基于MMdetection3D配置BEV3d检测代码重现分析

目录

[简介 2](#_Toc175321380)

[环境搭建 4](#_Toc175321381)

[PointPillar 5](#_Toc175321382)

[Model定义 5](#_Toc175321383)

[检测实际代码分析 11](#_Toc175321384)

[Data定义 11](#_Toc175321385)

[Schedule定义 11](#_Toc175321386)

[Runtime定义 11](#_Toc175321387)

[Detr3d 12](#_Toc175321388)

[Model定义 12](#_Toc175321389)

[检测实际代码分析 12](#_Toc175321390)

[Data定义 12](#_Toc175321391)

[Schedule定义 12](#_Toc175321392)

[Runtime定义 12](#_Toc175321393)

[Futr3d 激光only 12](#_Toc175321394)

[Model定义 12](#_Toc175321395)

[检测实际代码分析 13](#_Toc175321396)

[Data定义 13](#_Toc175321397)

[Schedule定义 13](#_Toc175321398)

[Runtime定义 13](#_Toc175321399)

[Futr3d 激光视觉融合 13](#_Toc175321400)

[Model定义 13](#_Toc175321401)

[检测实际代码分析 13](#_Toc175321402)

[Data定义 13](#_Toc175321403)

[Schedule定义 13](#_Toc175321404)

[Runtime定义 13](#_Toc175321405)

# 简介

本文档适用于MMdetection3d 1.0.0版本，首先以PointPillar的实现为例子，通过整个代码的解释，分析整个MMdetection3d的框架与互相调用关系，之后再以detr3d为例分析自己如何构建一个plugin代码，以及BEV3d检测的一般流程。

