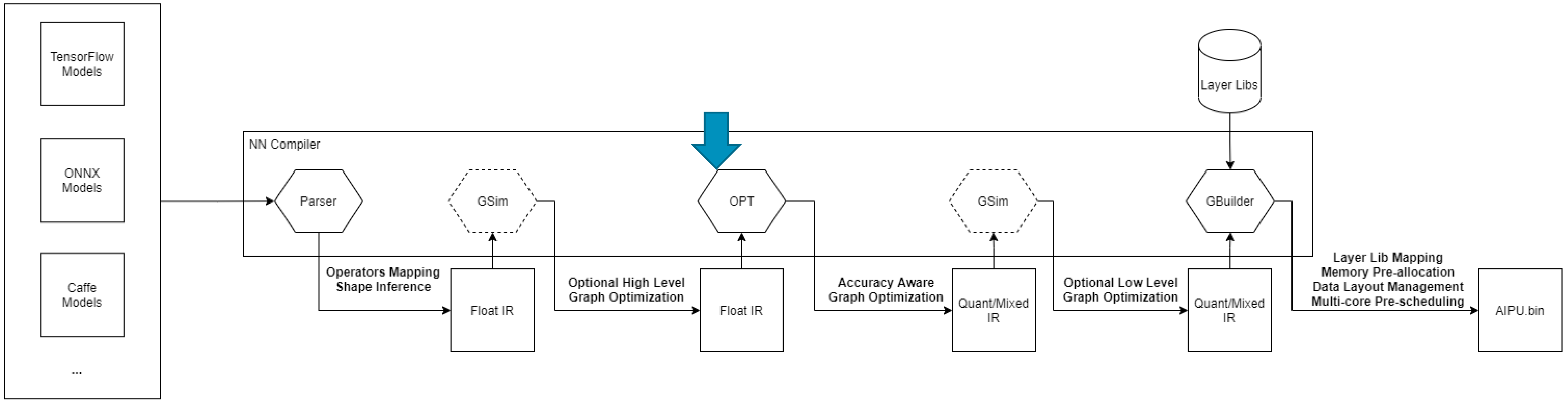
安谋科技
arm CHINA

Zhouyi Compass OPT Tutorial

Contents

Background: The Role of OPT

- **opt**imizer is part of the Zhouyi Compass Neural Network Compiler (the Python package name is AIPUBuilder). It is designed for converting the float Intermediate Representation (IR) generated by ‘Compass Unified Parser’ to an optimized quantized or mixed IR (through techniques like quantization and graph optimization) which is suited for Zhouyi NPU hardware platforms.



Background: How to Optimize a Model

- ➢Less params (pruning):
  - SVD
  - pruning filters
  - knowledge distillation
- ➢More zeros (sparsification):
  - structured/unstructured sparsification
  - zero value reordering
- ➢Less operations (graph optimization):
  - constant folding
  - op fusion
  - arithmetic optimization
- ➢Less bits (quantization):
  - quantization aware training
  - post-training quantization

安谋科技
arm CHINA

Background: How to Optimize a Model

Quantization: *almost the most widely used easy way to simplify a model*

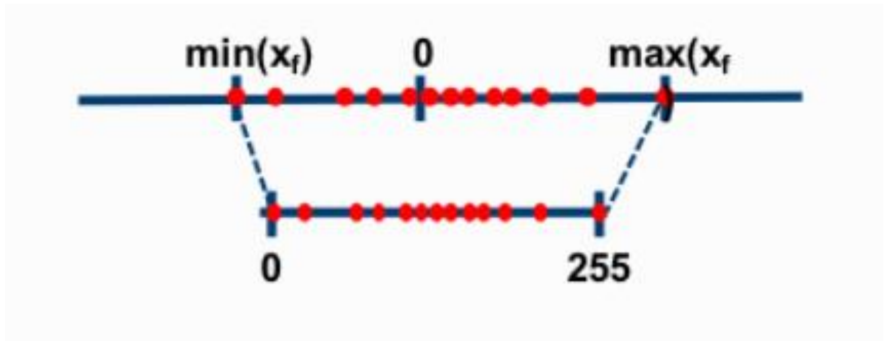
- ➢ Less dependent on training
  - Quantization aware training usually needs few rounds of fine-tuning
  - Post-training quantization is totally independent of training
- ➢ Less harmful to model accuracy
  - Quantization aware training can usually matches the float model’ s accuracy
  - Post-training quantization can usually narrow the accuracy drop to acceptable level
- ➢ Less costly to hardware resources
  - Reducing model memory footprint and storage size
  - Exploiting low-cost integer math hardware units
- ➢ Widely supported by main deep learning frameworks and AI chips

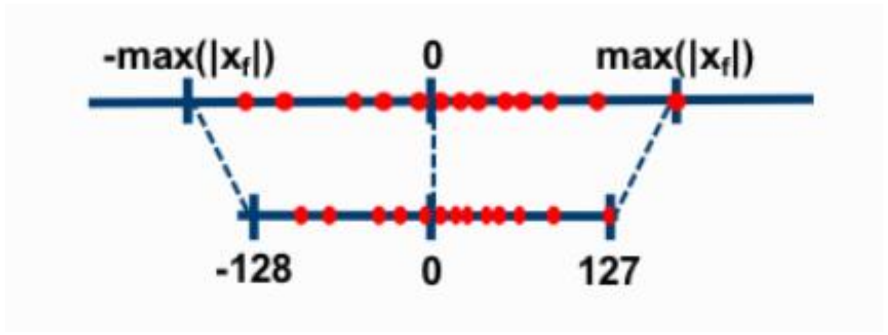
Background: How to Optimize a Model

Quantization: definition

maps floating-point weights and activations in a trained model to low-bitwidth integer values

$$x_q = round\left((x_f - min_{x_f})\underbrace{\frac{2^n-1}{max_{x_f} - min_{x_f}}}_{s_x}\right)$$
$$\approx round(s_x x_f - \underbrace{round(min_{x_f} s_x)}_{zp_x})$$
$$= round(s_x x_f - zp_x)$$





- *Sx is the scale factor, and ZPx is the offset value (also known as zero point, corresponding to the 0.0 in floating domain)*
- *The formula can also be written as:* $x'_f = (x'_q - zp'_x) * s'_x$.

  Its equivalent, where $s'_x = 1/s_x, zp'_x = -zp_x$.
- *The corresponding dequantization formula is:* $x''_f = (x_q + zp_x)/s_x$

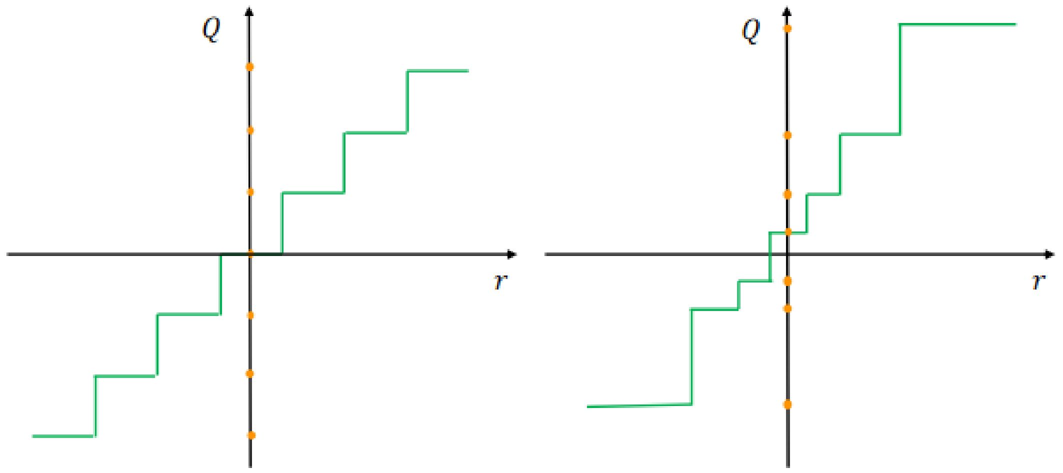
  Note that x''_f *will not exactly match* x_f *and quantization is lossy.*

安谋科技 arm CHINA

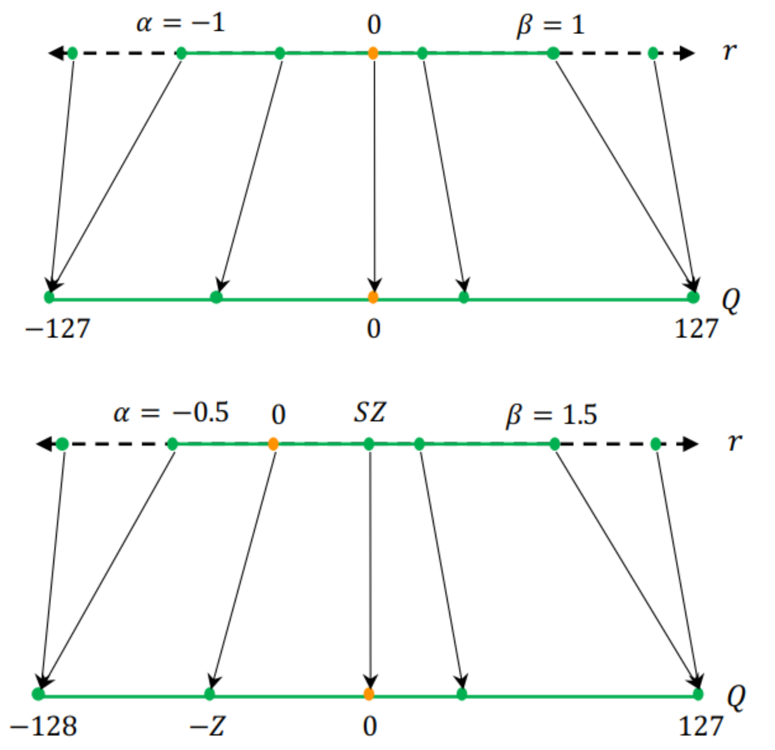
Background: How to Optimize a Model

Quantization: some concepts [1]

uniform quantization vs **non-uniform quantization**



asymmetric quantization vs **symmetric quantization**



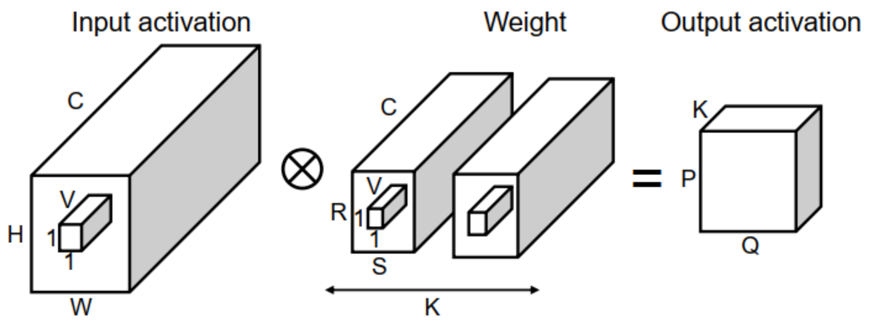
Background: How to Optimize a Model

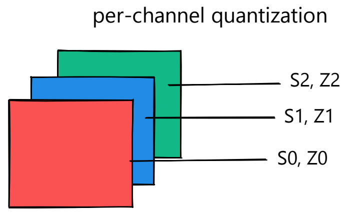
Quantization: some concepts [1]

