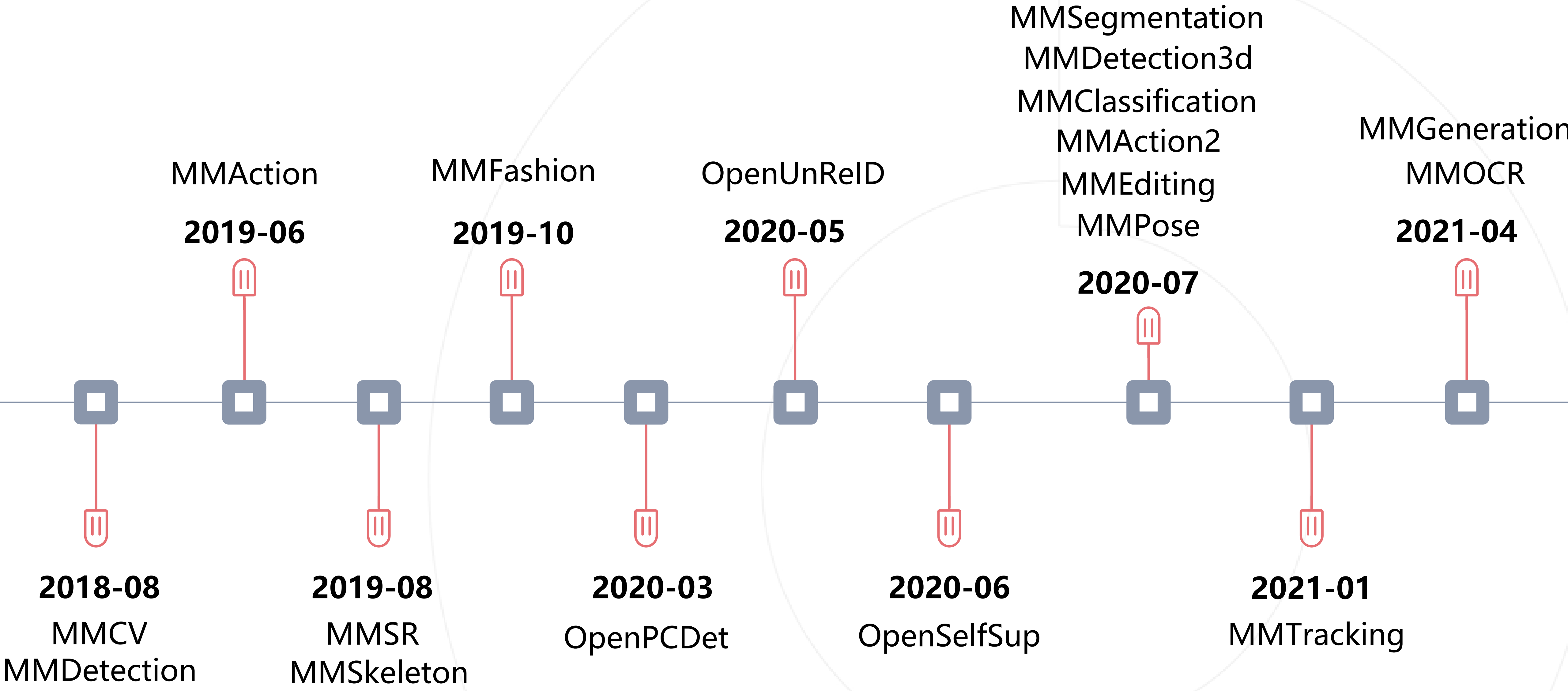
150+ Algorithms

- Implement SOTA algorithms

1300+ Pretrained Models

- Benchmark and out-of-box usage

History



2018-08
MMCV
MMDetection

2019-06
MMAction

2019-08
MMSR
MMSkeleton

2019-10
MMFashion

2020-03
OpenPCDet

2020-05
OpenUnReID

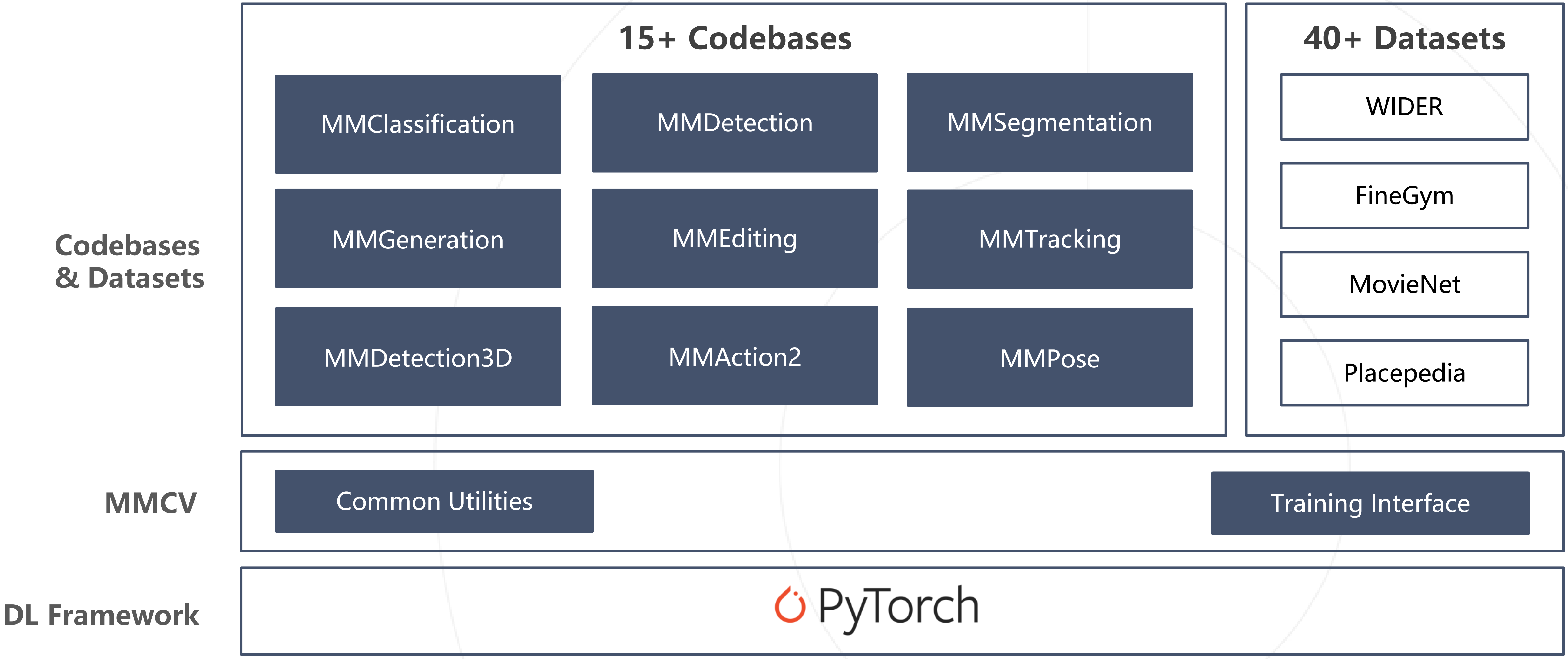
2020-06
OpenSelfSup

2020-07
MMSegmentation
MMDetection3d
MMClassification
MMAction2
MMEditing
MMPose

2021-01
MMTracking

2021-04
MMGeneration
MMOCR

Framework



Codebases & Datasets

15+ Codebases

