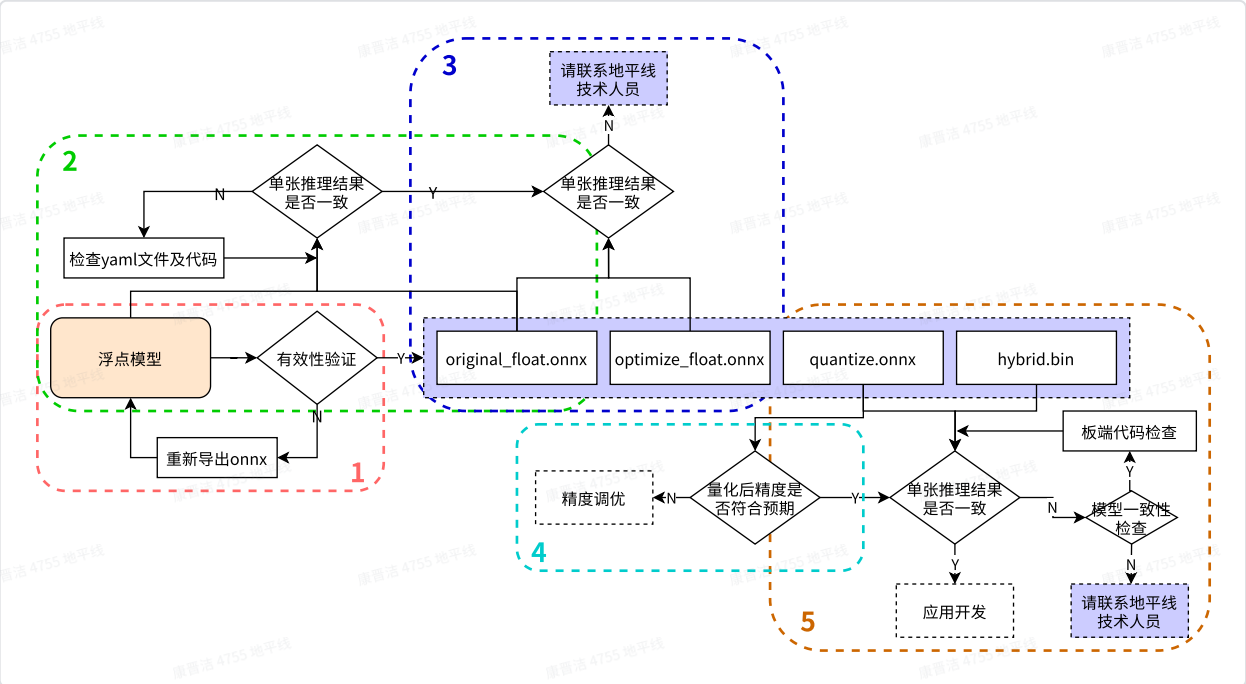
模型精度验证及调优

0 精度验证推荐流程



流程	验证目的	验证途径	参考代码/其他说明
1	确保导出的浮点onnx的有效性	测试浮点onnx模型的单张结果，应与训练后的模型推理结果完全一致	请参考后文：**1.1 验证onnx模型推理结果正确性**
2	确保yaml配置文件以及前后处理代码的正确性	测试original_float.onnx模型的单张结果，应与浮点onnx推理结果完全一致（nv12格式除外，由于nv12数据本身有损，可能会引入少许差异）	请参考后文：**1.2 yaml配置文件与前处理代码详解**
3	确保图优化阶段未引入精度误差	测试optimize_float.onnx模型的单张结果，应与original_float.onnx推理结果完全一致	推理代码同上（若结果不一致，请先尝试更新OE开发环境至最新版本，若仍然有问题，请至地平线开发者社区工具链板块提问，并提供原始浮点onnx、配置文件、测试数据）
4	验证量化精度是否满足预期	测试quantized.onnx的精度指标	建议可直接复用浮点模型评测代码： 1. 将模型加载/推理部分的代码改为加载/推理onnx模型