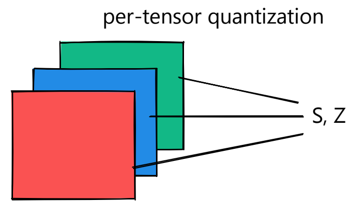
Full Range vs **Restricted Range** in symmetric quantization

	Full Range	Restricted Range
Quantized Range	$[-2^{n-1}, 2^{n-1}-1]$	$[-(2^{n-1}-1), 2^{n-1}-1]$
8-bit example	[-128, 127]	[-127, 127]
Scale Factor	$s_x = \frac{(2^n-1)/2}{max(abs(x_f))}$	$s_x = \frac{2^{n-1}-1}{max(abs(x_f))}$

Quantization granularity: **per-tensor/per-layer** vs **per-(output)-channel** vs **per-vector** [2]







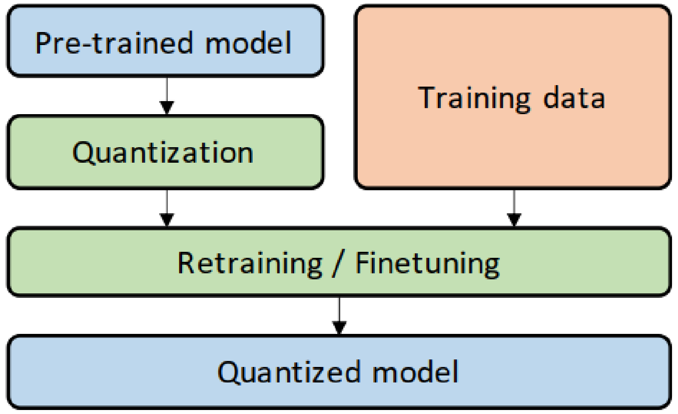
Background: How to Optimize a Model

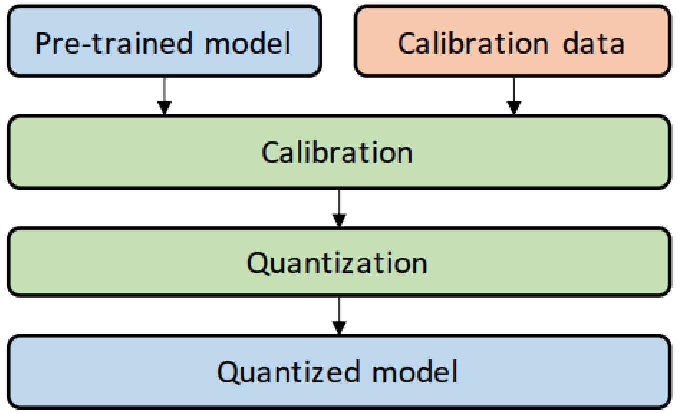
Quantization: some concepts [1]

Static/offline quantization vs **dynamic/online quantization**

- *Dynamic quantization dynamically computes the clipping ranges for each input and often achieves higher accuracy with a high computation overhead*
- *Static quantization statically computes the clipping ranges for all inputs and fixes them during inference.*

Quantization aware training vs **post-training quantization**





安谋科技
arm CHINA

Background: How to Optimize a Model

Quantization: some concepts [1]

Zero shot (aka data free) quantization

- *Performs the entire quantization without any access to the original training/validation data*
- *A popular branch of research in ZSQ is to generate synthetic data similar to the real data from which the target pre-trained model is trained, the synthetic data is then used for calibrating and/or finetuning the quantized model*

Stochastic quantization in QAT

- *For example, maps the floating number up or down with a probability associated to the magnitude of the weight update*
- *For example, quantizes a different random subset of weights during each forward pass and trains the model with unbiased gradients*

Extreme (low-bit precision) quantization

- *Quantization below 8 bits. For example, binarization*

Background: How to Quantize an Operator

Convolution: $y_f = Conv(x_f) = w_f x_f + b_f$

According to the quantization formula, we have:

$$(y_q + zp_y)/s_y = ((w_q + zp_w)/s_w)((x_q + zp_x)/s_x) + (b_q + zp_b)/s_b$$

$$y_q = \left((w_q + zp_w)(x_q + zp_x) + (b_q + zp_b)s_w\, s_x/s_b\right) s_y/s_w\, s_x - zp_y$$

Then let $s_b = s_w s_x, zp_b = 0$ for simplicity,

and replace float value $s_y/s_w\, s_x$ with approximate form: $M/2^N$, where M and N are integers.

Finally we got the integer-only computation flow:

$$y_q = \left((w_q + zp_w)(x_q + zp_x) + b_q\right) M/2^N - zp_y$$

Note:

- zp_x related computation is independent of inputs, and can be done offline and stored in biases
- b_q is represented with higher bits (32bits in int8 quantization, 48bits in int16 quantization is recommended in OPT) as each bias-vector entry is added to many output activations, any quantization error in the bias-vector tends to act as an overall bias[3]

安谋科技 arm CHINA

Background: How to Quantize an Operator

Nonlinear activation: y = f(x)

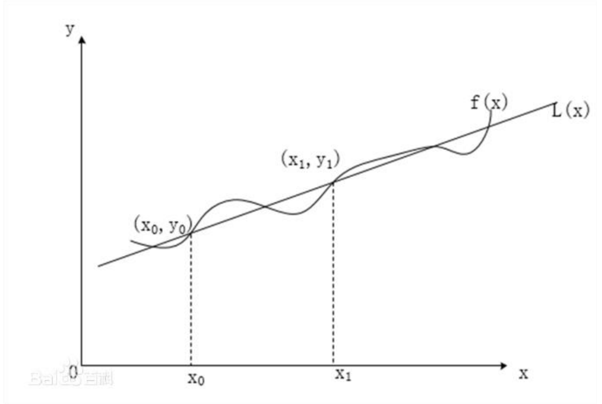
$$(y_q + zp_y)/s_y = f\left((x_q + zp_x)/s_x\right)$$

$$y_q = f\left((x_q + zp_x)/s_x\right)s_y - zp_y$$

Because x_q is discrete and has limited scope, so we can use a lookup table to store the computed result:

1. Construct an uniform sampling set {X} of x_q
2. Dequantize {X} to {Xf}
3. Calculate {Yf} = f({Xf})
4. Quantize {Yf} to {Y} as the final lookup table

During inference, linear interpolation is used when the table size is smaller than the input range.



Note:

- The size of lookup table is positively related to accuracy and negatively related to computation overhead.
- OPT uses the 'lut_items_in_bits' field to control the size (with 2**lut_items_in_bits items) of the lookup table.
- OPT recommends to set 'lut_items_in_bits' to 10+ when quantizing activations to 16bit.

Background: How to Quantize an Operator

Nonlinear activation: clip, relu (aka, clip(0, +00)), relu6 (aka, clip(0, 6))

$$y_f = Clip\left(x_f, cmin_f, cmax_f\right)$$

$$\left(y_q + zp_y\right)/s_y = Clip\left(\left(x_q + zp_x\right)/s_x\,, cmin_f, cmax_f\right)$$

$$y_q = Clip\left(\left(x_q + zp_x\right)/s_x\,, cmin_f, cmax_f\right)s_y - zp_y$$

$$y_q = Clip\left(\left(x_q + zp_x\right) s_y/s_x\,,\ cmin_f * s_y,\ cmax_f * s_y\right) - zp_y$$

$$y_q = Clip\left(\left(x_q + zp_x\right) s_y/s_x - zp_y,\ cmin_f * s_y - zp_y,\ cmax_f * s_y - zp_y\right)$$

$$y_q = Clip\left(\left(x_q + zp_x\right) M/2^N - zp_y,\ cmin_q,\ cmax_q\right)$$

Where $M/2^N$ is the approximate form of s_y/s_x

Additionally, to utilize the full bit width, the min/max used to quantize y_f can be directly set to $cmin_f$ and $cmax_f$

So we do not need a lookup table and the computation steps are simple

安谋科技 arm CHINA

Background: How to Quantize an Operator

Pad:

- CONSTANT PAD

$$y_q = Pad\left(\left(x_q + zp_x\right)/s_x, v_f\right)s_y - zp_y$$
$$y_q = Pad\left(\left(x_q + zp_x\right) s_y/s_x, v_f s_y\right) - zp_y$$
$$y_q = Pad\left(\left(x_q + zp_x\right) s_y/s_x - zp_y, v_f s_y - zp_y\right)$$

Then let $s_y = s_x, zp_y = zp_x$ for simplicity

$$y_q = Pad\left(x_q, v_q\right)$$

Where v_q is quantized using y_f's scale and zero point

- REFLECT PAD，SYMMETRIC PAD and REPLICATE PAD

$$y_q = Pad\left(\left(x_q + zp_x\right) s_y/s_x - zp_y\right)$$

let $s_y = s_x, zp_y = zp_x$ for simplicity

$$y_q = Pad\left(x_q\right)$$

安谋科技 arm CHINA

Background: How to Quantize an Operator

Concat: $y_f = Concat\left(x_{1_f}, x_{2_f}, \dots\right)$

$$\left(y_q + zp_y\right)/s_y = Concat\left(\left(x_{1_q} + zp_{x_1}\right)/s_{x_1}, \left(x_{2_q} + zp_{x_2}\right)/s_{x_2}, \dots\right)$$

$$y_q = Concat\left(\left(x_{1_q} + zp_{x_1}\right)/s_{x_1}, \left(x_{2_q} + zp_{x_2}\right)/s_{x_2}, \dots\right) s_y - zp_y$$

$$y_q = Concat\left(\left(x_{1_q} + zp_{x_1}\right) s_y/s_{x_1}, \left(x_{2_q} + zp_{x_2}\right) s_y/s_{x_2}, \dots\right) - zp_y$$

❑ Question: How to decide $s_y, s_{x_1}, s_{x_2}, \dots$ and $zp_y, zp_{x_1}, zp_{x_2}, \dots$ to balance accuracy and performance?