MMdetection3d通过configs配置文件，实现对整个三维检测框架的实现，一般由以下4个部分组成

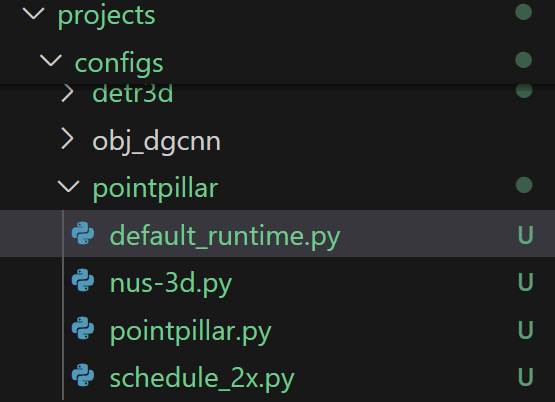
1 model:核心组件，通过脚本定义了模型的参数

2 data: 定义了数据集及其调用的相关设定

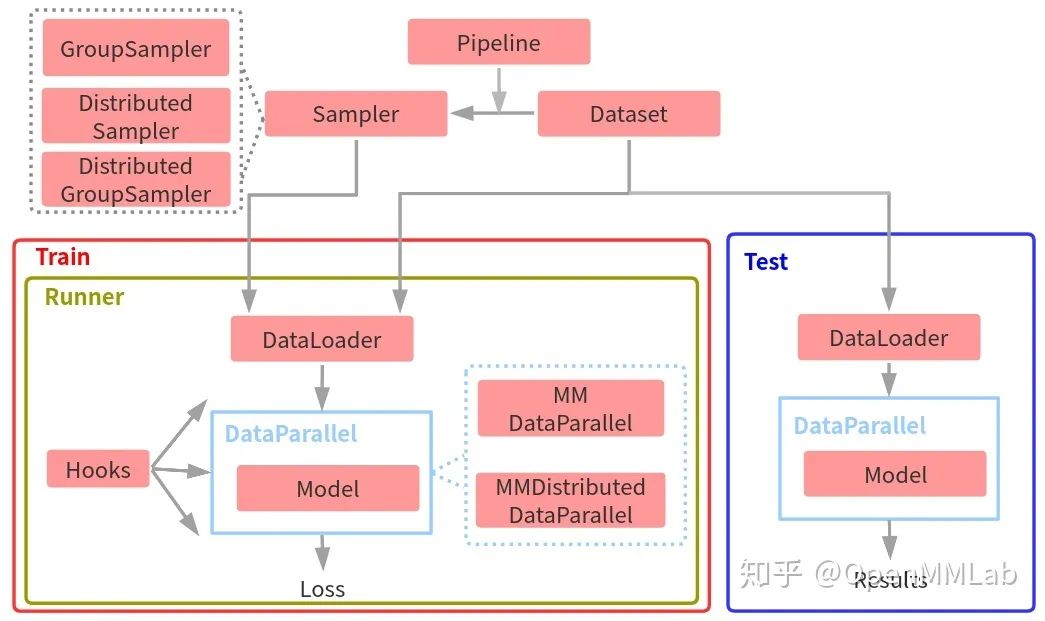
3 schedule: 训练optimizer相关设定

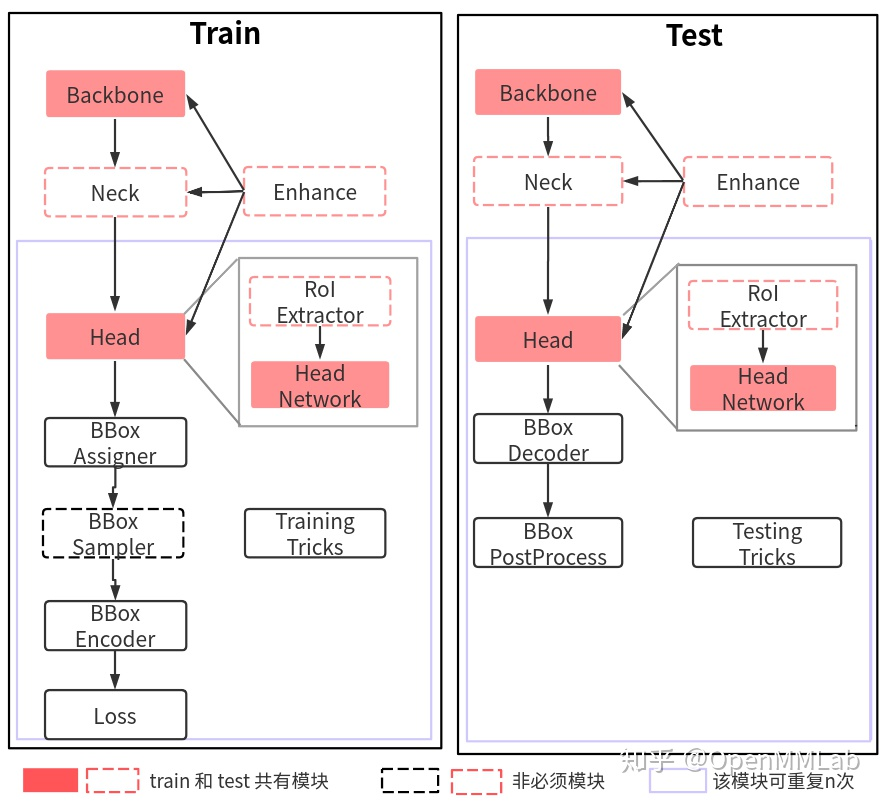
4 default\_runtime:运行环境设定，一般直接采用default\_runtime.py