MMClassification

MMDetection

MMSegmentation

MMGeneration

MMEditing

MMTracking

MMDetection3D

MMAction2

MMPose

40+ Datasets

WIDER

FineGym

MovieNet

Placepedia

MMCV

Common Utilities

Training Interface

DL Framework

PyTorch

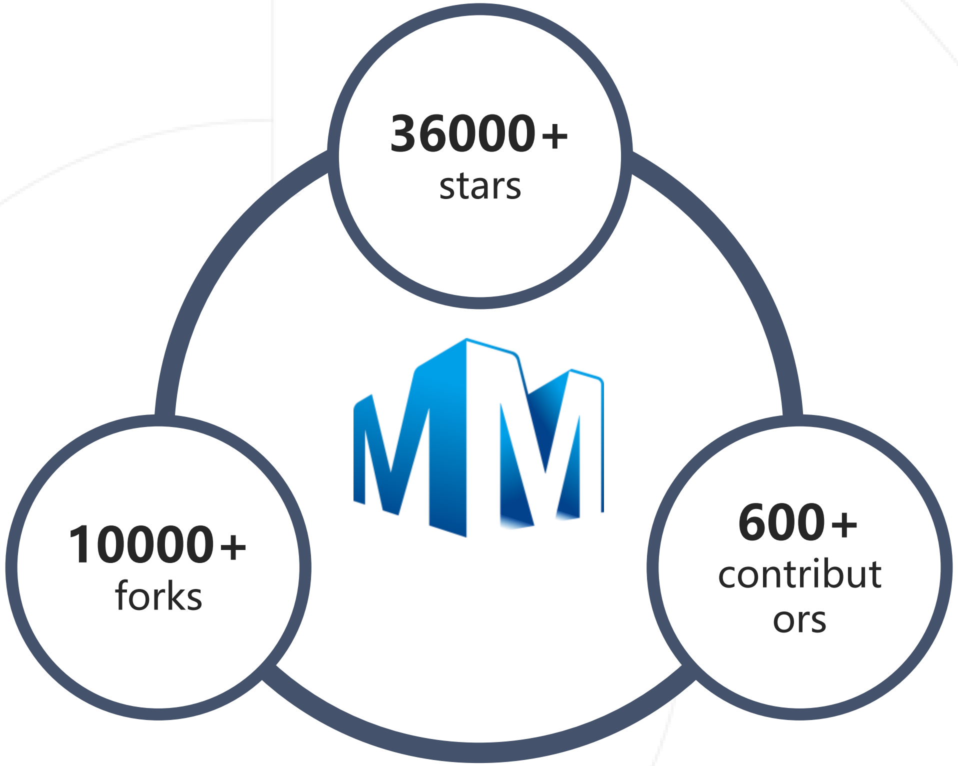
Community Impact

GitHub Statistics

- More than 36k stars
- 600+ contributors from academia and industry
- More than 800k page views of GitHub pages of all projects

Academic Impact

- MMDetection gained more than **450** citations, and selected as PapersWithCode 2020 Top10 Trending Libraries
- Many codebases such as MMDetection, MMDetection3D, MMSegmentation, MMPose are used by winners of various challenges
- More than **35** papers in CVPR 2021 adopt OpenMMLab projects as their implementation or benchmark



36000+
stars

10000+
forks

600+
contribut
ors

Major Features

- **Unified benchmark**: Provide a fair research platform and various baselines
- **Modular design**: Fast to develop and try new components
- **High-quality implementation**: Efficiency, accuracy, code style