✓ $zp_y = zp_{x_1} = zp_{x_2} = \cdots$, and $s_y = s_{x_1} = s_{x_2} = \cdots$
  - Set unify_scales_for_concat = True in OPT's cfg file to enable

✓ $zp_y, zp_{x_1}, zp_{x_2}, \dots$ = 0, and $s_y = s_{x_1}$

✓ $zp_y, zp_{x_1}, zp_{x_2}, \dots$ = 0, and $s_y = s_{x_i}$, where x_i is the branch that has most elements

✓ Compute them as is

✓ …

安谋科技 arm CHINA

Background: How to Quantize an Operator

EltementwiseAdd: $y_f = a_f + b_f$

$$\left(y_q + zp_y\right)/s_y = \left(a_q + zp_a\right)/s_a + \left(b_q + zp_b\right)/s_b$$

$$y_q = \left(\left(a_q + zp_a\right)s_b + \left(b_q + zp_b\right)s_a\right) s_y/s_a\, s_b - zp_y$$

Replace $s_a, s_b, s_y/s_a s_b$ with $M_1/2^{N_1}, M_2/2^{N_2}, M_3/2^{N_3}$, and assume $N_1 \geq N_2$, then:

$$y_q = \left(\left(a_q + zp_a\right)M_2 2^{N_1-N_2} + \left(b_q + zp_b\right)M_1\right) M_3/2^{N_1+N_3} - zp_y$$

On most of Zhouyi platforms, we should also constrain:

- $0 \leq M_2 2^{N_1-N_2}, M_1, M_3 \leq 32767$
- $N_1 + N_3 \geq 0$

❑ Question: How about EltementwiseSub, EltementwiseMax, EltementwiseMin, and EltementwiseMul?

安谋科技 arm CHINA

Background: How to Quantize an Operator

Some Tips for Quantizing an Operator:

- ✓ Formula derivation is always helpful, and should take the whole operator into consideration
- ✓ Remember the bit-width restrictions (and other hardware constrains) and take care of overflow
- ✓ Avoid computational accuracy loss as much as possible

PS: Take EltementwiseAdd as an example again, why not:

$$y_q = (a_q + zp_a)\, s_y/s_a + (b_q + zp_b)\, s_y/s_b - zp_y \approx (a_q + zp_a)\, M_1/2^{N_1} + (b_q + zp_b)\, M_2/2^{N_2} - zp_y$$

or

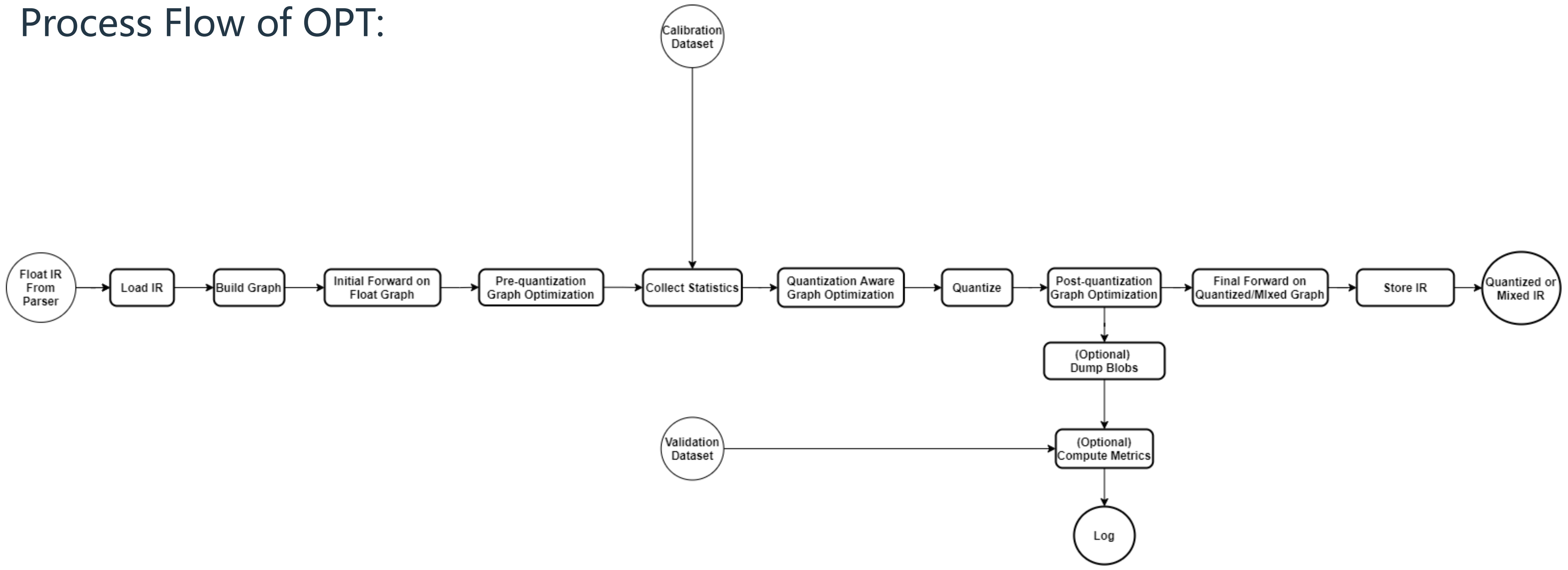
$$y_q = \left((a_q + zp_a) + (b_q + zp_b)\, s_a/s_b\right) s_y/s_a - zp_y \approx \left((a_q + zp_a) + (b_q + zp_b)\, M_1/2^{N_1}\right) M_2/2^{N_2} - zp_y$$

?

安谋科技
arm CHINA

OPT Usage: Common Usage

Process Flow of OPT:


Calibration Dataset
Float IR From Parser
Load IR
Build Graph
Initial Forward on Float Graph
Pre-quantization Graph Optimization
Collect Statistics
Quantization Aware Graph Optimization
Quantize
Post-quantization Graph Optimization
Final Forward on Quantized/MIxed Graph
Store IR
Quantized or Mixed IR
(Optional) Dump Blobs
Validation Dataset
(Optional) Compute Metrics
Log

- OPT usually uses a txt config file as the input. You can run it as 'optimizer_main.py --cfg opt.cfg'
  - All options are under the `Common` section
  - `#` acts as an annotation symbol
- You can use 'optimizer_main.py –field' to check all configurable fields

OPT Usage: Common Usage

```
[Common]
#the paths for this model's IR
graph = ./squeezenet_s.txt
bin = ./squeezenet_s.bin
model_name = squeezenet_caffe
#the name of dataset plugin for this model's input dataset
#if omitted, will use all zeros as input data for executing forward
dataset = numpynhwcrgb2ncbgrhwdataset
#the path of dataset used for calibration during quantization
#if omitted, will use all zeros as input data for executing calibration
calibration_data = ./calibration2.npy
#the batch_size used for calibration during quantization
calibration_batch_size = 1
#the name of metric plugins for computing accuracy metrics for this model
#if ommitted, will not computing accuracy metrics
metric = TopKMetric
#the path of dataset (and corresponding labels) used for computing accuracy metrics for this model
#if ommitted, will not computing accuracy metrics
data = ./validation10.npy
label = ./vlabel10.npy
#the batch_size used for computing accuracy metrics for this model
metric_batch_size = 2
#the quantization method used for weights, default to 'per_tensor_symmetric_restricted_range'
quantize_method_for_weight = per_channel_symmetric_restricted_range
#the quantization method used for activations, default to 'per_tensor_symmetric_full_range'
quantize_method_for_activation = per_tensor_asymmetric
#the bits used for quantizing weight tensors, default to 8
weight_bits = 8
#the bits used for quantizing bias tensors, default to 32
bias_bits = 32
#the bits used for quantizing activation tensors, default to 8
activation_bits = 8
#Maximal LUT items (in bits, as only support LUT with 2**N items) amount when representing nonlinear functions in quantization,
#default to 8, suggest to set to 10+ when quantizing activations to 16bit
lut_items_in_bits = 8
#the output directory path, default to pwd
output_dir = ./
#the dataloader thread numbers for torch dataset, default to 0,
#which means do not using multi-threads to accelerate data loading
dataloader_workers=4
```

安谋科技 arm CHINA

OPT Usage: Enabling Mixed Precision

OPT offers two ways to configure mixed precision quantization:

- Automatically search: Trigger the ‘mixed_precision_auto_search’ option in the cfg file
  - Give the maximum allowed accuracy loss threshold, automatically recommend which layers should be quantized to 8bit, which layers should be quantized to 16bit, and which layers should not be quantized. Must be set as 'batches, threshold, greater_equal_or_less_equal_than_baseline' (batches < 1 means disable). default value='0,0.,L'
  - '1,0.02,L’ (use 1 batch data to metric, and score(baseline model) - score(target model) <= 0.02)
  - '1,-0.02,G' (use 1 batch data to metric, and score(baseline model) - score(target model) >= -0.02)
  - More batches of data (taken from the validation dataset) used for search, more accurate the result will be, but more time will be consumed
  - If the result contains layers which should not be quantized (aka float layers), and the target platform does not support float type operators, you may need to manually schedule the IR (on the CPU and NPU)
- Manually set
  - Through the JSON file (generated by OPT), you can freely configure the quantization hyper-parameters of each layer (see the next page)

OPT Usage: Per-layer Configuration

OPT will generate a JSON template file each run time, and you could copy and edit this file to configure per-layer hyper-parameters (<u>will be treated with higher priority when conflicting with global configurations in the cfg file</u>), through the `opt_config` field in the cfg file.

