

AI 工具链快速使用手册

文档版本 V3.6

发布日期 2025-08-19

前言

概述

本文档主要说明，XMTVM 软件的使用方法，以及介绍该软件使用过程中遇到的日志 及故障处理方法。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本

读者对象

本文档（本指南）主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 危险	表示如不避免则将会导致死亡或严重伤害的具有高等级风险的危

符号	说明
	害。
 警告	表示如不避免则可能导致死亡或严重伤害的具有中等级风险的危害。
 注意	表示如不避免则可能导致轻微或中度伤害的具有低等级风险的危害。
 须知	用于传递设备或环境安全警示信息。如不避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
 说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

修订日期	版本	修订说明
2025-02-19	V3.0	工具链 005/020 版本发布。
2025-03-31	V3.1	更新某些参数及算子细节，另 005 指代芯片 7205，020 指代芯片 7606。
2025-05-27	V3.2	更新 3010 版本，指代芯片 7206。
2025-06-12	V3.3	增加模型量化误差分析说明及更新某些算子的规格参数。
2025-07-09	V3.4	更改 docker 工作目录名。
2025-08-01	V3.5	3010 更换交叉编译工具包，增加 private_data 写进 xmm 的功能。
2025-08-19	V3.6	增加标准 QDQ w4 与 w8 混合量化模型支持。

目 录

前 言	i
1 概述	1
1.1 基本概念	1
1.2 XMTVM 部署结构	1
1.3 XMTVM 软件运行流程	3
2 安装	4
2.1 加载 docker 镜像及样例准备	4
2.2 样例测试	5
3 使用	6
3.1 Model Zoo Sample 流程介绍	6
3.2 各子功能介绍	6
3.2.1 配置载入: load_config()	6
3.2.2 模型载入:	7
3.2.3 模型量化和编译: build()	7
3.2.4 模型压缩导出: export()	8
3.2.5 模型推理: inference()	8
3.2.6 模型精度评估:	9
3.2.7 模型量化误差分析: quant_profile()	12
3.2.8 量化算子精度优化:	16
3.2.9 标准 QDQ 量化模型导入: load_onnx_qdq()	16
4 配置文件	18
4.1 常用配置参数说明	18
4.2 非常用配置参数说明	20

5 附录	22
5.1 onnx 框架算子支持限定	22

仅限深圳市安远威视科技有限公司使用

表格目录

表 3-1 load_config() 参数清单.....	6
表 3-2 load_onnx() 或 load_onnx_qlinear() 参数清单.....	7
表 3-3 build() 参数清单.....	7
表 3-4 export() 参数清单.....	8
表 3-5 inference() 参数清单.....	9
表 3-6 classification_evaluate() 参数清单.....	10
表 3-7 detect_evaluate() 参数清单.....	11
表 3-8 quant_profile() 参数清单.....	12
表 3-9 load_onnx_qdq() 参数清单.....	16
表 3-10 前后量化参数需保持一致的算子清单.....	17
表 4-1 常用参数清单.....	18
表 4-2 非常用参数清单.....	20

插图目录

图 1-1 用户部署架构	2
图 1-2 基本运行流程	3
图 2-1 docker 镜像测试成功界面	5
图 3-1 quant_profile 功能基本输出	14
图 3-2 advanced_profil 输出	14
图 3-3 conv_0 算子输出分布	15
图 3-4 mul_2_hardswish 算子输出分布	15
图 3-5 high_precision_layers 字段配置	16
图 3-6 QDQ Concat 算子前后量化参数保持一致示意图	17