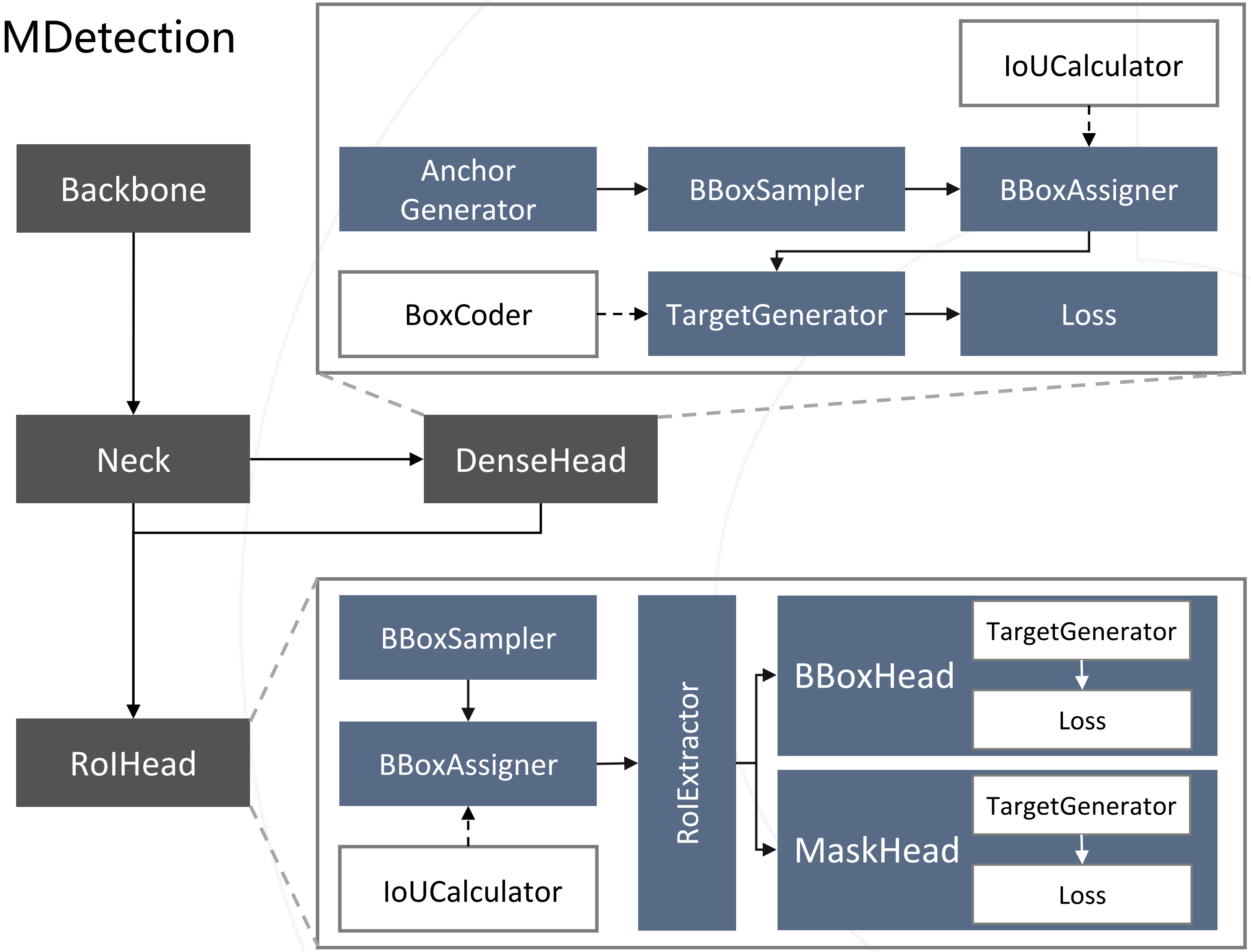
Unified Benchmark

		Paper A	Paper B	Paper C
1	backbone	standard ResNet torchvision pretrain	ResNet variant third-party pretrain	ResNet variant third-party pretrain
2	augmentation	769x769 crop horizontal flip	769x769 crop horizontal flip color jittor	512x1024 crop horizontal flip color jittor
3	method	method A	method B	method C
4	training schedule	200 epoch initial LR 0.1 poly	120k iteration initial LR 0.01 step	100 epoch initial LR 0.01 poly
5	inference	multi-scale crop	single-scale crop	single-scale whole image

mIoU(A) > mIoU(B) > mIoU(C)