			2. dataloader的batchsize改为与onnx shape相对应 3. 预处理代码做对应修改（参照前文建议） 若精度不满足预期，则可直接进入精度调优环节（可参考后文**1.4 精度调优建议**）
5	确保模型编译过程无误且板端推理代码正确	使用hb_model_verifier工具验证quantized.onnx和.bin的一致性，模型输出应至少满足小数点后2-3位对齐	hb_model_verifier工具（详细介绍可参考）的使用方式： `hb_model_verifier -q quantized.onnx -b xxx.bin -a board_ip` > 目前该工具暂只支持比对单输入模型 > J5从OE1.1.40以及XJ3从OE2.5.2开始提供了hb_verifier工具，可支持多输入模型一致性校验，可参考后文1.3节相关介绍 此外，可在板端使用hrt_model_exec infer工具推理得到.bin模型原始输出与quantized.onnx做对比，同样可排除工程代码有误带来的影响。 （若quantized.onnx与.bin模型一致性校验失败，请请先尝试更新OE开发环境至最新版本，若仍然有问题，请至地平线开发者社区工具链板块提问，并提供原始浮点onnx、配置文件、测试数据）

1 参考代码及注意事项详解

1.1 验证onnx模型推理结果正确性

我们更推荐使用HB_ONNXRuntime而不是公版onnxruntime的原因在于公版onnxruntime对于部分算子的实现与原始训练框架有差异，进而有可能会导致模型推理结果不一致。

1. **验证原始浮点onnx模型的正确性**

> 特指从DL框架导出的onnx模型

Python

```python
from horizon_tc_ui import HB_ONNXRuntime
import numpy as np
import cv2

def preprocess(input_name):
    # BGR->RGB、Resize、CenterCrop···
    # HWC->CHW
    # normalization
    return data

def main():
    # 加载模型文件
    sess = HB_ONNXRuntime(model_file=MODEL_PATH)
    # 获取输入&输出节点名称
    input_names = [input.name for input in sess.get_inputs()]
    output_names = [output.name for output in sess.get_outputs()]
    # 准备模型输入数据
    feed_dict = dict()
    for input_name in input_names:
        feed_dict[input_name] = preprocess(input_name)

    # 原始浮点onnx，数据dtype=float32
    outputs = sess.run_feature(output_names, feed_dict, input_offset=0)

    # 后处理
    postprocess(outputs)

if __name__ == '__main__':
    main()
```

2. **验证转换工具产出物的正确性**

> 特指original_float.onnx、optimize_float.onnx、quantized.onnx

Python

```python
from horizon_tc_ui import HB_ONNXRuntime
import numpy as np
import cv2

def preprocess(input_name):
    # BGR->RGB、Resize、CenterCrop···
    # HWC->CHW（通过onnx模型输入节点的具体shape来判断是否需要做layout转换）
    # input_type_train->input_type_rt*（主要是nv12模型，需要将数据处理至yuv444这个中间类型，关于中间类型的解读请参考后文1.2.1节）
    # ~~normalization~~（若已通过yaml文件将norm操作放入了模型中，则不要在预处理中做重复操作）
    #-128（图像输入模型，仅在使用hb_session.run接口时需要自行在预处理完成-128，其他接口通过input_offset控制即可）
    return data

def main():
    # 加载模型文件
    sess = HB_ONNXRuntime(model_file=MODEL_PATH)
    # 获取输入&输出节点名称
    input_names = [input.name for input in sess.get_inputs()]
    output_names = [output.name for output in sess.get_outputs()]
    # 准备模型输入数据
    feed_dict = dict()
    for input_name in input_names:
        feed_dict[input_name] = preprocess(input_name)
    #图像输入的模型（RGB/BGR/NV12/YUV444/GRAY），数据dtype= uint8
    outputs = sess.run(output_names, feed_dict, input_offset=128)
    # featuremap模型，数据dtype=float32
    outputs = sess.run_feature(output_names, feed_dict, input_offset=0)
    # 混合多输入（即同时包含featuremap和图像输入）模型
    outputs = sess.hb_session.run(output_names, feed_dict)  #-128的操作需要在预处理时完成
    # 后处理
    postprocess(outputs)

if __name__ == '__main__':
    main()
```

1.2 yaml配置文件与前处理代码详解

模型转换完成浮点模型到地平线混合异构模型的转换。 为了使得这个异构模型能快速高效地在嵌入式端运行， 模型转换重点在解决 **输入数据处理** 和 **模型优化编译** 两个问题。本节重点解析在**输入数**