1 概述

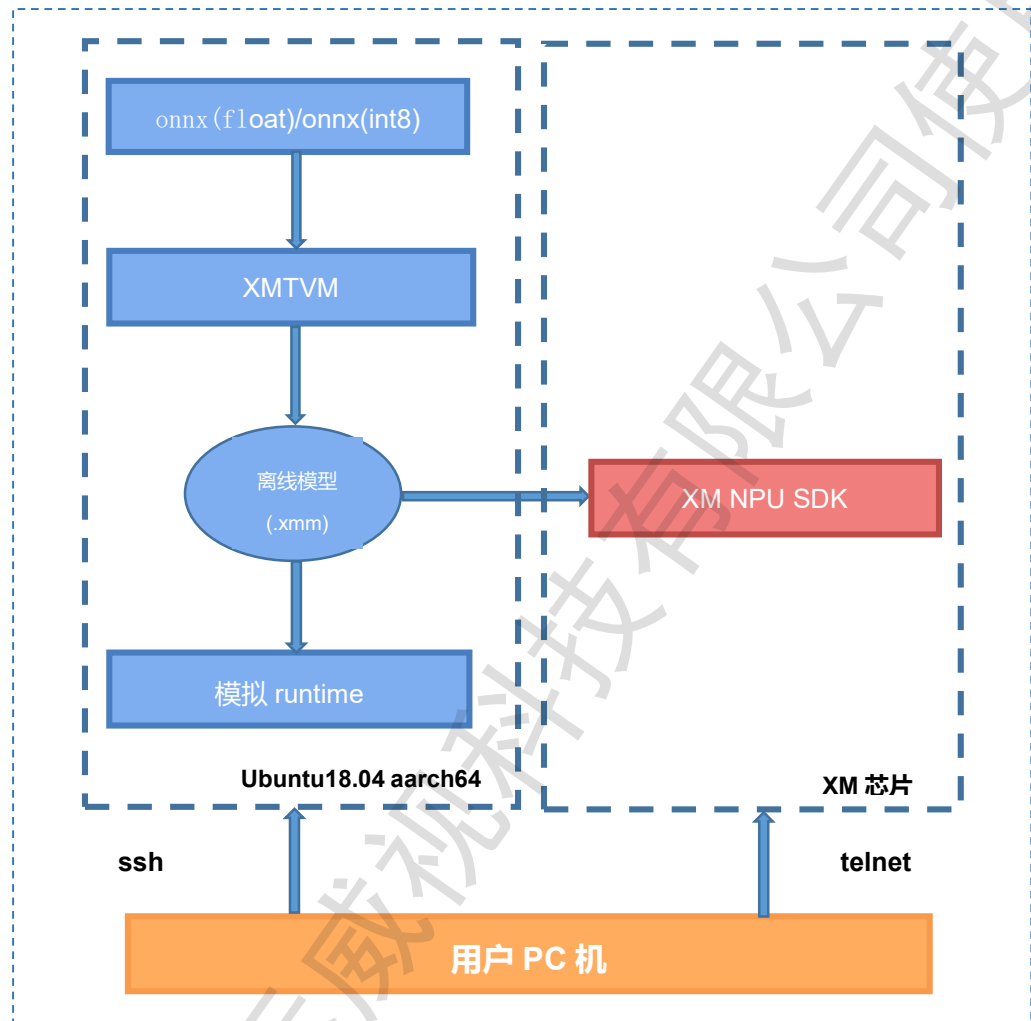
1.1 基本概念

XMTVM 工具提供了模型编译，压缩方法，生成的部署模型，可以在 NPU 处理器上进行高性能运算。该工具在编译压缩过程中，可以进行量化，算子融合，提高部署效率。

1.2 XMTVM 部署结构

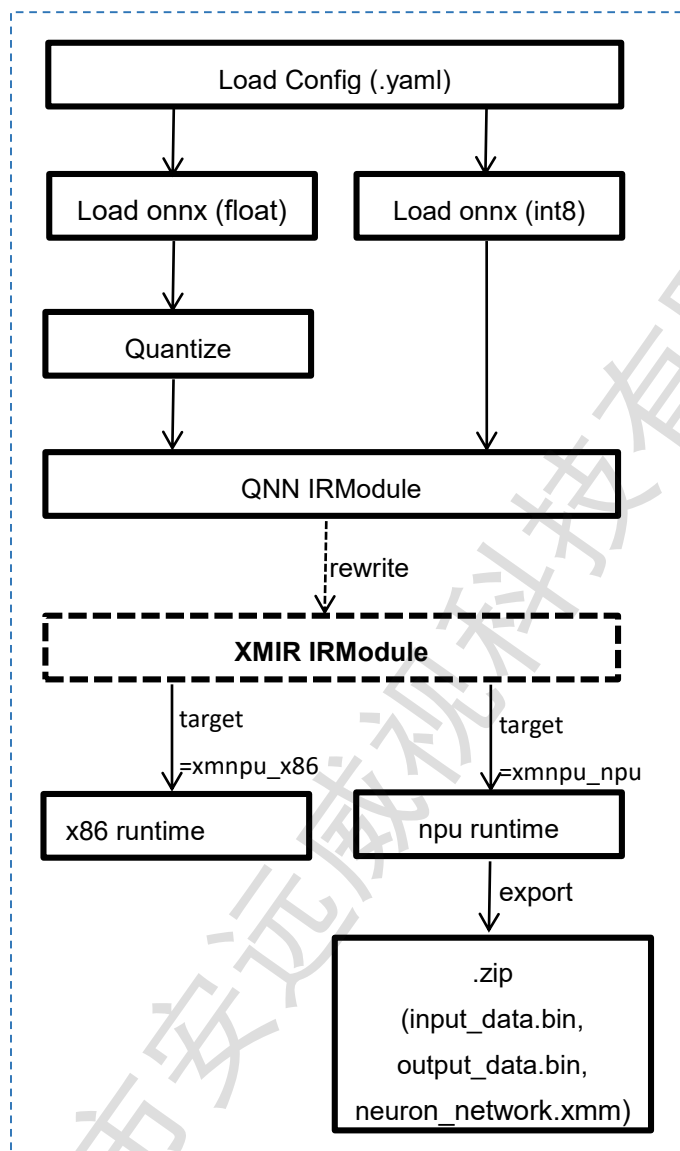
用户 XMTVM 部署是在 docker 镜像中完成模型量化，编译，压缩等工作，该方法无需用户安装 XMTVM，docker 镜像中已经搭建完成，具体操作流程参考第 2 节内容。生成离线模型后，可将.xmm 离线模型部署到 NPU 芯片上，或服务器上。

图1-1 用户部署架构



1.3 XMTVM 软件运行流程

图1-2 基本运行流程



2 安装

2.1 加载 docker 镜像及样例准备

下载工具链 docker 镜像和 opensource_model_zoo 测试样例包，将镜像和测试样例包拷贝到想要运行的服务器之后，进行下列操作：

在加载镜像之后，先确认下 docker 环境是否正常：

- `docker --version`

加载工具链镜像：

- `docker load -i ubuntu18.04_xmtvm_***.tar`

- 这里的*指代镜像包的具体版本名称

创建及启动容器：

- `docker run -dit --name #### --net=host --shm-size 32G ubuntu18.04_xmtvm:*** /bin/bash`

- 这里的 #### 指代容器名字，*** 指代 镜像版本号

将测试样例包拷贝进镜像中：

- `docker cp opensource_model_zoo.tar.gz ####:/root/xmedia/`

- 这里的 #### 指代上一步创建的容器名字

进入容器：

- `docker exec -it #### bash`

- 这里的 #### 指代前面创建的容器名字

解压缩测试样例包：

- `cd root/xmedia`
- `tar xf opensource_model_zoo.tar.gz`

2.2 样例测试

- `cd opensource_model_zoo/object_detection/yolov5`
- `python build_coco.py`

等待测试完成，如图 2-1 所示。

图2-1 docker 镜像测试成功界面

```
74      29      0.3      0.8235      0.4828      0.64      0.4608
75      32      0.3      0.5625      0.2812      0.4125      0.3212
77      35      0.3      0.6667      0.1714      0.4359      0.2814
0it [00:00, ?it/s] ./cache/xmmpu_x86_test_results not exists, a new folder will be created.
500it [07:23, 1.13it/s]
100% | 500/500 [00:06<00:00, 75.12it/s]
```