虽然我们也可以将所有配置写在一个文件中，但是分成四个文件更有助于我们对框架不同部分的解耦与理解



图表 1 configs文件组成





# 环境搭建

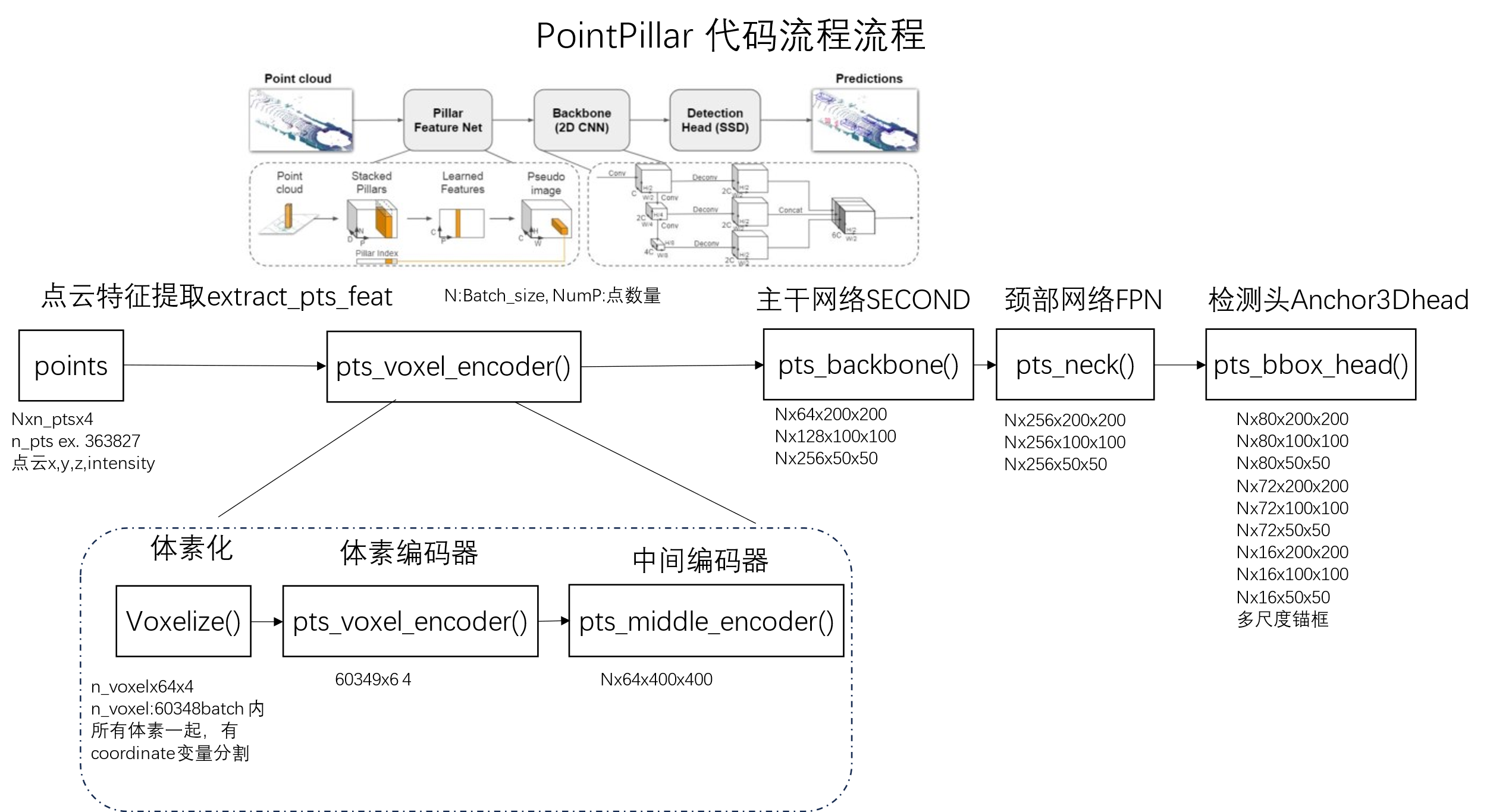
# PointPillar

在这一部分，我们以较为简单的PointPillar为例，数据集采用Nuscenes数据集，介绍一个典型的3D检测任务的框架

在训练过程中，我们采用命令

Python tools/train.py projects/configs/pointpillar/pointpillar.py –gpus 1

其中tools/train.py 是mmdetection3d模块的入口，而pointpillar.py指向了配置文件



图表 2 pointpillar模型代码流程

tools/train.py里有以下几个模块

## Model定义

Model是整个config文件中，最核心的部分，也是我们主要建立的部分，它指定了模型的各个环节，从而调用不同的对象并进行初始化，从而构建网络。针对激光3d检测网络，需要设定以下几个部分，