Typical content for a layer (with the layer name from input IR as its primary key) in this JSON file is:

```
    "fc2/MatMul": {
        "q_mode_activation": "per_tensor_symmetric_full_range",
        "q_mode_weight": "per_tensor_symmetric_restricted_range",
        "q_bits_activation": 8,
        "q_bits_weight": 8,
        "q_bits_bias": 32,
        "lut_items_in_bits": 8,
        "force_dtype_int": false,
        "force_shift_positive": false,
        "q_strategy_activation": "mean",
        "q_strategy_weight": "extrema",
        "running_statistic_momentum": 0.9,
        "histc_bins": 2048,
        "just_for_display": {
            "quantization_info": "{'fc2/MatMulweights': {'scale': '3893.22265625', 'zerop': '0.0', 'qbits':
'8', 'dtype': 'Dtype.INT8', 'qmin': '-127', 'qmax': '127', 'qinvariant': 'False'}, 'fc2/MatMulbiases':
{'scale': '13716.759067741223', 'zerop': '0', 'qbits': '32', 'dtype': 'Dtype.INT32', 'qmin': '-2147483648',
'qmax': '2147483647', 'qinvariant': 'False'}, 'fc2/MatMul': {'scale': '14.128936767578125', 'zerop': '0.0',
'qbits': '8', 'dtype': 'Dtype.UINT8', 'qmin': '0', 'qmax': '255', 'qinvariant': 'False', 'similarity':
0.9967884385773828}}",
            "optimization_info": "{}",
            "brief_info": "layer_id = 21, layer_type = OpType.FullyConnected, similarity=0.9967884385773828, "
        }
    },
```

- `q_mode_activation`: quantization method for activations
- `q_mode_weight`: quantization method for weights
- `q_bits_activation`: quantization bit-width for activation
- `q_bits_weight`: quantization bit-width for weight
- `q_bits_bias`: quantization bit-width for bias
- `lut_items_in_bits`: Maximal LUT items (2**N items) amount
- `force_dtype_int`: whether to force layer top tensors to be quantized to int types
- `force_shift_positive`: whether to force requantization parameter 'shift' to be positive
- `q_strategy_activation`: local strategy used to calibrate activations
- `q_strategy_weight`: local strategy used to calibrate weights
- `running_statistic_momentum`: used for calculating weighted average of some statistics
- `histc_bins`: bins when statistic histograms
- `just_for_display`: read-only messages for debugging, unconfigurable

OPT Usage: Choose Calibration Strategy

OPT divides calibration strategies into two categories:

- Local calibration:
  - Only affect single layers
  - Support per-layer configuration
  - Weights and Activations can be configured differently
  - Usually deterministic and efficient

- Global calibration:
  - Affect multi-layers or the whole network
  - Do not support per-layer configuration
  - Weights and Activations are configured together
  - Maybe nondeterministic and time-consuming

OPT will apply global calibration after applying local calibration

OPT Usage: Choose Calibration Strategy

Local calibration is configured through the ` calibration_strategy_for_weight ` and ` calibration_strategy_for_activation` fields in the cfg file, or the `q_strategy_weight` and ` q_strategy_activation` fields in JSON per-layer configurations.

Usually the default strategy is general enough, and you can carefully choose from the following list:

Name	Valid Values	Description	Tips
Extrema	`extrema`	Default strategy for weights. naïve minimum/maximum values	Usually for weights

Averaging	`mean`	Default strategy for activations.
		min_value = momentum * pre_batch_min_value + (1-momentum) * cur_batch_min_value
		max_value = momentum * pre_batch_max_value + (1-momentum) * cur_batch_max_value
			Usually for activations and may get better results when tuning `momentum`

Percentile	`[R]percentile`	Where R is one positive real number and defaults to 1.0 min_value = extrema_min * R, max_value = extrema_max * R	Usually for layer-wise tuning
STD	`[N]std`	Where N is one positive integer and defaults to 1 min_value = mean_value - N * std_value, max_value = mean_value + N * std_value	Usually for layer-wise tuning
KLD	`[N]kld`	Refer to https://on-demand.gputechconf.com/gtc/2017/presentation/s7310-8-bit-inference-with-tensorrt.pdf (pick threshold which minimizes KL_divergence(ref_distr, quant_distr)) N means picking the maximum threshold among those topN thresholds that minimize KL_divergence(ref_distr, quant_dist), defaults to 1 (aka original kld algo results. When N is large enough, will be equivalent to disabling kld algo)	Usually for activations and may get better results when tuning `N` and/or `histc_bins`
ACIQ	`[R]aciq_laplace` or `[R]aciq_gauss`	Refer to https://arxiv.org/abs/1810.05723 R is one positive real number and defaults to 1.0, acting as a fine tuning coefficient which will multiply on the original optimal threshold	Useful for extreme quantization, e.g. 4-bit quantization
Weighted Scale	`weighted_scale_param[r1, r2, r3, r4]`	Calculate the scale as alpha * pos_scale + (1.0-alpha) * neg_scale, where pos_scale and neg_scale are scales calculated with only positive values or only negative values. Where r1, r2, r3, and r4 are positive real numbers used to decide the alpha value. Setting r1 > 1.0 will search the optimal r1, r2, r3, and r4 automatically (will be recorded in the debug level log)	Usually for per-tensor quantization of weights, may tune params based on auto searched initial values
Customized	`in_ir`	Will use customized min/max values that are recorded in the input IR. (aka `layer_top_range`, `weights_range` and `biases_range` fields)	

OPT Usage: Choosing Calibration Strategy

Global calibration is configured through the ` global_calibration ` field in the cfg file, and is disabled by default. You can carefully choose from the following list:

Name	Valid Values	Description	Tips
Easy Quant	\`easy_quant\` or \`easy_quant[batches, epochs, alpha, beta, nsteps, ngroups]\`	Refer to https://arxiv.org/pdf/2006.16669.pdf The optional parameters: \`batches\` means that how many (calibration) data batches will be used. \`epochs\` means the maximum epochs to iterate. \`alpha\`, \`beta\` and \`nsteps\` are used to 'linearly divide interval of [αS, βS] into n candidate option' . \`ngroups\` means groups which will be divided into when meeting per-channel quantization parameters to speed up (0 means no speedup).	May get very time-consuming especially under per-channel situations (remember to increase \`ngroups\` to speed up)
Adaround	\`adaround\` or \`adaround[batches, epochs, batch_size, lr, reg_param, beta_start, beta_end, warm_start]\`	Refer to https://arxiv.org/pdf/2004.10568.pdf The optional parameters: \`batches\` means that how many (calibartion) data batches will be used. \`epochs\` means the maximum epochs to iterate. \`lr\` means the learning rate. \`reg_param\`, \`beta_start\`, \`beta_end\`, and \`warm_start\` are used to decide the regularization term of the loss function during iteration.	May use a high learning rate to get better results when the size of tuning dataset is limited
Adaquant	\`adaquant_zy\` or \`adaquant_zy[batches, epochs, batch_size, lr_weight, lr_bias, lr_quant_param_weight, lr_quant_param_activation]\`	Refer to https://arxiv.org/abs/2006.10518	May use high learning rates to get better results when the size of tuning dataset is limited
SVD Quant	\`svd_quant\` or \`svd_quant[mode, alpha, beta, nsteps, thresh]\`	SVD based denoising strategy. Refer to AIPUBuilder/Optimizer/features/calibration/global_calibration/svd_based_quant.py	Usually useful for models with some norm operators, e.g. transformer based networks

安谋科技
arm CHINA

OPT Usage: Graph Tiling

To increase the degree of parallelism for multi-core situation or reduce the feature map size for saving memory footprint, you can perform feature map tiling or image tiling (as the h and w dimensions are usually tiled).

- featuremap_tiling_param [**optional**]

  The featuremap_tiling_param is used to assign tiling configurations for specific layers (corresponding to layer_id in float IR): '[(i,j,h,w)]' or '[(i,j,h,w),...,(k,l,x,y)]', where 'i,j' and 'k,l' mean that the corresponding layers in the range will be tiled. 'h,w' and 'x,y' mean the number of parts that the feature map will be split into in the corresponding dimension.

- featuremap_splits_item_x [**optional**]

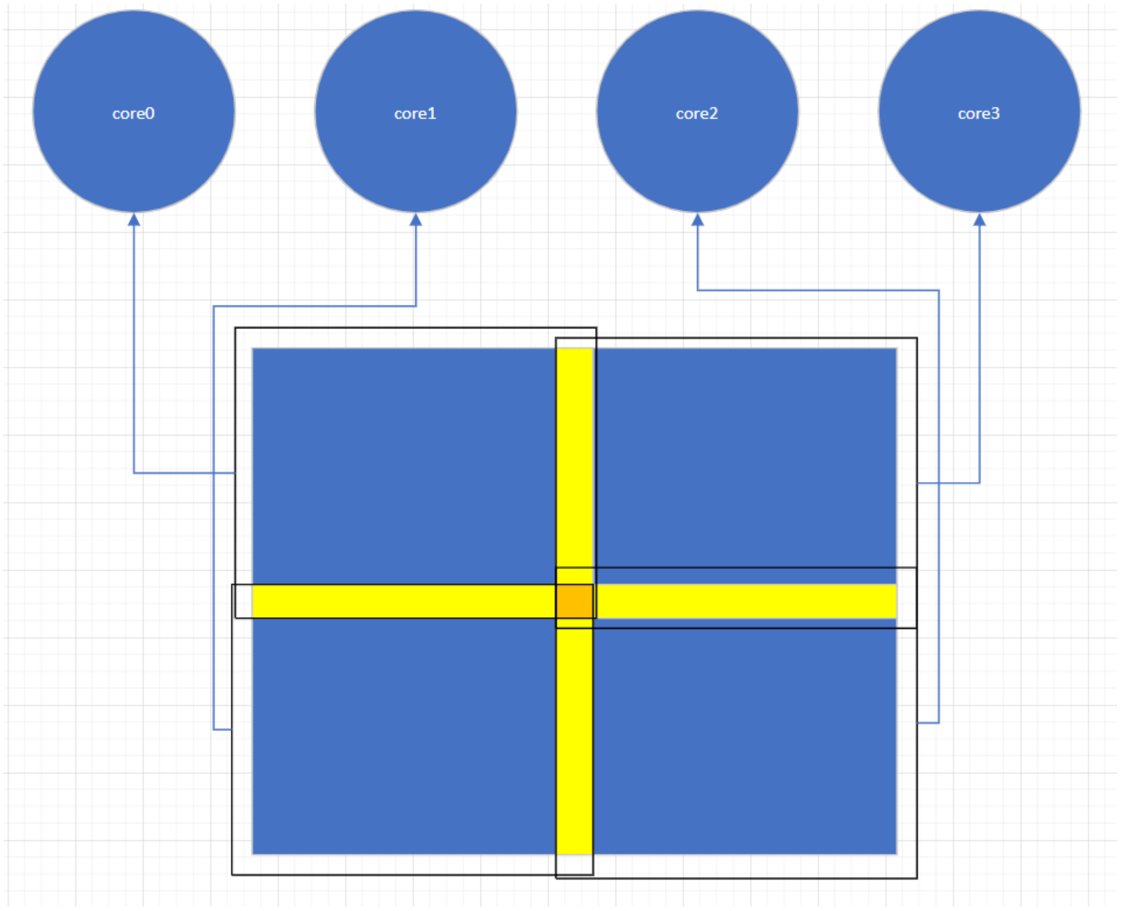
  Item number for feature map data parallel in the x dimension (that is the w dimension).