据处理方面的内部逻辑，便于大家理解预处理节点与模型前处理的配合关系。

1.2.1 预处理节点解析

因为地平线的边缘AI计算平台会为某些特定类型的输入通路提供硬件级的支撑方案， 但是这些方案的输出不一定符合模型输入的要求。 例如视频通路方面就有视频处理子系统，为采集提供图像裁剪、缩放和其他图像质量优化功能，这些子系统的输出往往是yuv420格式图像， 而我们的算法模型通常是基于bgr/rgb等一般常用图像格式训练得到的。为减少用户板端部署时的工作量，我们将几种常见的图像格式转换以及常用的图像标准化操作固化进了模型当中，其表现为模型input节点之后插入了预处理节点HzPreprocess（您可以使用开源工具 Netron 观察转换过程中的中间产物）。

由于HzPreprocess的存在，会使得转换后的模型其预处理操作可能会和原始模型有所不同，因此我们先来详细了解一下**预处理节点**的插入逻辑。

在mapper工具完成对caffe/onnx模型的转换时，首先会将caffe模型解析为onnx格式，并根据yaml配置文件中的配置参数（input_type_rt、input_type_train以及norm_type）决定是否为模型加入HzPreprocess节点，预处理节点会出现在转换过程产生的所有产物中。

理想状态下，这个HzPreprocess节点应该完成 `input_type_rt` 到 `input_type_train` 的完整转换， 实际情况是整个type转换过程会配合地平线AI芯片硬件完成，ONNX模型里面并没有包含硬件转换的部分。 因此**ONNX的真实输入类型**会使用一种**中间类型**，这种中间类型就是硬件对 `input_type_rt` 的处理结果类型， 数据layout(NCHW/NHWC)会保持和原始浮点模型的输入layout一致。 每种 `input_type_rt` 都有特定的对应中间类型，如下表：

	A	B	C	D	E	F
1	**nv12**	**yuv444**	**rgb**	**bgr**	**gray**	**featuremap**
2	yuv444_128	yuv444_128	RGB_128	BGR_128	GRAY_128	featuremap

> 💡 表格中第一行是 `input_type_rt` 指定的数据类型，第二行是特定 `input_type_rt` 对应的中间类型， 这个中间类型就是模型转换中间产物三个onnx模型的输入类型。每个类型解释如下：
>
> - yuv444_128/RGB_128/BGR_128/GRAY_128为对应 `input_type_rt` 减去128的结果。
> - featuremap 是一个四维张量数据（J5支持非四维），每个数值采用float32表示。

为避免误用，并不是所有的 `input_type_rt` 和 `input_type_train` 的组合都可以支持。依据实际生成经验，目前开放的组合如下：

	A	B	C	D	E	F	G
1		**nv12**	**yuv444**	**rgb**	**bgr**	**gray**	**featurema**
2	**yuv444**	Y	Y	N	N	N	N
3	**rgb**	Y	Y	Y	Y	N	N
4	**bgr**	Y	Y	Y	Y	N	N
5	**gray**	N	N	N	N	Y	N
6	**featuremap**	N	N	N	N	N	Y

此外，如果yaml文件中的norm_type参数配置为data_mean、data_scale、data_mean_and_scale，则预处理节点中还将包含norm操作。

> 💡 请注意：yaml文件中的mean和scale参数与训练时的mean、std需要进行换算。

HzPreprocess节点中的计算公式为： $norm\_data = (data - mean) * scale$ ，以yolov3为例，其训练时的预处理代码为：

```
def base_transform(image, size, mean, std):
    x = cv2.resize(image, (size, size)).astype(np.float32)
    x /= 255.
    x -= mean
    x /= std
    return x


class BaseTransform:
    def __init__(self, size, mean=(0.406, 0.456, 0.485), std=(0.225, 0.224, 0.229)):
        self.size = size
        self.mean = np.array(mean, dtype=np.float32)
        self.std = np.array(std, dtype=np.float32)
```

则计算公式为： $norm\_data = (\frac{data}{255} - mean) * \frac{1}{std}$ ，改写为HzPreprocess节点的计算方式：

$$norm\_data = (\frac{data}{255} - mean) * \frac{1}{std} = (data - 255mean) * \frac{1}{255std}$$