Modular Design

Example of MMDetection



Architecture Design

Core concepts for training

- Runner&Hook
- Registry

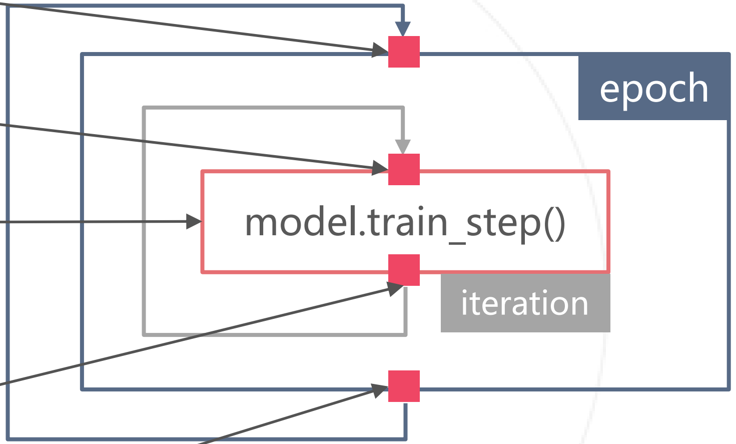
Concept: Runner&Hook

```
model = torch.nn.parallel.DistributedDataParallel(SomeNet(), device_ids=[args.gpu])
optimizer = torch.optim.SGD(...)
train_loader = torch.utils.data.DataLoader(...)

def adjust_learning_rate():
    pass

def record_and_log_loss():
    pass

for epoch in range(args.epochs):

    adjust_learning_rate(optimizer, epoch, args)

    # train for one epoch
    for i, (images, target) in enumerate(train_loader):
        # measure data loading time
        data_time.update(time.time() - end)

        # compute output
        output = model(images)
        loss = criterion(output, target)

        # compute gradient and do SGD step
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # measure accuracy and record loss
        record_and_log_loss(loss)

        # measure elapsed time
        batch_time.update(time.time() - end)
        end = time.time()

        # print progress
        if i % args.print_freq == 0:
            progress.display(i)

    # evaluate on validation set
    save_checkpoint()
```