Class	Targets	conf	P	R	mAP@0.5	mAP@0.5:0.95
all	3774	0.3	0.6487	0.3261	0.5033	0.3568
0	1092	0.3	0.7796	0.4405	0.6374	0.4299
1	42	0.3	0.7778	0.1667	0.4904	0.2299
2	207	0.3	0.7015	0.2271	0.4738	0.3054
3	40	0.3	0.6	0.075	0.3405	0.2657
4	14	0.3	0.7143	0.7143	0.7503	0.4087
5	22	0.3	0.75	0.5455	0.7019	0.614
6	15	0.3	0.7333	0.7333	0.8208	0.7035
7	49	0.3	0.4138	0.2449	0.3235	0.2081
8	46	0.3	0.5714	0.1739	0.3768	0.2482
9	45	0.3	0.5455	0.1333	0.3554	0.2472
11	12	0.3	0.875	0.5833	0.7658	0.7207
13	29	0.3	0.5833	0.2414	0.4315	0.2467
14	44	0.3	0.7143	0.1136	0.4306	0.3646
15	18	0.3	0.8667	0.7222	0.8349	0.6677
16	18	0.3	0.45	0.5	0.6097	0.506
17	18	0.3	0.5185	0.7778	0.6798	0.4396
18	44	0.3	0.6667	0.3182	0.5264	0.3143
19	45	0.3	0.5455	0.1333	0.3552	0.1601
20	12	0.3	1	0.5833	0.7916	0.6621
22	23	0.3	0.7308	0.8261	0.8593	0.6592
23	20	0.3	0.85	0.85	0.8911	0.6879
24	34	0.3	1	0.02941	0.5147	0.2059
25	49	0.3	0.5556	0.3061	0.3933	0.2261
26	40	0.3	0	0	0	0
27	19	0.3	0.6667	0.3158	0.5186	0.3225
28	42	0.3	0.6667	0.04762	0.3642	0.2404
30	12	0.3	0.375	0.25	0.3235	0.2024
32	35	0.3	0.6	0.1714	0.4894	0.2552
33	49	0.3	1	0.2041	0.602	0.4105
34	28	0.3	0.75	0.2143	0.4741	0.2919
35	19	0.3	0.6667	0.3158	0.5207	0.3547
36	25	0.3	0.6667	0.48	0.5804	0.3208
37	21	0.3	0.75	0.2857	0.5405	0.2207
38	28	0.3	0.8333	0.3571	0.6246	0.3779
39	114	0.3	0.9091	0.06772	0.5021	0.4123
40	14	0.3	0.8333	0.3571	0.6246	0.4582
41	71	0.3	0.5833	0.1972	0.4036	0.2362
42	13	0.3	0.6667	0.3077	0.5076	0.357
43	24	0.3	1	0.04167	0.5208	0.4687
44	15	0.3	1	0.06667	0.5333	0.48
45	81	0.3	0.7207	0.3333	0.539	0.4401
46	57	0.3	0.4286	0.05263	0.257	0.08161
47	23	0.3	0.5556	0.8667	0.3333	0.1416

3.1 Model Zoo Sample 流程介绍

Model Zoo 中目前有包含了常见的分类和目标检测模型的 Sample Code, 其中分类模型有包含 mobilenetv2、resnet18、shufflenetv2 等, 目标检测模型有包含 yolov3、yolov4、yolov5、yolov8 等。Sample 为 python 格式, 主要由处理代码、配置文件、校准数据和模型构成。处理代码的主接口是 XMEDIA(), 主接口中又包含了配置载入、模型载入、模型量化和编译、模型压缩导出、模型推理、模型精度评估等功能的子接口。下面分别对各子接口做详细介绍:

3.2 各子功能介绍

首先得引入主接口:

```
from xmedia import XMEDIA
xmedia = XMEDIA()
```

3.2.1 配置载入: load_config()

加载配置文件

参数介绍:

表3-1 load_config() 参数清单

名字	介绍
config	配置文件的路径 (YAML 格式) 。

说明:

每个模型样例中都带有*_fast.yaml 和*_slow.yaml 两份配置文件, 其中*_fast.yaml 是快速量化的配置, 量化所花时间比较短, 缺点是精度可能不那么高, 而*_slow.yaml 是

精度较高的量化的配置，缺点就是量化所花时间会比较长，但精度会有比较大的提升，可以根据需求来选择。默认是*_fast.yaml。

完整示例：

```
xmedia.load_config(config="./yolov5n_fast.yaml") # 以yolov5n为例
```

3.2.2 模型载入：

只支持 onnx 模型载入，onnx opset version 范围为 7 到 21，超出范围不支持。

此阶段有两个接口可选，load_onnx()适用于加载浮点模型，load_onnx_qlinear()适用于加载已量化的 int8 (w8a8) 模型，为 QOperator 量化格式，为专用量化算子，如 QLinearConv、QLinearGemm、QLinearMatMul 等等，此为自研模型定制格式。

参数介绍：

表3-2 load_onnx() 或 load_onnx_qlinear() 参数清单

名字	介绍
model	浮点或已量化模型路径。

完整示例：

```
xmedia.load_onnx(model="./yolov5n.onnx") # 浮点模型载入
# or
xmedia.load_onnx_qlinear(model="./yolov5n_int8.onnx") # int8 (w8a8) 量化模型载入
```

3.2.3 模型量化和编译：build()

注意：不管是使用 load_onnx()，还是使用 load_onnx_qlinear()，build()也是流程中必不可少的。

参数介绍：

表3-3 build() 参数清单

名字	介绍
do_quantization	True or False，是否需要量化模型，只针对浮点模型，对 Int8 量化模型无效。

名字	介绍
calibrate_dataset	校准数据路径，注意只能是文件路径，这也是模型精度评估所用到的数据。
compare_similarity	True or False，模型推理结果相似度比较，会比较浮点与量化结果的相似度，以及量化结果与 xmnpu_x86 结果的相似度，相似度包含 3 种：cosine、rmse 和 mse，一般 cosine 值越大越好，而 rmse 和 mse 则是值越小越好。只针对浮点模型，对 Int8 量化模型无效。

完整示例：

```
xmedia.build(do_quantization=True, calibrate_dataset="../coco/coco_calibrate/",
compare_similarity=True) # 针对浮点模型
# or
xmedia.build() # 针对Qlinear模型
```

3.2.4 模型压缩导出：export()

将编译好的模型压缩导出，模型的输入数据和输出结果也会一并导出。

参数介绍：

表3-4 export() 参数清单

名字	介绍
path	导出模型的命名，zip 格式。

完整示例：

```
xmedia.export(path="../yolov5n.zip")
```

3.2.5 模型推理：inference()

inference()是公有的模型推理接口，此接口返回的仅仅是模型输出的 feature，并非最终的结果，要针对 feature 做相应的后处理才能得到最终的结果。

当前有针对常见的一些模型添加了后处理接口，比如分类模型的 SoftmaxPostProcess 接口，目标检测的 YOLOV3、YOLOV4、YOLOV5、YOLOV8、YOLOV8_POSE 等都有相应的后处理接口，详情可参见 Sample 代码。

参数介绍：

表3-5 inference() 参数清单

名字	介绍
input_data	输入的数据文件，支持 jpg、jpeg、png、bin 等数据格式。
backend	可供选择：onnxruntime、mqbench 和 xmnpu_x86，指代 target，分别对应浮点模型推理、量化模型推理、XMIR 模型推理。
preprocess	对输入数据做预处理的方式，默认 resize_pad_0，一般此参数可不设置，保持默认即可。