则： $mean\_yaml = 255mean$、$scale\_yaml = \frac{1}{255std}$

1.2.2 预处理节点与前处理代码

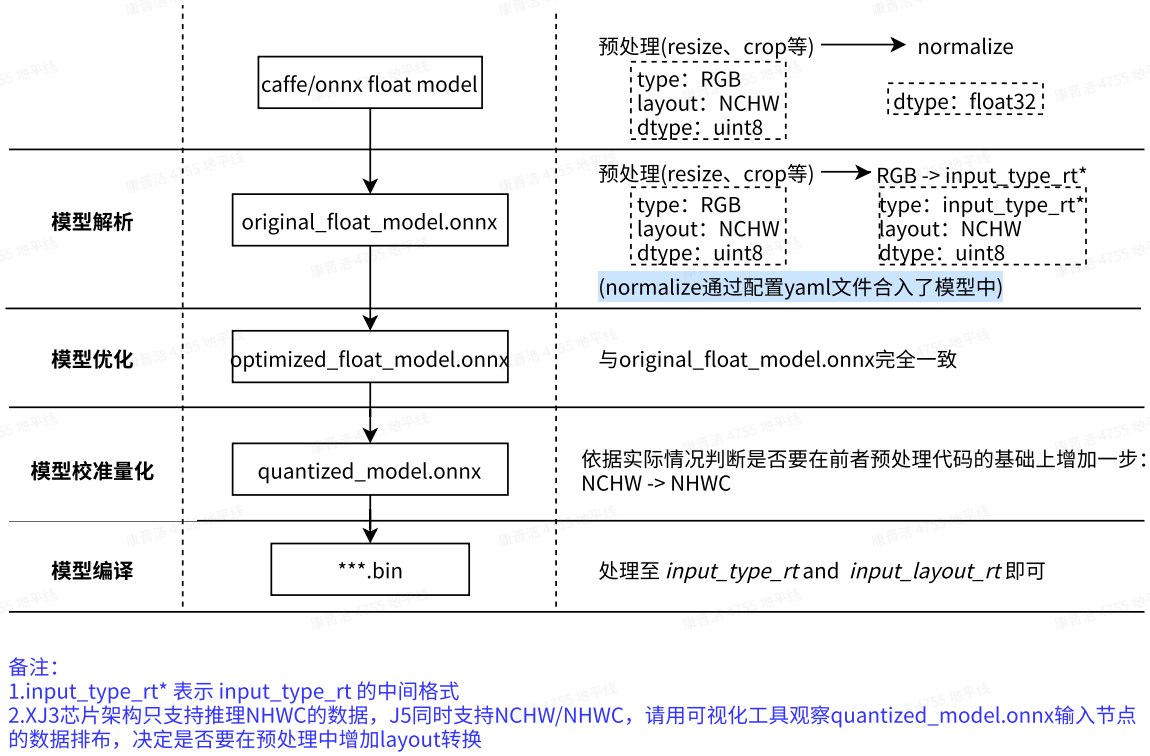
由于HzPreprocess节点的存在，使得转换生成的模型其前处理会与原始模型有所差异。总的来说，有两点需要注意：

- 推理转换的中间模型（original_float_model.onnx / optimized_float_model.onnx / quantized_model.onnx），需要在预处理时将输入数据处理至 `input_type_rt` 的中间类型

（-128的操作请通过配置onnx模型推理API的`input_offset`参数实现，该参数的应用可参考发布包中任意转换示例或前文1.1 验证浮点onnx模型正确性）；

- 注意不要与HzPreprocess做重复的norm操作。

各阶段模型预处理示例如下图所示：



> 校准数据只需处理到input_type_train即可，同时也要注意不要做重复的norm操作。

1.3 python端与板端一致性校验

1.3.1 hb_model_verifier工具介绍

> 💡 我们已经对该工具进行了升级，升级后的新工具为`hb_verifier`工具（J5 OE1.1.40/XJ3 OE2.5.2及以上版本可支持），我们推荐您优先使用新版工具，当前工具会在之后的版本中逐渐弃用。