ImageNet Example

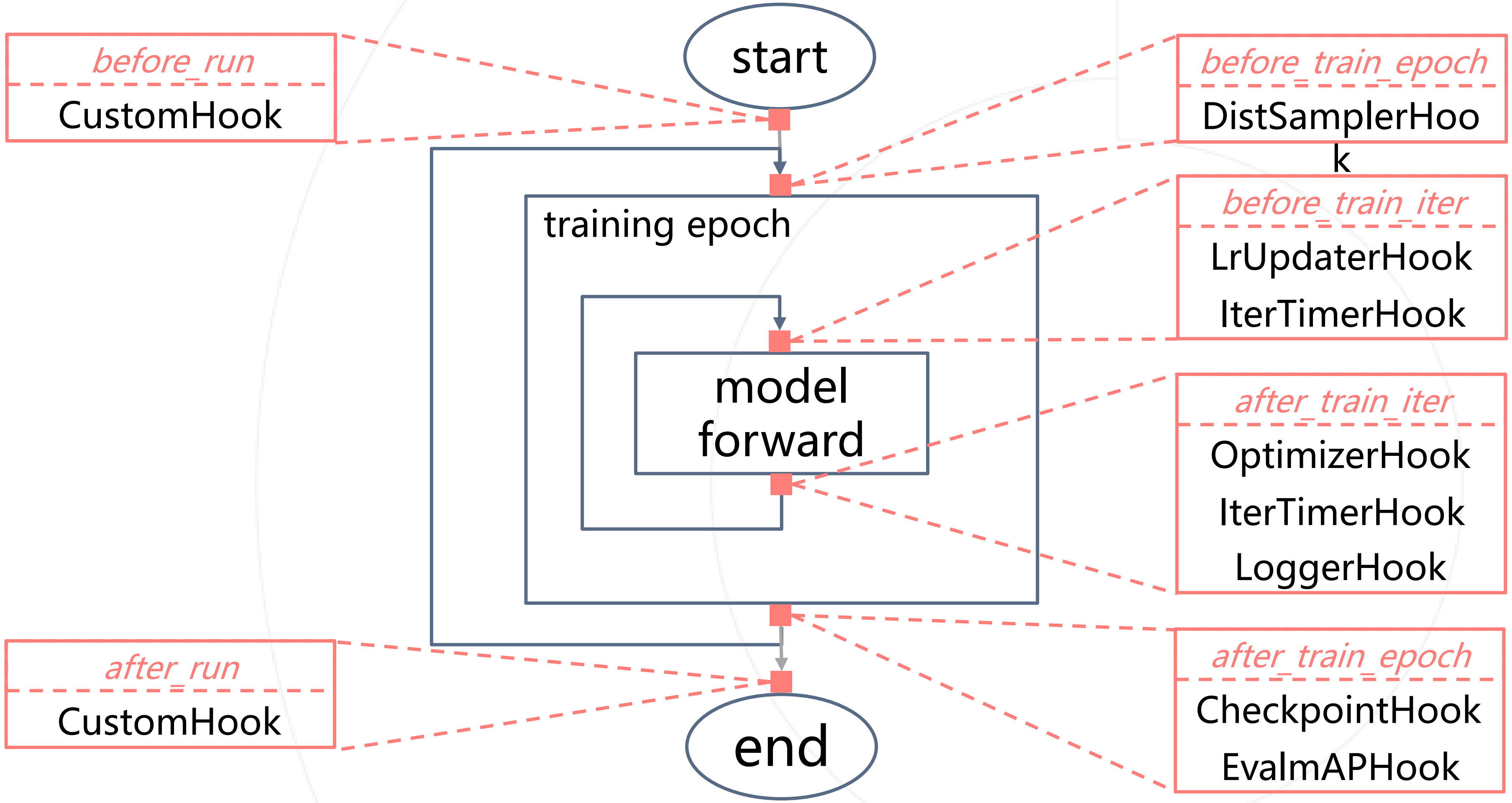
Runner: execution loop and core logic (fetch data and forward model)

Hook: custom logic and facilities (logging, visualization, lr scheduler, checkpointing, etc)