完整示例：

```
features = xmedia.inference("../imagenet_calibrate/images/ILSVRC2012_val_00000388.JPEG",
backend="onnxruntime") # 浮点模型推理

# or
features = xmedia.inference("../imagenet_calibrate/images/ILSVRC2012_val_00000388.JPEG",
backend="mqbench") # 量化模型推理，此点只针对是由浮点做过量化得来量化模型

# or
features = xmedia.inference("../imagenet_calibrate/images/ILSVRC2012_val_00000388.JPEG",
backend="xmnpu_x86") # XMIR模型推理
```

3.2.6 模型精度评估：

没有针对所有模型的通用的接口，分类模型可以使用 classification_evaluate()接口，目标检测模型可使用 detect_evaluate()接口，详情可参见 Sample 代码。

参数介绍：

表3-6 classification_evaluate() 参数清单

名字	介绍
input_data	输入的数据路径，支持 jpg、jpeg、png、bin 等数据格式。
label_path	输入数据文件所对应的类别 ID 文件，详情可参见 Sample。
postprocess	后处理接口，对应 SoftmaxPostProcess()接口，详情可参见 Sample。
backend	可供选择：onnxruntime、mqbench 和 xmnpu_x86，指代 target，分别对应浮点模型推理、量化模型推理、XMIR 模型推理。
preprocess	对输入数据做预处理的方式，默认 resize_pad_0，一般此参数可不设置，保持默认即可。

完整示例：

```
from xmedia import SoftmaxPostProcess

class_index = "../imagenet_calibrate/imagenet_class_index.json" ## 可以是直接dict类型 或 dict类型的json文件
class_num = 1000
postprocess = SoftmaxPostProcess(class_index, class_num)

img_path = "../imagenet_calibrate/images"
lables_path = "../imagenet_calibrate/lables.txt"
accuracy = xmedia.classification_evaluate(img_path, lables_path, postprocess,
backend="onnxruntime")
# or
accuracy = xmedia.classification_evaluate(img_path, lables_path, postprocess,
backend="mqbench")
# or
accuracy = xmedia.classification_evaluate(img_path, lables_path, postprocess,
backend="xmnpu_x86")
```

表3-7 detect_evaluate() 参数清单

名字	介绍
input_data	输入的数据路径，支持 jpg、jpeg、png、bin 等数据格式。
evaluator	Evaluator 接口，有 COCOEvaluator、VOCEvaluator 和 YOLOEvaluator 可供选择，分别对应不同的数据格式。
backend	可供选择：onnxruntime、mqbench 和 xmnpu_x86，指代 target，分别对应浮点模型推理、量化模型推理、XMIR 模型推理。
preprocess	对输入数据做预处理的方式，默认 resize_pad_0，一般此参数可不设置，保持默认即可。
draw	True or False，是否绘制 bbox 并保存，默认为 True。
draw_id	list，绘制指定 ID 对应的 bbox，默认绘制所有 bbox。
print_limit	int，检测目标打印阈值，检测到 target > print_limit 才打印检测结果。

完整示例：(以 COCOEvaluator 接口为例，其余接口使用方式类似，可参见 Sample 代码)

```
from xmedia import YOLOV5, COCOEvaluator

def yolov5():
    """yolov5后处理部分，取模型最后一层conv输出作为模型输出"""
    class_num = 80 # 输出类别数量
    layer_num = 3 # 输出conv分支数量
    strides = [8, 16, 32]
    anchors = [[10, 13, 16, 30, 33, 23],
               [30, 61, 62, 45, 59, 119],
               [116, 90, 156, 198, 373, 326]]
    conf_thres = 0.3 # confidence 阈值
    iou_thres = 0.65 # nms iou 阈值
    max_det = 1000 # 最大检测数量

    yolov5 = YOLOV5(
```

```

        class_num=class_num,
        layer_num=layer_num,
        strides=strides,
        anchors=anchors,
        conf_thres=conf_thres,
        iou_thres=iou_thres,
        max_det=max_det
    )
    return yolov5

yolov5 = yolov5()
print(yolov5)

img_path = "../coco/coco_calibrate"
target_path = "../coco/instances_val2017.json"
evaluator = COCOEvaluator(target_path, yolov5, conf=0.3)
mAP = xmedia.detect_evaluate(img_path, evaluator, backend="onnxruntime", draw=True)
# or
mAP = xmedia.detect_evaluate(img_path, evaluator, backend="mqbench", draw=True)
# or
mAP = xmedia.detect_evaluate(img_path, evaluator, backend="xmnpu_x86", draw=True)

```

3.2.7 模型量化误差分析：quant_profile()

使用 quant_profile 接口可以分析模型的量化误差。可以获得网络逐层量化误差及中间层数据分布。

参数介绍：

表3-8 quant_profile() 参数清单

名字	介绍
profile_dataset	str, profile 数据路径。推荐与 build 函数 calibrate_dataset 参数一致。
advanced_profile	str, 是否开启 advanced_profile。默认不启用，推荐设置为 standalone 以分析单层量化对最终输出影响。 注意：standalone 会增加 profile 耗时。
sample_count	int, 设置 profile 使用的数据数量。默认为 0 表示使用路径下所有数据。

名字	介绍
plot_channel	bool, 是否输出网络中间层数据分布。默认为 False, 。
profile_all	bool, 是否只 profile 可优化的 layer。默认为 False, 表示只输出可通过 high_precision_layers 字段调优的算子。为 True 则输出模型中所有算子。

完整示例:

以 mobilenetv3_small 为例, quant_profile 函数需要在 build 函数之后调用。

```
from xmedia import XMEDIA, SoftmaxPostProcess

if __name__ == "__main__":
    xmedia = XMEDIA()

    print("--> Loading configuration <--")
    xmedia.load_config(config="mobilenetv3-small.yaml")

    print("--> Loading ONNX model <--")
    xmedia.load_onnx(model="mobilenetv3-small.onnx")

    print("--> Building ONNX model <--")
    xmedia.build(do_quantization=True, calibrate_dataset="../imagenet/ilsvrc2012_calibrate",
compare_similarity=True)

    print('--> Export model binaries <--')
    xmedia.export(path="mobilenetv3-small.zip")

    xmedia.quant_profile("../imagenet/ilsvrc2012_calibrate", advanced_profile="standalone",
plot_channels=True)
```