pts\_voxel\_layer:对点云数据进行体素化

pts\_voxel\_encoder: 体素编码

pts\_middle\_encoder:中间编码

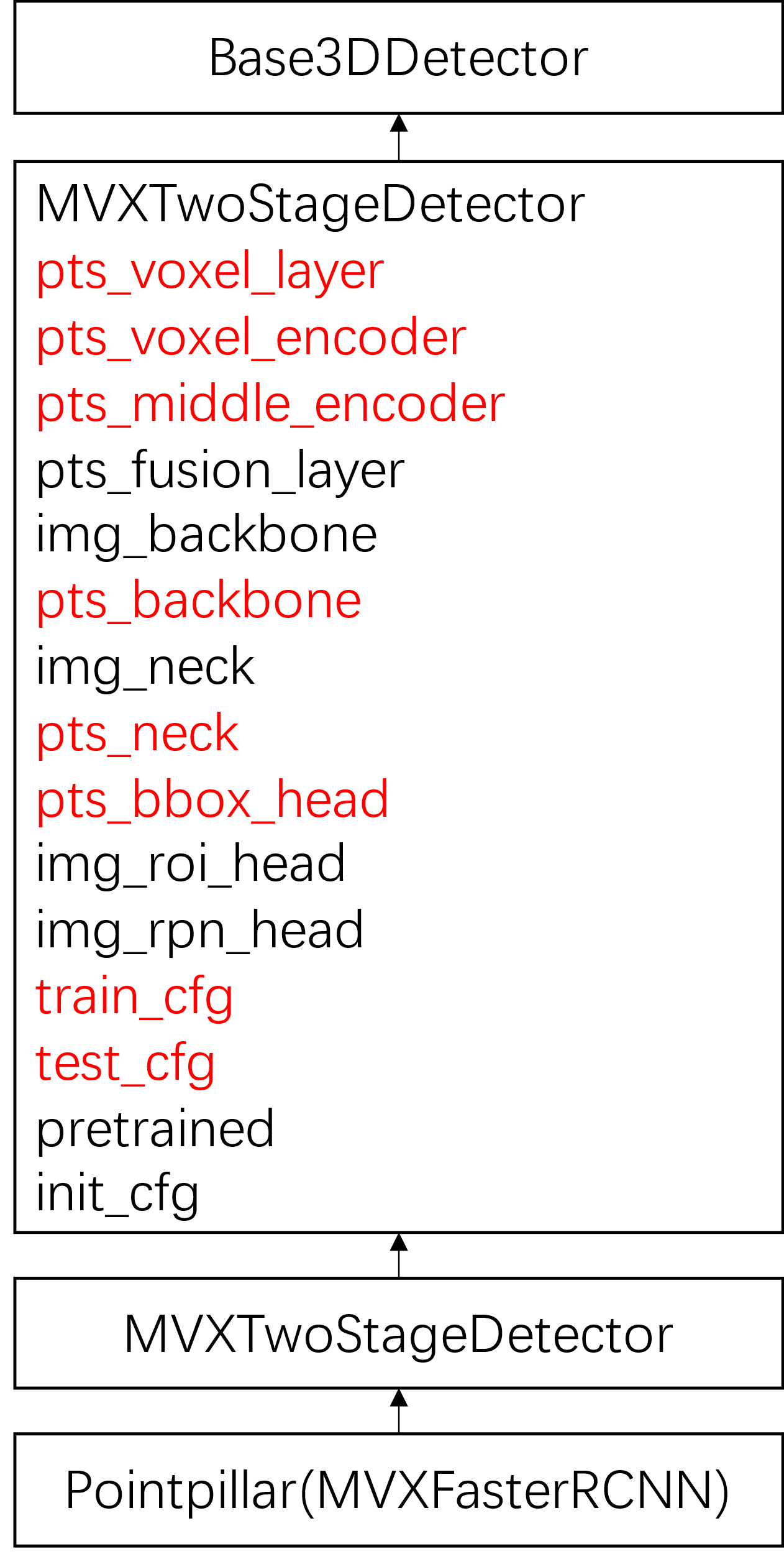
pts\_backbone: 体素特征提取

pts\_neck:颈部网络

pts\_bbox\_head: 检测头

train\_cfg: 训练配置

test\_cfg: 测试配置



图表 3 pointpillar模型为MVXFasterRCNN，以及其需要初始化的部件

注意，在单卡模式下，需要对pointpillar.py做一定的修改，防止出现多线程错误。需要将代码内所有的norm\_cfg修改成

norm\_cfg=dict(type='BN1d'

一共有三处

norm\_cfg=dict(type='naiveSyncBN1d', eps=1e-3, momentum=0.01))改为

norm\_cfg=dict(type='BN1d', eps=1e-3, momentum=0.01))

norm\_cfg=dict(type='naiveSyncBN2d', eps=1e-3, momentum=0.01),改为

norm\_cfg=dict(type='BN2d', eps=1e-3, momentum=0.01),

norm\_cfg=dict(type='naiveSyncBN1d', eps=1e-3, momentum=0.01))改为

norm\_cfg=dict(type='BN1d', eps=1e-3, momentum=0.01))

通过对pointpillar.py的分析，我们可以看到它采用了MVXFasterRCNN类，实现检测器。该检测器内部有如下几个部分

#

\_base\_ = [

    './nus-3d.py', './schedule\_2x.py',

    './default\_runtime.py'

]