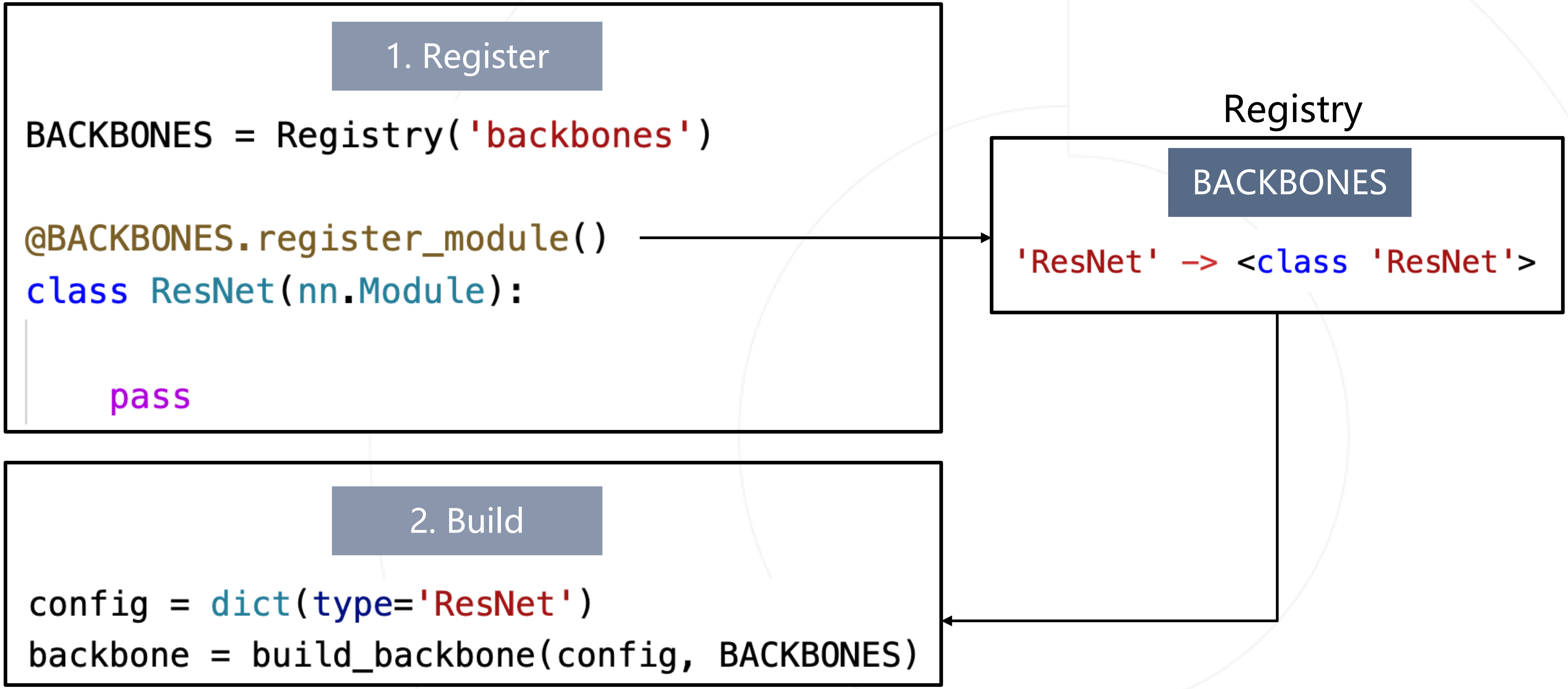


Concept: Runner&Hook

Another example



Concept: Registry



Build an instance with custom configs

1. Register

```
BACKBONES = Registry('backbones')

@BACKBONES.register_module()
class ResNet(nn.Module):

    pass
```

Registry

BACKBONES

```
'ResNet' -> <class 'ResNet'>
```

2. Build

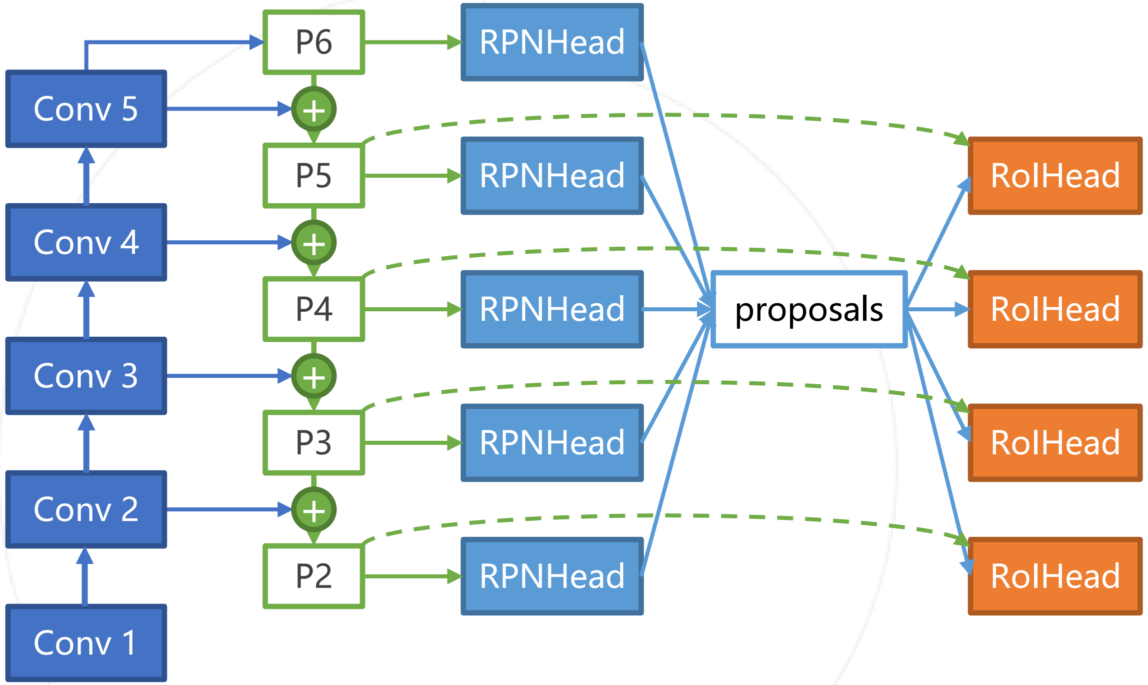
```
config = dict(type='ResNet')
backbone = build_backbone(config, BACKBONES)
```