- featuremap_splits_item_y [**optional**]

  Item number for feature map data parallel in the y dimension (that is the h dimension).

- featuremap_splits_overlap_rate [**optional**]

  Maximum allowed overlap rate for feature map data parallel. The value should be in the range [0, 100). The default value is 50.



Note: The tiling feature in OPT is deprecated and replaced by auto tiling feature in GBuilder of commercial version `AIPUBuilder`. However, you can always manually customize experimental tiling strategy as you want with OPT.

OPT Usage: Miscellaneous

- ❑ Dump tensors: You can enable dumping through the following fields in the cfg file
  `dump`: Whether to enable dumping all tensors and other data.
  `dump_dir`: Dumping all data files will save to this path when `dump=true`

PS: The input data used to dump is taken from the calibration dataset if provided otherwise from the validation dataset, and the batch_size is decided as `batch_size=min(len(xdataset), max(1, batch_size_in_input_IR))`,
and the input data is also used to calculate the quantization similarity score stored in the generated JSON file

- ❑ Deal with model which has no batch dim:
  - Trigger the `without_batch_dim=True` field in the cfg file
  - Arrange datasets with the original shape as the input IR described
  - Customize a dataset plugin and manually expand a batch dim before feeding to torch dataloader (you can refer to the pre-defined `NumpyMultiInputWithoutBatchDimDataset` plugin)

- ❑ Deal with particular start nodes:
  - `set_qinvariant_for_start_nodes`: Point out those start nodes (input layers or constant layers) which should always have 'scale=1.0, zero_point=0' when being quantized (for example, tensors represent indexes). Must be a list of unsigned integers (corresponding to layer_id in the input IR) like '0' or '0,2' or `0,1,2`. Usually there is no need to be set, because the inferred results based on the input IR are correct.

OPT Usage: Miscellaneous

❑ Adapt to lib’ s spec or hardware limitation

- `cast_dtypes_for_lib`: Whether to cast dtypes of OPs to adapt to lib's dtypes' spec (may cause model accuracy loss due to corresponding spec's restriction). 'False' means no. 'True' means yes to all OPs. A list of valid operator type names (case insensitive, corresponding to layer_type in the float IR), for example, Abs,reshape,tile means yes to all the specified OPs
- `bias_effective_bits`: The effective high bits for bias data which really takes part in computation (lower bits will be set to 0), due to hardware restriction (it is 16bit in most Zhouyi platforms). You may also need to set lower `bias_bits` when accuracy loss occurs
- `remain_shift`: In AIFF, computation layer such as Convolution, FullyConnected and Matmul uses ahead shift feature to avoid overflow in following ITP process. The middle result comming from MAC has 48bit width, while ITP only takes 32bit data as input. So AIFF will shift M bits after MAC and before ITP, then clip the data of 32bit width. In ITP, the shifter will right shift remaining bits (i.e, origin shift bits - M bits) to get final result. To simulate this process (usually only necessary under 16bit quantization), config this field (within range of 0-32).
- `unify_shifts_for_aiff`: Whether to unify shifts for AIFF OPs due to hardware limitations. This often happens under per-channel quantization, and usually has no harm to accuracy when set to true
- `force_shift_positive`: Whether to force requantization parameter 'shift' to be positive (due to hardware limitations, accuracy drop may occurs). This often happens when the model is quantized with a random dataset, set it to true if you just want to evaluate performance and ignore the quantization accuracy
- `force_dtype_int`: Whether to force layer top tensors to be quantized to int types (may cause accuracy drop or be rejected by lib's restriction) instead of being decided automatically

安谋科技 arm CHINA

OPT Usage: Miscellaneous

❑ Enable float operators

- `trigger_float_op` : For activating layer lib's float implementation, and the configurable options is: `disable`, `float16_preferred`, `bfloat16_preferred`, `float32_preferred`, `float16_act_int_wht`, `float32_act_int_wht`, `bfloat16_act_int_wht`. Where 'float16_preferred', 'bfloat16_preferred' and 'float32_preferred' means corresponding float type will be selected preferentially if existed (the calibration dataset may still needed, as probably not all of the model's operators do have float type layer lib implementation and quantization will be applied under such circumstances). Option ended with _int_wht means weight-only quantization will be applied (activations will be kept as float). If you want to ignore the implementation limitations of libs' dtype spec and force the specified float type to be used, you can append a '!' behind (so 'disable' is no need to append a '!'), e.g. 'float16_preferred!', 'float16_act_int_wht!'. The default value is 'disable'.

❑ Reduce running time

- Set `eval_original_model = False` when the accuracy metric result of the original IR is already known
- Set larger `dataloader_workers` to enable multi-thread data loading (<u>better to set to 0 when debugging</u>)
- Set `save_statistic_info =True` to store a copy of statistic info file and use ` statistic_file =/path/to/file` to load (and surely you can also edit this file to customize your own calibration strategy). It is <u>not recommended if your cfg or JSON configurations are not fixed</u>

OPT Usage: Miscellaneous

❑ Sacrifice accuracy for performance

- `resize_degrade_to_nearest`: Whether to degrade the resize method to the nearest neighbor to speed up resizing
- `unify_scales_for_concat`: Whether to unify scales of each concat's branch when possible, and you may also need to adjust the `unify_scales_for_concat_threshold` and `unify_scales_for_concat_max_depth` fields to deal with complicated network structures
- `with_winograd`: Whether to enable the Winograd algorithm when possible. The quantization accuracy difference between original weights and weights applied with the Winograd algorithm is usually negligible

❑ Debug problematical operators or locate quantization sensitive subgraphs

- `fake_quant_operators_for_debug`: Run float forward in related layers in assigned operators when calling quanted forward for debug usage. Must be a list of valid operator type names (case insensitive, corresponding to layer_type in the input IR), for example, Abs,reshape,tile
- `fake_quant_scopes_for_debug`: Run float forward in related layers in assigned scopes when calling quanted forward for debug usage. Must be a list of unsigned integer (corresponding to layer_id in the input IR) tuples like '[(i,j)]' or '[(i,j),(n,m),...,(k,l)]

PS: These fields are disabled by default. **Remember to disable them after debugging** by removing them from your cfg file.

❑ Choose running devices:

- Automatically use CUDA to accelerate calculation when it is available. You can set the environment variable CUDA_VISIBLE_DEVICES to ‘-1’ or ‘’ to disable this feature.