图3-1 quant_profile 功能基本输出

1	Basic Quant Layer Profile						
2	+-----+-----+-----+-----+-----+-----+-----+						
3		name		shape		cosine	cosine_rank mean_rmse mean_rmse_rank
4	+-----+-----+-----+-----+-----+-----+-----+						
5		conv_normalize_0		[1, 3, 224, 224]		0.999980	31 0.004878 46
6		conv_0		[1, 16, 112, 112]		0.994880	1 0.151519 1
7		mul_2_hardswish		[1, 16, 112, 112]		0.997632	2 0.107809 2
8		conv_6		[1, 8, 1, 1]		nan	32 0.000000 50
9		conv_8		[1, 16, 1, 1]		0.999992	42 0.004965 45
10		conv_11		[1, 16, 56, 56]		0.999141	3 0.063504 3
11		conv_12		[1, 72, 56, 56]		0.999580	14 0.044597 8
12		conv_14		[1, 72, 28, 28]		0.999905	33 0.026433 27
13		conv_17		[1, 88, 28, 28]		0.999929	30 0.019184 31
14		conv_19		[1, 88, 28, 28]		0.999965	34 0.015208 32
15		conv_23		[1, 96, 28, 28]		0.999622	16 0.036858 17

以 mobilenetv3_small 为例，图 3-1 为模型量化各个算子量化对 featuremap 的影响。

设置 advanced_profile 为 standalone，会额外输出各个算子单独量化对最终结果的影响。

图3-2 advanced_profil 输出

1	+-----+-----+-----+-----+-----+-----+-----+									
2		quantizer		tensor range		shape		difference		rank
3	+-----+-----+-----+-----+-----+-----+-----+									
4		conv_normalize_0		[-2.117904-2.640000]		[1, 3, 224, 224]		out[0] cosine 0.999968 rmse 0.037861 cosine	30 rmse	31
5										
6		conv_0		[-145.122787-148.067429]		[1, 16, 112, 112]		out[0] cosine 0.300010 rmse 5.277766 cosine	1 rmse	1
7										
8		mul_2_hardswish		[-0.375000-148.067429]		[1, 16, 112, 112]		out[0] cosine 0.462884 rmse 4.855586 cosine	2 rmse	2
9										
10		conv_6		[0.000000-0.000000]		[1, 8, 1, 1]		out[0] cosine 1.000000 rmse 0.000000 cosine	50 rmse	50
11										
12		conv_8		[-1.137220-1.355582]		[1, 16, 1, 1]		out[0] cosine 0.999985 rmse 0.027339 cosine	35 rmse	36
13										
14		conv_11		[-20.959900-17.788248]		[1, 16, 56, 56]		out[0] cosine 0.996474 rmse 0.448663 cosine	4 rmse	4
15										
16		conv_12		[0.000000-20.520910]		[1, 72, 56, 56]		out[0] cosine 0.994674 rmse 0.524146 cosine	3 rmse	3
17										
18										

以 mobilenetv3_small 模型为例，可以定位到 conv_0, mul_2_hardswish 两个算子对输出影响最大。

设置 `plot_channel` 为 `True`，会在 `cache/channel_plot` 路径下绘制各个算子的输出各 `channel` 的统计分布。

图3-3 conv_0 算子输出分布

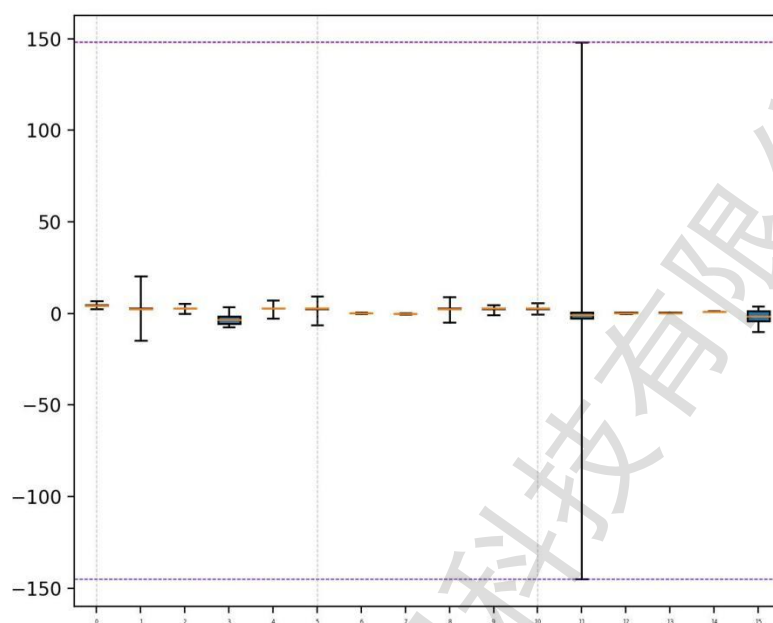
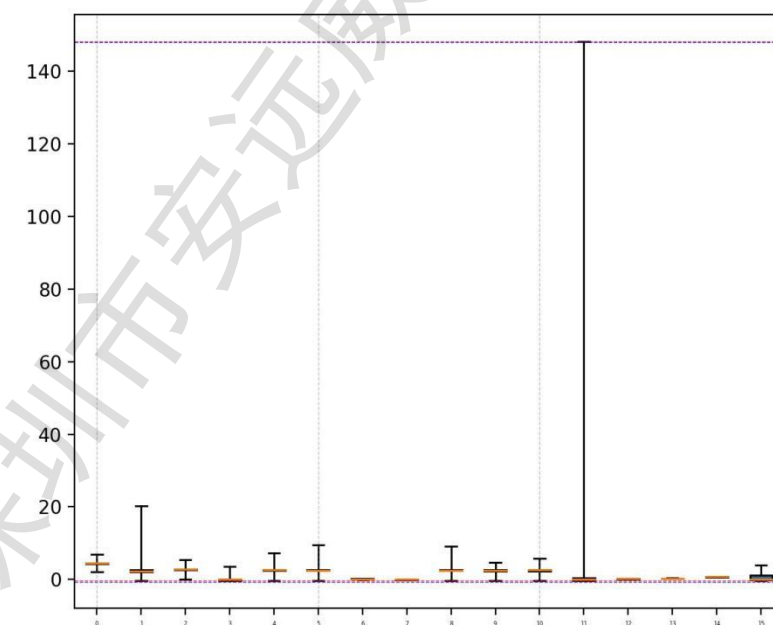