`hb_model_verifier`工具是用于对指定的定点onnx模型和bin模型进行结果验证的工具。 该工具会使用指定图片（若未指定图片，则工具会用默认图片进行推理，featuremap模型会使用随机生成的tensor数据），进行定点模型推理，bin模型板端和x86端模拟器上的推理， 并对其三方的结果进行两两比较，给出是否通过的结论。

bin模型在板端的推理需确保给定ip可以ping通且板端已经安装`hrt_tools`，若无则可以使用OE包中`ddk/package/board`下的`install.sh`脚本进行安装；bin模型在x86端的推理需确保

host端已经安装 `hrt_tools`，若无则可以使用OE包中 `ddk/package/host` 下的 `install.sh` 脚本进行安装。

1.3.1.1 工具简介

1. **参数说明**

```Python
hb_model_verifier -q ${quanti_model} \
                  -b ${bin_model} \
                  -a ${board_ip} \
                  -i ${input_img} \
                  -d ${digits}
```

`--quanti_model, -q`

定点模型名称。

`--bin_model, -b`

bin模型名称。

`--arm-board-ip, -a`

上板测试使用的arm board ip地址。

`--input-img, -i`

推理测试时使用的图片。 若不指定则会使用默认图片或随机tensor。 对于二进制形式的图片文件需要后缀名为 `.bin` 形式。

`--compare_digits, -d`

比较推理结果数值精度，若不指定则会默认比较小数点后五位。

2. **输出内容解析**

结果对比最终会在终端展示, 工具会对比ONNX模型运行结果, 模拟器运行及上板结果的两两对比情况, 若无问题应显示如下:

```Shell
Quanti onnx and Arm result Strict check PASSED
```

在定点模型和runtime模型精度不一致时会输出不一致结果的信息并提示check FAILED。

1.3.1.2 使用示例

YAML

```yaml
# 随机数据
hb_model_verifier -q quanti.onnx -b model.bin -a 10.10.10.10
# 指定数据
hb_model_verifier -q quanti.onnx -b model.bin -a 10.10.10.10 -i data.bin
```

💡
1. `hb_model_verifier` 目前只支持单输入模型。
2. 若模型有多个输出，则只会比较第一个输出的结果情况。
3. 暂时不支持对已打包的*.bin模型进行验证

1.3.2 hb_verifier工具介绍

`hb_verifier` 工具是用于对指定的定点onnx模型和bin模型进行结果验证的工具。 该工具会使用指定图片（若未指定图片，则工具会用默认图片进行推理，featuremap模型会使用随机生成的tensor数据），进行定点模型推理，bin模型板端和x86端模拟器上的推理， 并对其三方的结果进行两两比较，给出是否通过的结论。同时该工具支持比对移除了Dequantize节点的.bin模型与定点onnx模型的比对。

1.3.2.1 工具简介

1. **参数说明**

Shell

```shell
hb_verifier -m   ${quanti_model},${bin_model} \
            -b   ${board_ip} \
            -s   True / False \
            -i   ${input_img} \
            -c   ${digits}  \
            -r   True / False
```

`--model/-m`

定点模型名称和bin模型名称，多模型之间用”,”进行区分。

`--board-ip/-b`

上板测试使用的arm board ip地址。

`--run-sim/-s`

设置是否使用X86环境的libdnn做bin模型推理，默认为False。

- 当该参数设置为 `True` 时，工具将会使用x86环境的libdnn做bin模型推理。

- 当该参数设置为 `False` 时，工具不会使用x86环境的libdnn做bin模型推理。

`--input-img/-i`

指定推理测试时使用的图片。

若不指定则会使用随机生成的tensor数据。

若指定图片为二进制形式的图片文件，其文件形式需要为后缀名为 .bin 形式。

多输入模型添加图片的方式有以下两种传参方式，多张图片之间用”,”分割：

- input_name1:image1,input_name2:image2, …
- image1,image2…

`--compare_digits/-c`

设置比较推理结果的数值精确度（即比较数值小数点后的位数），若不进行指定则工具会默认比较至小数点后五位。

`--dump-all-nodes-results/-r`

设置是否保存模型中各个算子的输出结果，并对算子输出名称相同的结果进行对比，默认为False。

- 当该参数设置为 True 时，工具将会获取模型中所有节点的输出，并根据节点输出的名字做匹配，从而进行对比。出于性能考虑，暂不支持在X86环境下使用 dump 功能。
- 当该参数设置为 False 时，工具将会只获取模型最终输出的结果，并进行对比。

2. **输出内容解析**

结果对比最终会在终端展示，工具会对比多个模型在不同场景下的运行结果，若无问题应显示如下:

```
Shell
1  Quanti.onnx and Arm result Strict check PASSED
```