安谋科技 arm CHINA

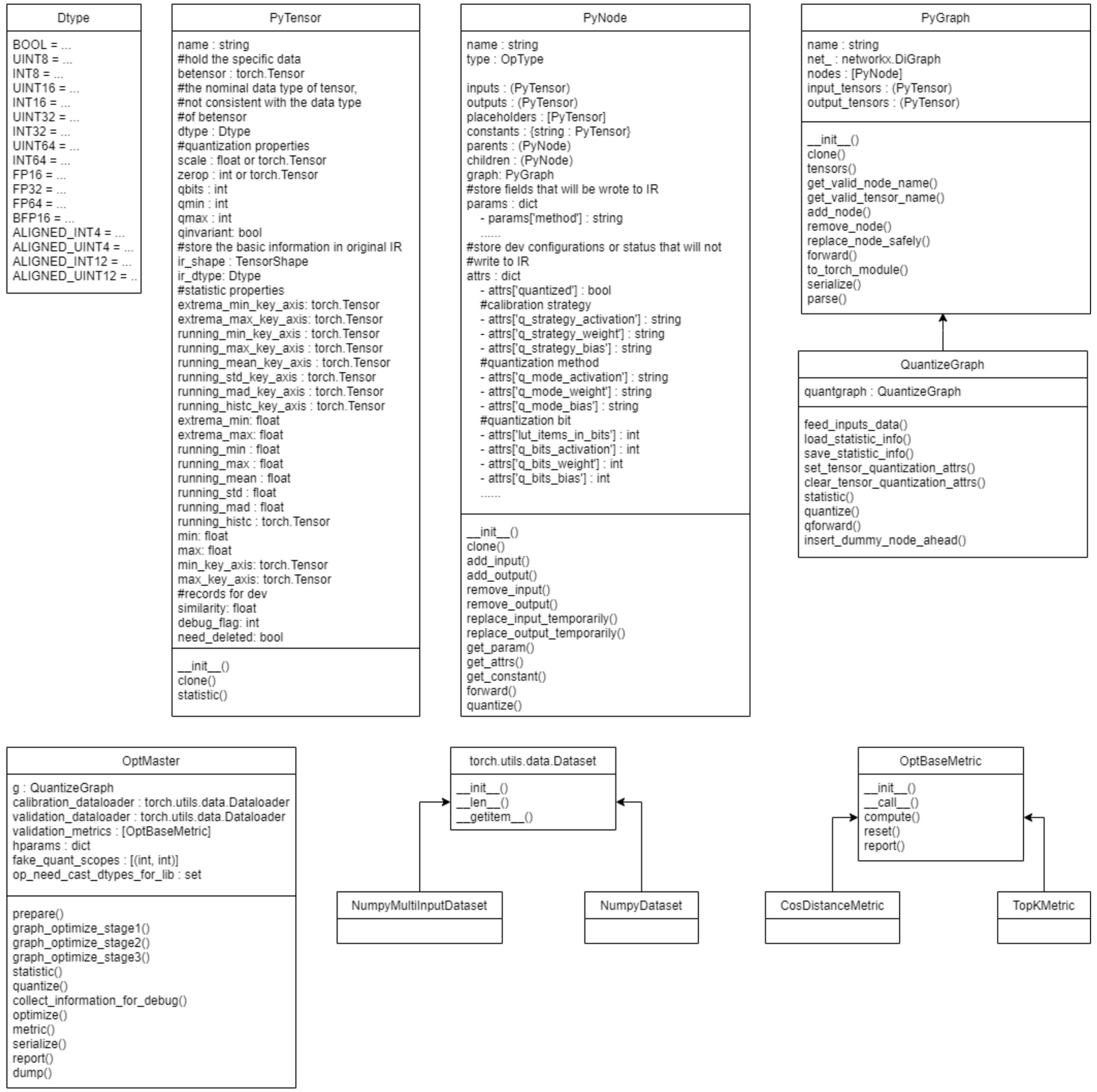
OPT Usage: Miscellaneous

❑ Per-layer configuration through cfg file

- You can also use 'global_value & <[(layer_id1,layer_id2),(layer_id3,layer_id4), ...]:local_value1> < [operator_type1, operator_type2, ...]:local_value2> ...' formart (where 'global_value' is the default value to configure each layer, and 'lobal_value' is for overwriting the default value on specific layers you assigned, 'layer_id' stands for layer_id in input IR and '(layer_id1, layer_id2)' specify the layers which will be applied (layer_id1 <= layer_id <= layer_id2), 'operator_type' stands for valid operator type names that specify the operators which will be applied) for per-layer configuration directly in cfg file. Currently supported fields are: calibration_strategy_for_activation, calibration_strategy_for_weight, quantize_method_for_weight, quantize_method_for_activation, weight_bits, activation_bits, bias_bits, lut_items_in_bits, force_dtype_int, force_shift_positive, running_statistic_momentum, histc_bins, resize_degrade_to_nearest, with_winograd, trigger_float_op, trim_infinity_before_statistic, ...

OPT Development: Core Data Structures

- Dtype defines basic data types that may occur in the Compass IR
- PyTensor is the tensor wrapper class in OPT. It is actually stored and calculated through torch.Tensor
- PyNode represents the layer concept in NN models. Connections between layers are represented as shared tensors (the PyTensor instances stored in each layer's inputs and outputs)
- PyGraph represents the model's computation graph. Its network structure is maintained internally through a networkx.DiGraph instance. QuantizeGraph is inherited from PyGraph, and held by OptMaster
- OptMaster controls the whole process flow, and instances model's corresponding dataset plugin and metric plugins according to the configuration file



OPT Development: Writing Plugins

- Naming:
  - aipubt_opt_op_ for the optimizer operator plugin.
  - aipubt_opt_dataset_ for the dataset plugin of the optimizer.
  - aipubt_opt_metric_ for the metric plugin of the optimizer.

- Search Paths:
  - Paths defined in the environment variable AIPUPLUGIN_PATH, for example：
    export AIPUPLUGIN_PATH=/home/user/aipubuilder_plugins/:$AIPUPLUGIN_PATH
  - The current working directory, which is ./plugin/.

OPT Development: Write Plugins

❑ Dataset plugin

- Dataset plugin inherits from the torch.utils.data.Dataset class. It should be implemented with three methods `__init__`, `__len__` and `__getitem__`
- When a dataset plugin is instanced, two parameters will be passed to it: data_file and label_file. The two parameters in the cfg file may be specific paths of the data file and label file, also they can be files that record the real paths of specific data or label files. The parsing procedure is totally controlled by the developer
- If the graph has multiple input tensors, the data part sample[0][i] should be consistent with input_tensors[i] in IR. If the graph has multiple output tensors, the label part sample[1][j] should be consistent with output_tensors[j] in IR

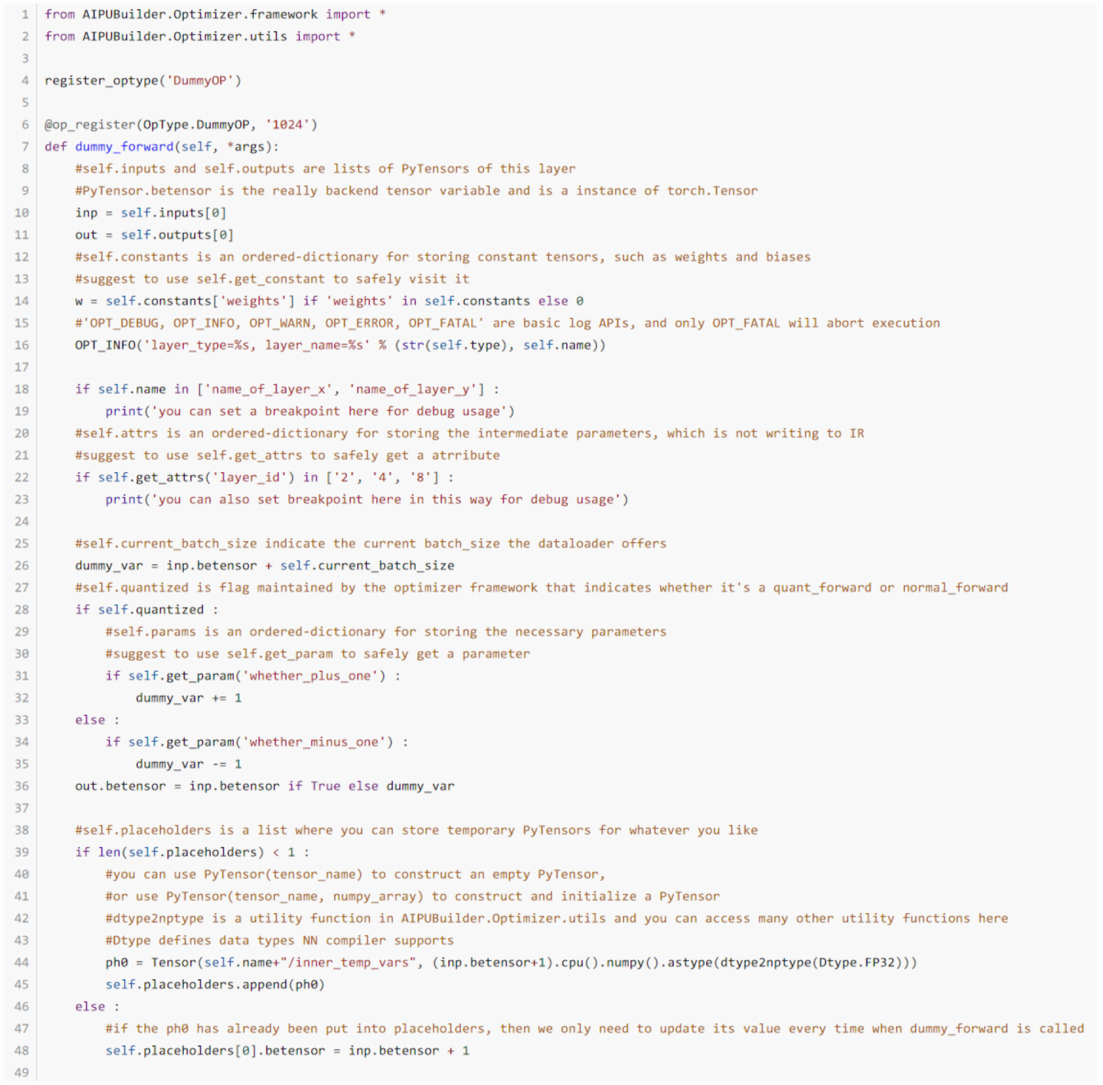
```python
from AIPUBuilder.Optimizer.framework import *
from AIPUBuilder.Optimizer.logger import *
from torch.utils.data import Dataset
import numpy as np

@register_plugin(PluginType.Dataset, '1.0')
class NumpyDataset(Dataset):
    #when used as calibration dataset, label_file can be omitted.
    def __init__(self, data_file, label_file=None):
        self.data = None
        self.label = None
        try:
            self.data = np.load(data_file, mmap_mode='c')
        except Exception as e:
            OPT_FATAL('the data of NumpyDataset plugin should be Numpy.ndarray and allow_pickle=False.')
        if label_file is not None:
            try:
                self.label = np.load(label_file, mmap_mode='c')
            except ValueError:
                self.label = np.load(label_file, allow_pickle=True)
    def __len__(self):
        return len(self.data)
    def __getitem__(self, idx):
        #Assume that all preprocesses have been done before save to npy file.
        #If the graph has single input tensor,
        #the data part sample[0] will be passed to the input node as is,
        #if the graph has multiple input tensors,
        #the data part sample[0][i] should be consistent with input_tensors[i] in IR.
        #If the graph has multiple output tensors,
        #the label part sample[1][i] should be consistent with output_tensors[i] in IR.
        sample = [[self.data[idx]], float("-inf")]
        if self.label is not None:
            sample[1] = self.label[idx]
        return sample
```

安谋科技 arm CHINA

OPT Development: Writing Plugins

❑ Operator plugin

- The operator plugin needs to implement two methods — forward and quantize：
  - \`op_register(OpType, version)\` to register a forward function \`forward_function_name(self, *args)\`
  - \`quant_register(OpType, version)\` to register a quantize function \`quant_function_name(self, *args)\`
- If you want to replace the existing operator's implementation, just set the corresponding operator type OpType.layer_type_name, and set the version greater than 1.0 (the existing inner operator's version is 1.0)
- If you want to implement a new operator type, you need to globally call the register_optype('new_layer_type_name') function to register the new operator type to OpType, and then use the OpType.new_layer_type_name.
- \`self\` is pointed to a PyNode instance (represents a layer in IR)
- The forward simulation is usually not bit-wise aligned with those on the simulator or emulator (for efficiency)