图3-4 mul_2_hardswish 算子输出分布



可以看到，两个算子量化误差较大，可以通过配置 `high_precision_layers` 字段进行优化。其中横线表示量化截断范围。

3.2.8 量化算子精度优化：

通过 quant_profile 功能定位到对模型精度影响较大的算子。之后可以通过配置 yaml 文件 high_precision_layers 字段对需要优化的算子进行调优。

以 mobilenetv3_small 为例，可以在 high_precision_layers 字段配置 conv_0, mul_2_hardswish 以提升量化精度。

图3-5 high_precision_layers 字段配置

```
1 quantize:
2   quantize_type: naive_ptq
3   extra_qconfig_dict:
4     w_observer: MinMaxObserver
5     w_fakequantize: AdaRoundFakeQuantize
6     a_observer: MinMaxObserver
7     a_fakequantize: QDropFakeQuantize
8   high_precision_layers:
9     - conv_0
10    - mul_2_hardswish
```

3.2.9 标准 QDQ 量化模型导入：load_onnx_qdq()

只支持标准 QDQ 格式 w4a8、w8a8 或两者混合的已量化 onnx 模型载入，onnx opset version 范围为 7 到 21，超出范围不支持。

参数介绍：

表3-9 load_onnx_qdq() 参数清单

名字	介绍
model	QDQ 已量化模型路径。

完整示例：

```
xmedia.load_config(config="config.yaml") # 加载配置
xmedia.load_onnx_qdq(model="./yolov5n_w4a8.onnx") # QDQ w4a8、w8a8量化模型载入
xmedia.build() # QDQ模型编译
xmedia.export(path="./yolov5n_w4a8.zip") # xmm模型导出
```

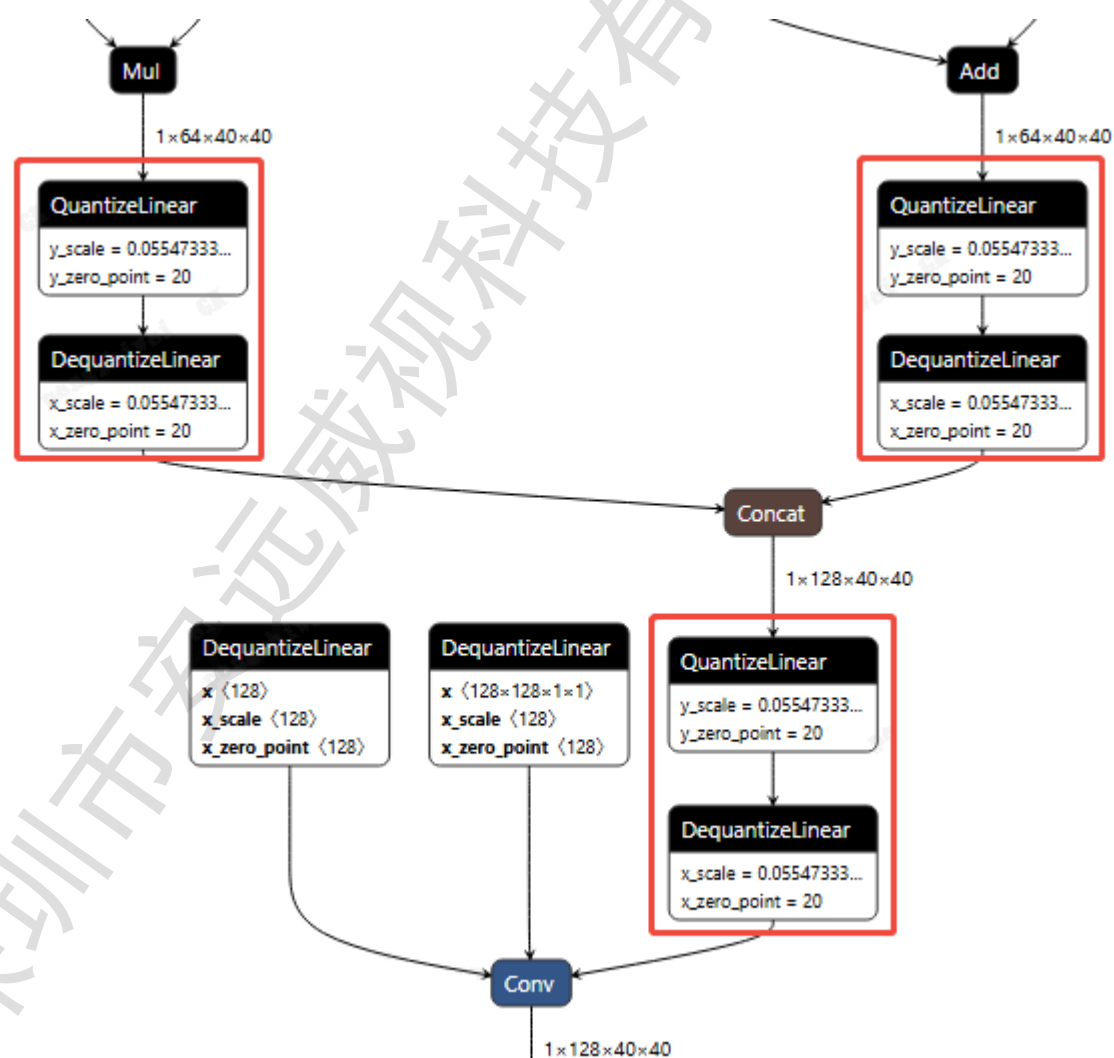
特别说明：

因 NPU 特性，为保证量化精度与 NPU 精度对齐，如下算子必须保证前后量化参数 (scale 与 zero_point) 保持一致：

表3-10 前后量化参数需保持一致的算子清单

AveragePool、Clip、Concat、Expand、Flatten、GlobalAveragePool、LeakyRelu、MaxPool、Mean、Pad、ReduceMean、Relu、Reshape、Resize、Slice、Split、Squeeze、Tile、Transpose、Unsqueeze、Upsample