# model settings

# Voxel size for voxel encoder

# Usually voxel size is changed consistently with the point cloud range

# If point cloud range is modified, do remember to change all related

# keys in the config.

voxel\_size = [0.25, 0.25, 8]

model = dict(

    type='MVXFasterRCNN',

    pts\_voxel\_layer=dict(

        max\_num\_points=64,

        point\_cloud\_range=[-50, -50, -5, 50, 50, 3],

        voxel\_size=voxel\_size,

        max\_voxels=(30000, 40000)),

    pts\_voxel\_encoder=dict(

        type='HardVFE',

        in\_channels=4,

        feat\_channels=[64, 64],

        with\_distance=False,

        voxel\_size=voxel\_size,

        with\_cluster\_center=True,

        with\_voxel\_center=True,

        point\_cloud\_range=[-50, -50, -5, 50, 50, 3],

        norm\_cfg=dict(type='BN1d', eps=1e-3, momentum=0.01)),

    pts\_middle\_encoder=dict(

        type='PointPillarsScatter', in\_channels=64, output\_shape=[400, 400]),

    pts\_backbone=dict(

        type='SECOND',

        in\_channels=64,

        norm\_cfg=dict(type='BN2d', eps=1e-3, momentum=0.01),

        layer\_nums=[3, 5, 5],

        layer\_strides=[2, 2, 2],

        out\_channels=[64, 128, 256]),

    pts\_neck=dict(

        type='FPN',

        norm\_cfg=dict(type='BN2d', eps=1e-3, momentum=0.01),

        act\_cfg=dict(type='ReLU'),

        in\_channels=[64, 128, 256],

        out\_channels=256,

        start\_level=0,

        num\_outs=3),

    pts\_bbox\_head=dict(

        type='Anchor3DHead',

        num\_classes=10,

        in\_channels=256,

        feat\_channels=256,

        use\_direction\_classifier=True,

        anchor\_generator=dict(

            type='AlignedAnchor3DRangeGenerator',

            ranges=[[-50, -50, -1.8, 50, 50, -1.8]],

            scales=[1, 2, 4],

            sizes=[

                [0.8660, 2.5981, 1.],  # 1.5/sqrt(3)

                [0.5774, 1.7321, 1.],  # 1/sqrt(3)

                [1., 1., 1.],

                [0.4, 0.4, 1],

            ],

            custom\_values=[0, 0],

            rotations=[0, 1.57],

            reshape\_out=True),

        assigner\_per\_size=False,

        diff\_rad\_by\_sin=True,

        dir\_offset=0.7854,  # pi/4

        dir\_limit\_offset=0,

        bbox\_coder=dict(type='DeltaXYZWLHRBBoxCoder', code\_size=9),

        loss\_cls=dict(

            type='FocalLoss',

            use\_sigmoid=True,

            gamma=2.0,

            alpha=0.25,

            loss\_weight=1.0),

        loss\_bbox=dict(type='SmoothL1Loss', beta=1.0 / 9.0, loss\_weight=1.0),

        loss\_dir=dict(

            type='CrossEntropyLoss', use\_sigmoid=False, loss\_weight=0.2)),

    # model training and testing settings

    train\_cfg=dict(

        pts=dict(

            assigner=dict(

                type='MaxIoUAssigner',

                iou\_calculator=dict(type='BboxOverlapsNearest3D'),

                pos\_iou\_thr=0.6,

                neg\_iou\_thr=0.3,

                min\_pos\_iou=0.3,

                ignore\_iof\_thr=-1),

            allowed\_border=0,

            code\_weight=[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.2, 0.2],

            pos\_weight=-1,

            debug=False)),

    test\_cfg=dict(

        pts=dict(

            use\_rotate\_nms=True,

            nms\_across\_levels=False,

            nms\_pre=1000,

            nms\_thr=0.2,

            score\_thr=0.05,

            min\_bbox\_size=0,

            max\_num=500)))

## 检测实际代码分析

Model部分实际上只是通过脚本的形式，调用了实际代码并进行了初始化，对于实际运行的model代码，需要自己写py文件，一般保存在plugin文件夹中，通过在config文件中引用，继而调用

Pointpillar相关文件为mmdection3d预实现

## Data定义

数据直接采用了nuscenes数据集，因而直接引入

## Schedule定义

## Runtime定义

# Detr3d

Detr3d是一个基于纯视觉的3D检测网络

## Model定义

## 检测实际代码分析

## Data定义

## Schedule定义

## Runtime定义

# Futr3d 激光模块

Futr3d中激光部分与pillarpoint不同，检测头是通过transformer完成的，其流程如下图所示

## Model定义

## 检测实际代码分析

## Data定义

## Schedule定义

## Runtime定义

# Futr3d 激光视觉融合

## Model定义

## 检测实际代码分析

## Data定义

## Schedule定义

## Runtime定义