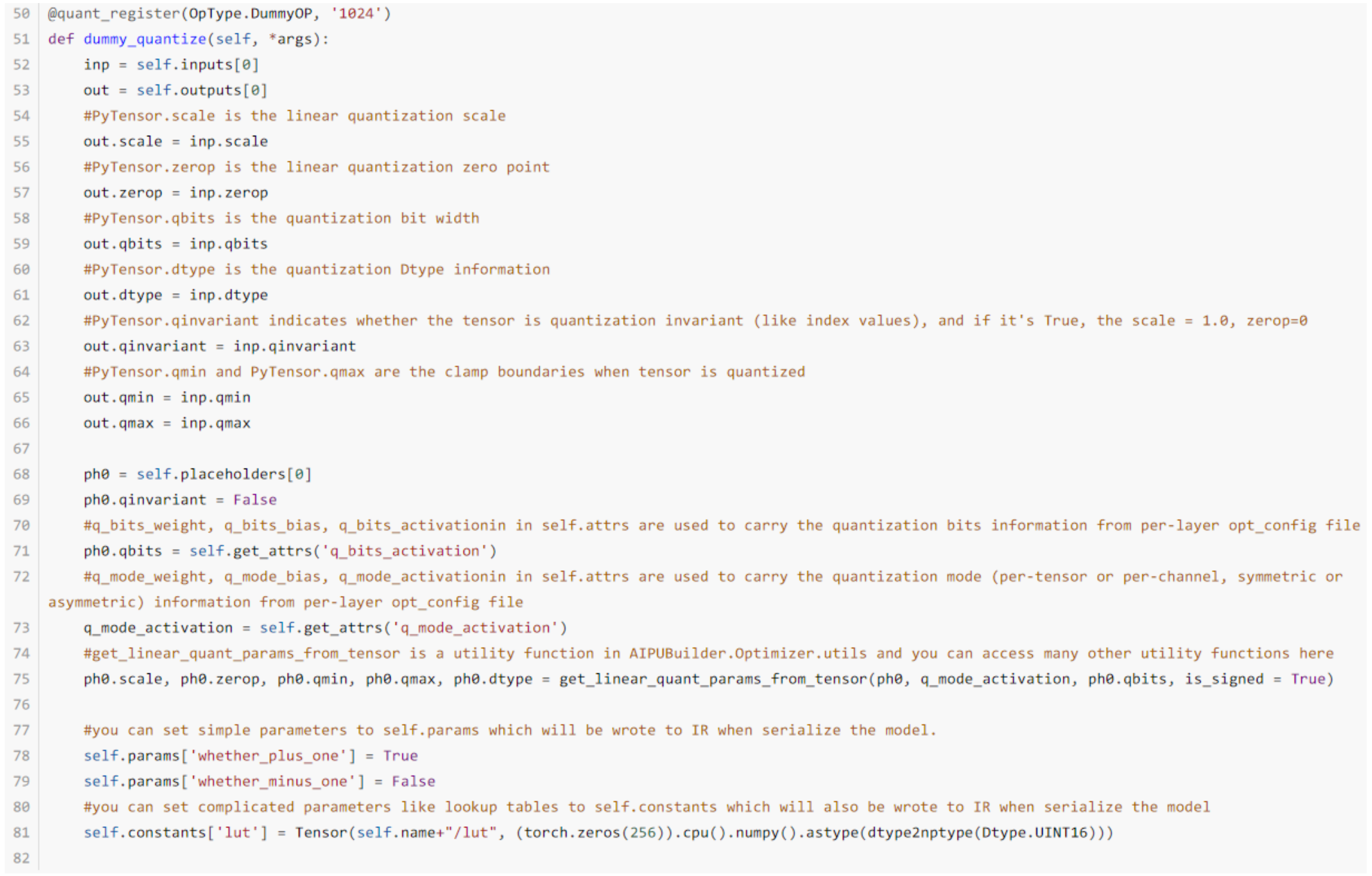


```python
from AIPUBuilder.Optimizer.framework import *
from AIPUBuilder.Optimizer.utils import *

register_optype('DummyOP')

@op_register(OpType.DummyOP, '1024')
def dummy_forward(self, *args):
    #self.inputs and self.outputs are lists of PyTensors of this layer
    #PyTensor.betensor is the really backend tensor variable and is a instance of torch.Tensor
    inp = self.inputs[0]
    out = self.outputs[0]
    #self.constants is an ordered-dictionary for storing constant tensors, such as weights and biases
    #suggest to use self.get_constant to safely visit it
    w = self.constants['weights'] if 'weights' in self.constants else 0
    #'OPT_DEBUG, OPT_INFO, OPT_WARN, OPT_ERROR, OPT_FATAL' are basic log APIs, and only OPT_FATAL will abort execution
    OPT_INFO('layer_type=%s, layer_name=%s' % (str(self.type), self.name))

    if self.name in ['name_of_layer_x', 'name_of_layer_y'] :
        print('you can set a breakpoint here for debug usage')
    #self.attrs is an ordered-dictionary for storing the intermediate parameters, which is not writing to IR
    #suggest to use self.get_attrs to safely get a atrribute
    if self.get_attrs('layer_id') in ['2', '4', '8'] :
        print('you can also set breakpoint here in this way for debug usage')

    #self.current_batch_size indicate the current batch_size the dataloader offers
    dummy_var = inp.betensor + self.current_batch_size
    #self.quantized is flag maintained by the optimizer framework that indicates whether it's a quant_forward or normal_forward
    if self.quantized :
        #self.params is an ordered-dictionary for storing the necessary parameters
        #suggest to use self.get_param to safely get a parameter
        if self.get_param('whether_plus_one') :
            dummy_var += 1
    else :
        if self.get_param('whether_minus_one') :
            dummy_var -= 1
    out.betensor = inp.betensor if True else dummy_var

    #self.placeholders is a list where you can store temporary PyTensors for whatever you like
    if len(self.placeholders) < 1 :
        #you can use PyTensor(tensor_name) to construct an empty PyTensor,
        #or use PyTensor(tensor_name, numpy_array) to construct and initialize a PyTensor
        #dtype2nptype is a utility function in AIPUBuilder.Optimizer.utils and you can access many other utility functions here
        #Dtype defines data types NN compiler supports
        ph0 = Tensor(self.name+"/inner_temp_vars", (inp.betensor+1).cpu().numpy().astype(dtype2nptype(Dtype.FP32)))
        self.placeholders.append(ph0)
    else :
        #if the ph0 has already been put into placeholders, then we only need to update its value every time when dummy_forward is called
        self.placeholders[0].betensor = inp.betensor + 1

```

OPT Development: Writing Plugins

❑ Operator plugin

- The forward function and the quantize function are registered with different APIs, so you can replace each of them or both of them for an existing operator
- OPT will execute a forward pass after parsing float IR to ensure that each operator's forward function always has been called at least once before its quantize function is called (conversely is not guaranteed)
- The scale (and other quantization related parameters) of the OpType.Input operator is calculated through automatic statistic on the calibration dataset by default. In some cases, if the scale (and other quantization related parameters) is known already, you can directly write a higher version plugin of the input operator.



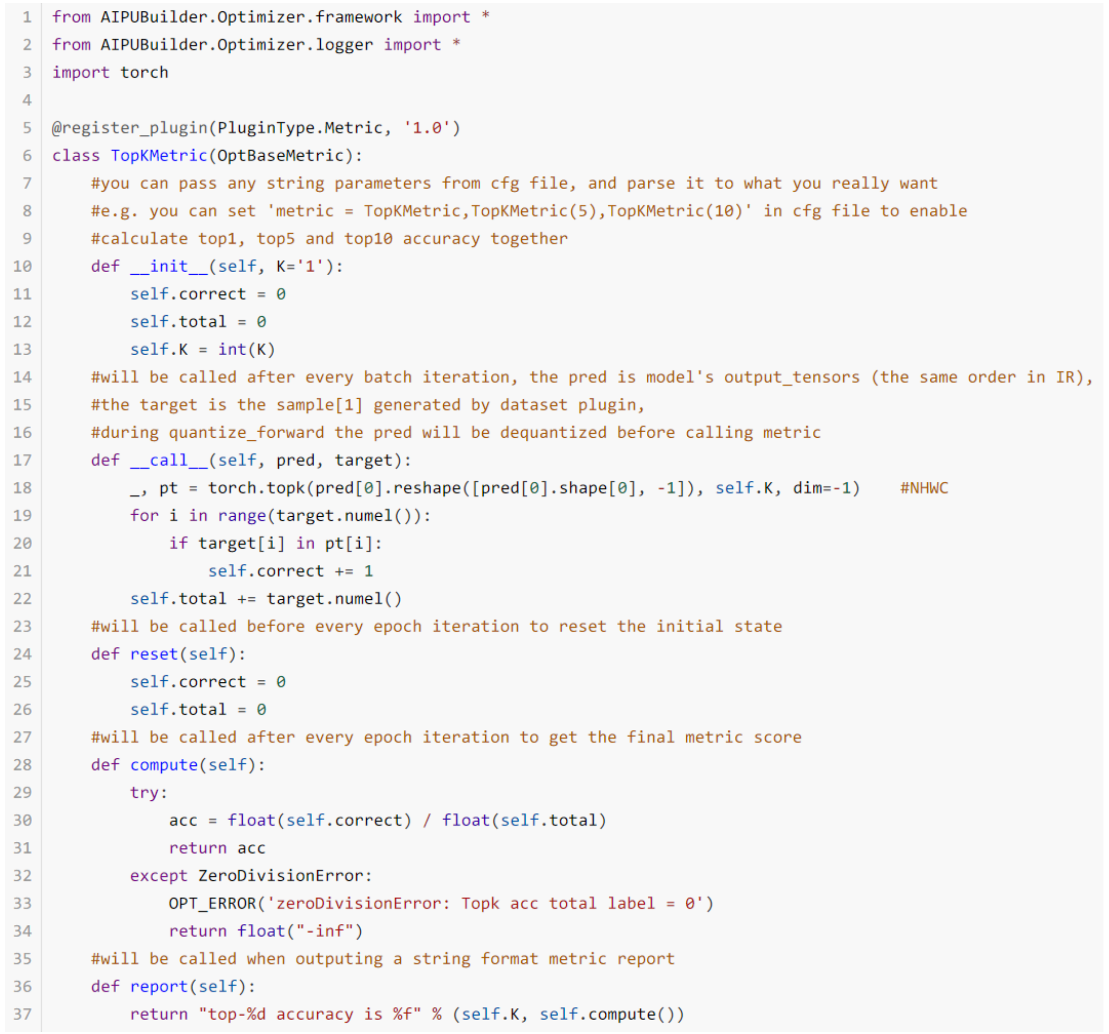
```python
@quant_register(OpType.DummyOP, '1024')
def dummy_quantize(self, *args):
    inp = self.inputs[0]
    out = self.outputs[0]
    #PyTensor.scale is the linear quantization scale
    out.scale = inp.scale
    #PyTensor.zerop is the linear quantization zero point
    out.zerop = inp.zerop
    #PyTensor.qbits is the quantization bit width
    out.qbits = inp.qbits
    #PyTensor.dtype is the quantization Dtype information
    out.dtype = inp.dtype
    #PyTensor.qinvariant indicates whether the tensor is quantization invariant (like index values), and if it's True, the scale = 1.0, zerop=0
    out.qinvariant = inp.qinvariant
    #PyTensor.qmin and PyTensor.qmax are the clamp boundaries when tensor is quantized
    out.qmin = inp.qmin
    out.qmax = inp.qmax

    ph0 = self.placeholders[0]
    ph0.qinvariant = False
    #q_bits_weight, q_bits_bias, q_bits_activationin in self.attrs are used to carry the quantization bits information from per-layer opt_config file
    ph0.qbits = self.get_attrs('q_bits_activation')
    #q_mode_weight, q_mode_bias, q_mode_activationin in self.attrs are used to carry the quantization mode (per-tensor or per-channel, symmetric or asymmetric) information from per-layer opt_config file
    q_mode_activation = self.get_attrs('q_mode_activation')
    #get_linear_quant_params_from_tensor is a utility function in AIPUBuilder.Optimizer.utils and you can access many other utility functions here
    ph0.scale, ph0.zerop, ph0.qmin, ph0.qmax, ph0.dtype = get_linear_quant_params_from_tensor(ph0, q_mode_activation, ph0.qbits, is_signed = True)

    #you can set simple parameters to self.params which will be wrote to IR when serialize the model.
    self.params['whether_plus_one'] = True
    self.params['whether_minus_one'] = False
    #you can set complicated parameters like lookup tables to self.constants which will also be wrote to IR when serialize the model
    self.constants['lut'] = Tensor(self.name+"/lut", (torch.zeros(256)).cpu().numpy().astype(dtype2nptype(Dtype.UINT16)))