图3-6 QDQ Concat 算子前后量化参数保持一致示意图



4 配置文件

4.1 常用配置参数说明

表4-1 常用参数清单

名字	介绍
version: npu:	对应 NPU_VERSION, 有 005、020、3010 可选, 跟开发板对应。005 对应芯片 7205, 020 对应芯片 7606, 3010 对应芯片 7206。
input_shape:	输入数据的维度。 020 / 3010 支持[batch,channel,height,width]、[batch,channel,height]、[batch,channel], batch 需为 1, 目前 npu 只支持 batch 为 1, 即可以支持 2 维到 4 维的数据输入; 005 只支持[batch,channel,height,width]的 4 维数据输入。 说明: layout 格式只支持 NCHW。
input_normalize:	输入的均值和方差。
input_format:	输入数据格式。 020 / 3010 支持 RGB、BGR、GRAY, DEFAULT 等, 默认 RGB, 其中 DEFAULT 为*.npy 数据的格式, 可以是 2 维到 4 维的 uint8 或 float32 数据。 005 支持 RGB、BGR、GRAY、NV21_BT601_FULL、NV21_BT709_FULL、NV21_BT601_TV、NV21_BT709_TV、YUV420SP_BT601_FULL、YUV420SP_BT709_FULL、YUV420SP_BT601_TV、YUV420SP_BT709_TV 等, 默认 RGB。

名字	介绍
model_format:	模型的格式。 020 支持 RGB、BGR、GRAY, DEFAULT 等, 默认 RGB, 其中 DEFAULT 是与 input_format 匹配的。 005 支持 RGB、BGR、GRAY。
output:	指定模型的输出节点(会删除指定节点后续的算子)。此参数针对需要裁剪的模型做设置, 即带有后处理的模型, 或者模型后段存在芯片不支持的算子的模型, 则可设置此参数做裁剪, 如模型不需要裁剪的话, 此参数不用设置。
quantize: quantize_type:	量化参数, 选定量化类型, 可选 advanced_ptq 和 naive_ptq, 一般默认为 naive_ptq, 不过 advanced_ptq 量化出来的精度会更好些, 缺点是花费时间。
quantize: extra_qconfig_dict: w_observer:	量化参数, weight 的 observer 选定, 可选 MinMaxObserver、MSEObserver、LSQObserver 和 AdaSearchObserver, 一般默认为 MinMaxObserver, 如果对精度要求比较高的话, 可以选 AdaSearchObserver, 缺点同样是比较花时间。
quantize: extra_qconfig_dict: w_fakequantize:	量化参数, weight 的 fakequantize 选定, 一般固定为 AdaRoundFakeQuantize。
quantize: extra_qconfig_dict: a_observer:	量化参数, activation 的 observer 选定, 可选 MinMaxObserver、EMAMSEObserver、EMAQuantileObserver 和 AdaSearchObserver, 一般默认为 MinMaxObserver, 如果对精度要求比较高的话, 可以选 AdaSearchObserver, 缺点同样是比较花时间。
quantize: extra_qconfig_dict: a_fakequantize:	量化参数, activation 的 fakequantize 选定, 一般固定为 QDropFakeQuantize。
quantize: data: preprocess:	量化参数, 校准数据的预处理, 可选有 resize, resize_{w}_{h}, resize_pad_{value}, resize_{w}_{h}_pad_{value}等, 一般默认为 resize_pad_0。比如: 图像尺寸大小为 640x360, 而模型输入尺寸为 224x224, 使用 resize_pad_0 则会先将图

名字	介绍
	像 resize 到 224x112, 然后再将 h 上没到 224 的其余位置补 0, 这样图像不会被拉伸, 不会变形; 而如果设置为 resize, 则是直接将图像拉伸到 224x224, 图像会变形, 其他设置依次类推。
compile: model_mode:	编译参数, 模型在开发板上运行的模式, 可选 high、normal 和 low, 一般默认为 high。
compile: runtime_input_layout:	编译参数, 此参数跟 RGB 数据的存储格式相关, 可选不设置, NCHW 和 NHWC。不设置以及 NCHW, 则都是默认的 NCHW 格式, 对应 RGB 的 Planar 存储格式; NHWC 则对应 RGB 的 Packed 存储格式。3010 必须设置为 NHWC, 即其只支持 RGB 的 Packed 存储格式。
compile: libc:	编译参数, 此参数跟交叉编译工具包相关, 默认是不配置, 是对应 020 的交叉编译工具, 3010 则必须配置为 uclibceabi, 另外 005 版本也不需要配置。

4.2 非常用配置参数说明

表4-2 非常用参数清单

名字	介绍
quantize: calibrate_sample:	使用校准数据数量, 默认设置 500。若设置为 0, 则使用校准数据目录下所有数据。
quantize: reconstruction: warm_up: weight: max_count: b_range:	量化参数, 一般 warm_up、weight、max_count 和 b_range 是一起调整以达到改善量化精度的作用, 默认 warm_up: 0.2, weight: 0.01, max_count: 1000, b_range: [20, 2], 可供参考调整的一组参数为: warm_up: 0.02, weight: 0.001, max_count: 2000, b_range: [40, 2]。
quantize: adasearch: sample_count:	使用 AdaSearchObserver 时生效。配置 AdaSearchObserver 使用的校准数据数量, 默认为 10。设置值越大越好, 但是同时耗时也会显著增加。

名字	介绍
quantize: smoothquant: enable: True config: alpha: 0.15	量化方法，如对模型量化精度有更高要求时，可选择开启 smoothquant 来量化，参数设置如左边所示，相应的量化时间会有增加。
runner: runtime_private_data:	runtime 参数，此参数是针对需要传递某些信息到编译后的 xmm 模型中的配置，比如 input_format、output name 与 tensor_id 对应的映射等，可以添加自己想添加的信息，但添加的数据必须是 JSON 格式，写到 private_data.bin 文件中去，此参数默认是不配置的，如果需要配置的话，则是需要配置一个路径，比如 ./private_data.bin，这个文件中可以写 JSON 格式的数据，也可以是个空文件，或者就只填一个路径，没有实际文件也可以。

5 附录

5.1 onnx 框架算子支持限定

onnx opset version 范围为 7 到 21，超出范围不支持。

大部分算子只支持 batch 为 1，少部分例外的会特别标注，表格中 B 代表 batch，C 代表 channel，H 代表 height，W 代表 width。

模型输入的 H 与 W 最大支持：005 支持到 1024，020 和 3010 支持到 2048。

NPU 版本中的 005 指代芯片 7205，020 指代芯片 7606，3010 指代芯片 7206。

算子支持	NPU 版本	规格限定
Add	005 / 020 / 3010	支持变量+变量，变量+常量，不支持常量+变量 005：只支持[B,C,H,W] 4 维输入，且 B 只能为 1，不支持广播 020 / 3010：支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入，且 B 只能为 1，其中 C,H,W 支持双向广播，但不支持不同维度广播
AveragePool	005 / 020 / 3010	只支持[B,C,H,W] 4 维输入，且 B 只能为 1 005：kernel size 只支持(2、3)，stride 支持(1、2、3)，padding 支持(0、1) 020 / 3010：stride 支持(1 -- 10)，padding 支持(0 -- 10)，kernel size 支持(1 -- 10)
BatchNormalization	005 / 020 / 3010	005 只支持[B,C,H,W] 4 维输入，且 B 只能为 1，会转换为 Conv 020 / 3010 支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入，且 B 只能为 1
Celu	020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入，且 B 只能为 1，