在定点模型和runtime模型精度不一致时会输出不一致结果的信息并提示check FAILED。

1.3.2.2 使用示例

1. `quanti.onnx` 模型推理、 `.bin` 模型板端推理、 `.bin` 模型x86端推理结果对比：

```
Shell
1  hb_verifier -m quanti.onnx,model.bin -b *.*.*.* -s True
```

2. `quanti.onnx` 模型推理与 `.bin` 模型板端推理结果对比：

```
Shell
1  hb_verifier -m quanti.onnx,model.bin -b *.*.*.*
```

3. `quanti.onnx` 模型推理与 `.bin` 模型在X86端推理结果对比：

```
Shell
1 hb_verifier -m quanti.onnx,model.bin -s True
```

4. `.bin` 模型在板端和端推理结果对比：

```
Shell
1 hb_verifier -m model.bin -b *.*.*.* -s True
```

5. 保存 `quanti.onnx` 模型推理、 `.bin` 模型板端推理过程中各个算子的输出，并对算子输出名称相同的结果进行对比：

```
Shell
1 hb_verifier -m quanti.onnx,model.bin -b *.*.*.* -r True
```

1.3.3 hrt_model_exec infer工具介绍

1.3.3.1 参数说明

hrt_model_exec infer命令用于模型推理，使用用户自定义输入数据，推理一帧。用户通过 `input_file` 指定输入数据路径，若为图片，工具将根据模型信息resize图片，整理模型输入信息。

该命令也会输出单线程运行单帧的模型运行时间。

	A	B
1	**可选参数**	**说明**
2	**core_id**	指定模型推理的核id，0：任意核，1：core0，2：core1；默认为 0。
3	**roi_infer**	使能resizer模型推理；若模型输入包含resizer源，设置为 true，默认为 false。
4	**roi**	roi_infer 为 true 时生效，设置推理resizer模型时所需的 roi 区域以分号间隔。
5	**frame_count**	设置 infer 运行帧数，单帧重复推理，可与 enable_dump 并用，验证输出一致性，默认为 1。
6	**dump_intermediate**	dump模型每一层输入数据和输出数据，默认值 0，不dump数据。 1：输出文件类型为 bin； 2：输出类型为 bin 和 txt，其中BPU节点输出为aligned数据； 3：输出类型为 bin 和 txt，其中BPU节点输出为valid数据。
7	**enable_dump**	dump模型输出数据，默认为 false。
8	**dump_precision**	控制txt格式输出float型数据的小数点位数，默认为 9。
9	**hybrid_dequantize_process**	控制txt格式输出float类型数据，若输出为定点数据将其进行反量化处理，目前只支持四维模型。
10	**dump_format**	dump模型输出文件的类型，可选参数为 bin 或 txt，默认为 bin。
11	**dump_txt_axis**	dump模型txt格式输出的换行规则；若输出维度为n，则参数范围为[0, n], 默认为 4。
12	**enable_cls_post_process**	使能分类后处理，目前只支持ptq分类模型，默认 false。

1.3.3.2 使用示例

1. 普通模型

Apache

```
hrt_model_exec infer --model_file=xxx.bin --input_file=xxx.jpg --enable_dump 1
--dump_format txt
```

2. resizer模型（J5 OE1.1.29后可支持，XJ3 OE2.4.2后可支持）

Bash

```
./hrt_model_exec infer --model_file=xxx.bin --input_file= xxx.jpg --
roi="2,4,123,125" --roi_infer=true --enable_dump 1 --dump_format txt
```

3. 移除了反量化节点的模型，仍然输出反量化后的浮点结果（J5 OE1.1.37后可支持，XJ3 OE2.5.2后可支持）

Nginx

```
hrt_model_exec infer --model_file=xxx.bin --input_file=xxx.jpg --
hybrid_dequantize_process 1 --enable_dump 1 --dump_format txt
```

1.4 精度调优建议

经过大量实际生产经验验证，如果能筛选出最优的量化参数组合，地平线的转换工具在大部分情况下，都可以将精度损失保持在1%以内。依据精度损失情况，可参照以下建议进行解决：