Concept: Registry

Config

```
model = dict(
   type='FasterRCNN',
   pretrained='torchvision://resnet50',
   backbone=dict(
      type='ResNet',
      depth=50,
      ...),
   neck=dict(
      type='FPN',
      ...),
   rpn_head=dict(
      type='RPNHead',
      ...),
   roi_head=dict(
      type='StandardRoIHead',
      bbox_roi_extractor=dict(
         type='SingleRoIExtractor',
         ...),
      bbox_head=dict(
         type='Shared2FCBBoxHead',
         ...))
)
```

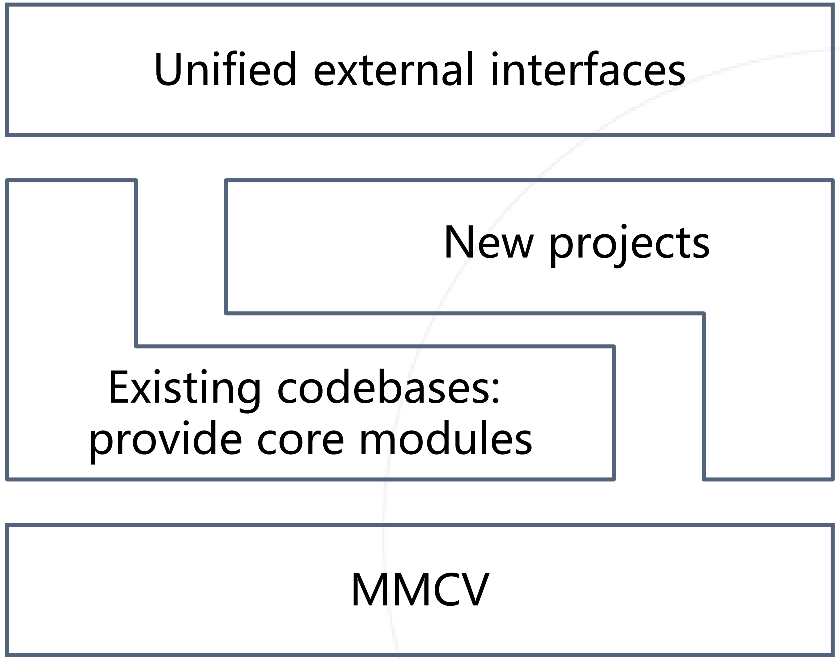
Module



Develop with OpenMMLab Codebases

1. Develop a new algorithm based on some existing codebase
2. Develop a new codebase based on existing codebases
3. Develop a new codebase following the architecture design of OpenMMLab

Develop with OpenMMLab Codebases



Unified external interfaces

New projects

Existing codebases:
provide core modules

MMCV