		会转换为 LUT
Clip	005 / 020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1
Concat	005 / 020 / 3010	005: 只支持[B,C,H,W] 4 维输入, 且 B 只能为 1, axis 只能为 1, 且输入数目最大到 12 020 / 3010: 支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, axis 支持(1、2、3), C、H、W 每个维度合并后的值最大到 4096, axis = 1 时, 输入数目最大到 49, axis != 1 时, 输入数目只能为 2
Conv	005 / 020 / 3010	只支持[B,C,H,W] 4 维输入, 且 B 只能为 1, stride 支持(1 -- 10), padding 支持(0 -- 10), kerne size 支持(1 -- 10)
ConvTranspose	005 / 020 / 3010	只支持[B,C,H,W] 4 维输入, 且 B 只能为 1 005: kernel size 支持(1、3、5、7), stride 只支持为 2, dilation 和 group 只支持为 1, padding 支持(0 -- 3) 020 / 3010: stride 支持(1 -- 10), padding 支持(0 -- 10), kerne size 支持(1 -- 10)
Div	020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, 其中 C,H,W 支持双向广播, 但不支持不同维度广播
Elu	005 / 020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, 会转换为 LUT
Exp	005 / 020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, 会转换为 LUT
Expand	005 / 020 / 3010	只支持[B,C,H,W] 4 维输入, 且 B 只能为 1, 会转换为 Tile
Flatten	005 / 020 / 3010	支持 2 到 5 维输入, Reshape 实现, 且 B 维度不支持操作
Gather	020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, axis 支持(1、2、3), 会转换为 Reshape+Slice

Gemm	005 / 020 / 3010	支持[B,C] 2 维输入, 且 B 只能为 1
GlobalAveragePool	005 / 020 / 3010	只支持[B,C,H,W] 4 维输入, 且 B 只能为 1
Gelu	020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, 会转换为 LUT
HardSigmoid	005 / 020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, 会转换为 LUT
HardSwish	005 / 020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, 会转换为 LUT
InstanceNorm	020 / 3010	支持 [B,C,H], [B,C,H,W]等 3 到 4 维输入, 且 B 只能为 1
LeakyRelu	005 / 020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1
Log	020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, 会转换为 LUT
LogSigmoid	020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, 会转换为 LUT
LSTM	020 / 3010	支持[B,C,H] 3 维输入, 且 B 只能为 1
LayerNorm	020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, axis 支持(1、2、3)
MatMul	005 / 020 / 3010	支持[B,C,H] 3 维输入, B 支持(1 -- 16)
Max	020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, ReduceMax 实现, axis 支持(1、2、3)
MaxPool	005 / 020 / 3010	只支持[B,C,H,W] 4 维输入, 且 B 只能为 1 005: kernel size 只支持(2、3), stride 支持(1、2、3), padding

		支持(0、1) 020 / 3010: stride 支持(1 -- 10), padding 支持(0 -- 10), kernel size 支持(1 -- 10)
Mean	020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, ReduceMean 实现, axis 支持(1、2、3)
Min	020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, ReduceMin 实现, axis 支持(1、2、3)
Mish	005 / 020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, 会转换为 LUT
Mul	005 / 020 / 3010	支持变量*变量, 变量*常量, 不支持常量*变量 005: 只支持[B,C,H,W] 4 维输入, 且 B 只能为 1, 不支持广播 020 / 3010: 支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, 其中 C,H,W 支持双向广播, 但不支持不同维度广播
PReLU	005 / 020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, 会转换为 LUT
Pad	005 / 020 / 3010	只支持[B,C,H,W] 4 维输入, 且 B 只能为 1 005: pad_mode = constant 020 / 3010: pad_mode = symmetric
Pow	020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, 会转换为 LUT
ReduceMax	020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, axis 支持(1、2、3)
ReduceMean	020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, axis 支持(1、2、3)
ReduceMin	020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1,

		axis 支持(1、2、3)
Relu	005 / 020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1
Relu6	005 / 020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, 会转换为 LUT
Reshape	005 / 020 / 3010	支持 2 到 5 维输入, 且 B 维度不支持操作
Resize	020 / 3010	支持[B,C,H,W] 4 维输入, 且 B 只能为 1, mode 支持 bilinear 和 nearest
Selu	020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, 会转换为 LUT
Silu	005 / 020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, 会转换为 LUT
Sigmoid	005 / 020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, 会转换为 LUT
Slice	005 / 020 / 3010	005 只支持[B,C,H,W] 4 维输入, 且 B 只能为 1, axes = 1 020 / 3010 支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, axes 支持(1、2、3)
Split	005 / 020 / 3010	只支持[B,C,H,W] 4 维输入, 且 B 只能为 1, axis = 1
Sqrt	020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, 会转换为 LUT
Squeeze	005 / 020 / 3010	支持 2 到 5 维输入, Reshape 实现, 且 B 维度不支持操作
Sub	020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, 其中 C,H,W 支持双向广播, 但不支持不同维度广播
Sum	020 / 3010	只支持[B,C,H,W] 4 维输入, 且 B 只能为 1, ReduceSum 实现,

		axis 支持(1、2、3)
Softmax	020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, axis 支持(1、2、3)
Softplus	020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, 会转换为 LUT
Softsign	020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, 会转换为 LUT
Swish	020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, 会转换为 LUT
Tanh	005 / 020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, 会转换为 LUT
Tanhshrink	020 / 3010	支持[B,C], [B,C,H], [B,C,H,W]等 2 到 4 维输入, 且 B 只能为 1, 会转换为 LUT
Tile	005 / 020 / 3010	只支持[B,C,H,W] 4 维输入, 且 B 只能为 1
Transpose	005 / 020 / 3010	支持 2 到 5 维输入, 且 B 维度不支持操作
Unsqueeze	005 / 020 / 3010	支持 2 到 5 维输入, Reshape 实现, 且 B 维度不支持操作
Upsample	005 / 020 / 3010	只支持[B,C,H,W] 4 维输入, 且 B 只能为 1, factor = 2