1.4.1 精度损失明显（4%以上）

若模型精度损失大于4%，通常是因为yaml配置不当，校验数据集不均衡等导致的，建议依次从pipeline、模型转换配置、一致性检查三个方面进行排查。

1. **pipeline及模型转换配置检查**

pipeline是指您完成数据准备、模型转换、模型推理、后处理、精度评测Metric的全过程。前文精度验证推荐的前两个步骤可以帮助您排查前后处理及yaml文件配置是否有误，精度评测Metric则需要您确保与原始浮点模型完全一致。

2. **数据处理一致性检查**

该部分检查主要针对参考OE开发包示例准备校准数据以及评测代码的用户，主要有以下常见错误：

- **未正确指定read_mode**：02_preprocess.sh中可通过`--read_mode`参数指定读图方式，支持opencv及skimage。此外preprocess.py中亦是通过`imread_mode`参数设定读图方式，也需要做出修改。使用`skimage`的图片读取，得到的是RGB通道顺序，取值范围为0~1，数值类型为float；而使用`opencv`，得到的是BGR通道顺序，取值范围为0~255，数据类型为uint8。
- **校准数据集的存储格式设置不正确**：目前我们采用的是numpy.tofile来保存校准数据，这种方式不会保存shape和类型信息，因此如果input_type_train为非featuremap格式，需要通过yaml中参数`cal_data_type`来设置二进制文件的数据存储类型。若为J5-OE1.1.16以及XJ3-OE1.13.3以前的版本，则会通过校准数据存放路径是否包含“f32”来判断数据dtype，若包含f32关键字，则按float32解析数据；反之则按uint8解析数据。
- **transformer实现方式不一致**：地平线提供了一系列常见预处理函数，存放在`/horizon_model_convert_sample/01_common/python/data/transformer.py`文件中，部分预处理操作的实现方式可能会有所区别，例如ResizeTransformer，我们采用的是opencv默认插值方式（linear），若为其他插值方式可直接修改`transformer.py`源码，确保与训练时预处理代码保持一致。

3. **精度debug工具**

精度debug工具计划于23年5月释放，包含节点敏感度分析、模型误差累计曲线绘制、节点统计量等分析功能。

1.4.2 精度损失较小（1.5%-3%）

为降低模型精度调优的难度，我们建议您优先尝试将`calibration_type`配置为`default`。default为自动搜索功能，以第一张校准数据输出节点余弦相似度为依据，从max、max-Percentile 0.99995和KL等校准方法中选取最优的方案，最终选取的校准方法可关注转换日志类似“*Select kl method.*”的提示。若自动搜索的精度结果仍然与预期有差距，可尝试以下建议进行调优：

1. **调整校准方式**

a. 手动指定 `calibration_type`，选择mix；（mix校准会先使用kl校准方式量化模型，取余弦相似度低于0.999的节点作为敏感节点，再分别使用max、max0.99995校准这些节点，取余弦相似度最佳的校准方式得到混合校准模型）

b. 将 `calibration_type` 配置为 `max`，并配置 `max_percentile` 为不同的分位数（取值范围是0-1之间），我们推荐您优先尝试 `0.99999`、`0.99995`、`0.9999`、`0.9995`、`0.999`，通过这五个配置观察模型精度的变化趋势，最终找到一个最佳的分位数；

c. 在前面尝试的基础上选择余弦相似度最高的方案，尝试启用 `per_channel`。

2. **调准校准数据集**

a. 可以尝试适当增加或减少数据数量（通常来说检测场景相较于分类场景需要的校准数据要少；此外可以观察模型输出的漏检情况，适当增加对应场景的校准数据）；

b. 不要使用纯黑纯白等异常数据，尽量减少使用无目标的背景图作为校准数据；尽可能全面的覆盖典型任务场景，使得校准数据集的分布与训练集近似。

3. **将部分尾部算子回退到 CPU 高精度计算**

a. 一般我们仅会尝试将模型尾部输出层的 1～2 个算子回退至 CPU，太多的算子会较大程度影响模型最终性能，判断依据可通过观察模型的余弦相似度；（若将某些中间节点run_on_cpu，发现精度没有提升，这是正常现象，因为反复重量化可能还会带来更大的精度损失，因此通常只建议将尾部节点回退cpu）

b. 指定算子运行在 CPU 上请通过yaml文件中的 `run_on_cpu` 参数，通过指定节点名称将对应算子运行在cpu上（参考示例：run_on_cpu: conv_0）。

c. 若run_on_cpu之后模型编译报错，请直接联系地平线技术支持人员