```

安谋科技 arm CHINA

OPT Development: Writing Plugins

❑ Metric plugin

- Metric plugin inherits from the OptBaseMetric class. It should be implemented with these interfaces: `__init__`, `__call__`, `reset`, `compute`, `report`
- Metric plugin can get initial parameters from the cfg file, but it only supports string type, so you need to parse these parameters from the string in the `__init__` function
- If the model has multiple outputs, the order of pred outputs is aligned with the order of ‘output_tensors’ in the header of the input IR
- The pred and target passed by OPT will be automatically dequantized in advance when running a quantized inference forward



```python
from AIPUBuilder.Optimizer.framework import *
from AIPUBuilder.Optimizer.logger import *
import torch

@register_plugin(PluginType.Metric, '1.0')
class TopKMetric(OptBaseMetric):
    #you can pass any string parameters from cfg file, and parse it to what you really want
    #e.g. you can set 'metric = TopKMetric,TopKMetric(5),TopKMetric(10)' in cfg file to enable
    #calculate top1, top5 and top10 accuracy together
    def __init__(self, K='1'):
        self.correct = 0
        self.total = 0
        self.K = int(K)
    #will be called after every batch iteration, the pred is model's output_tensors (the same order in IR),
    #the target is the sample[1] generated by dataset plugin,
    #during quantize_forward the pred will be dequantized before calling metric
    def __call__(self, pred, target):
        _, pt = torch.topk(pred[0].reshape([pred[0].shape[0], -1]), self.K, dim=-1)    #NHWC
        for i in range(target.numel()):
            if target[i] in pt[i]:
                self.correct += 1
        self.total += target.numel()
    #will be called before every epoch iteration to reset the initial state
    def reset(self):
        self.correct = 0
        self.total = 0
    #will be called after every epoch iteration to get the final metric score
    def compute(self):
        try:
            acc = float(self.correct) / float(self.total)
            return acc
        except ZeroDivisionError:
            OPT_ERROR('zeroDivisionError: Topk acc total label = 0')
            return float("-inf")
    #will be called when outputing a string format metric report
    def report(self):
        return "top-%d accuracy is %f" % (self.K, self.compute())
```

OPT Development: Adding Featur

❑ Code Style

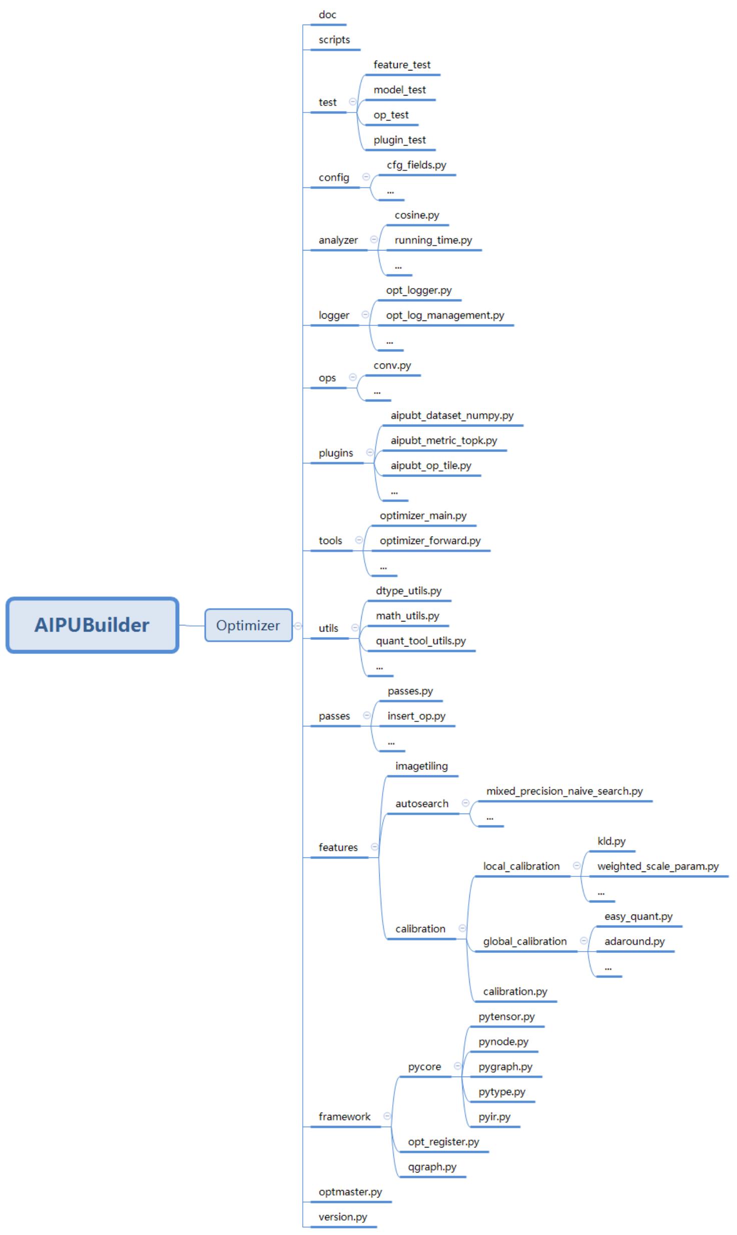
OPT uses autopep8 for code format checking. AIPUBuilder/Optimizer/scripts offers related git hooks. Ensure that you have installed autopep8 and it is available in your environment

❑ Local Test

Before pushing your codes, it is strongly recommended running enough local test cases to verify them. The most effective way is to sample cases from Zhouyi Model Zoo (https://github.com/Arm-China/Model_zoo) and have them all passed

❑ Directory Structure

Please refer to the current directory structure to arrange your codes or files

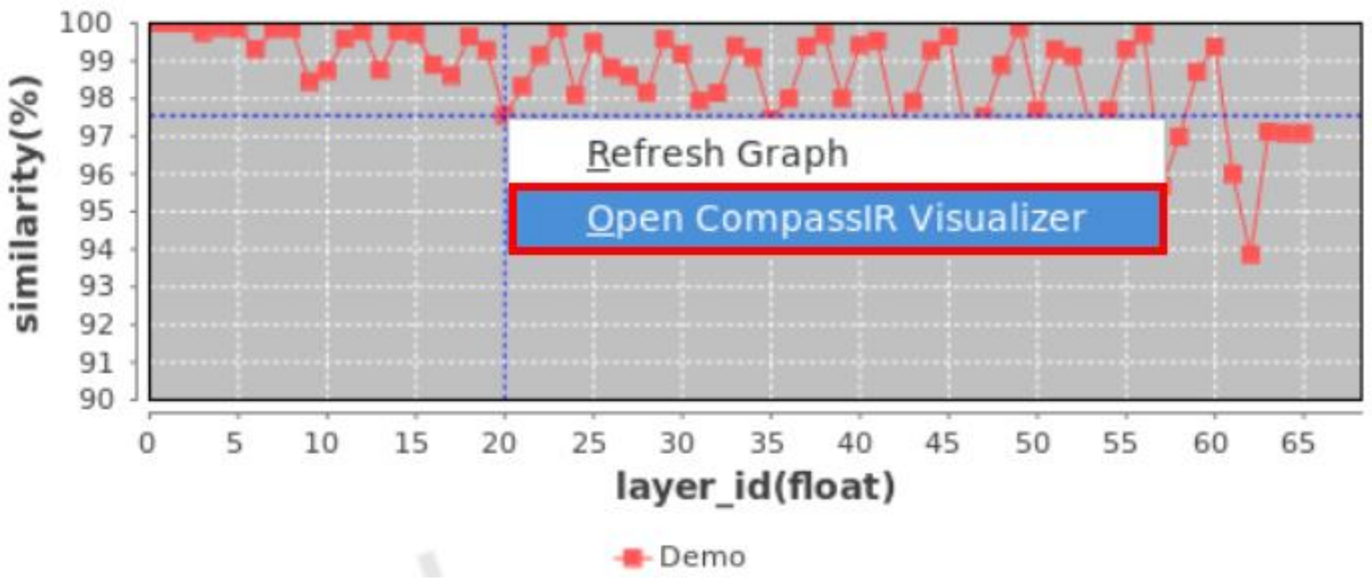


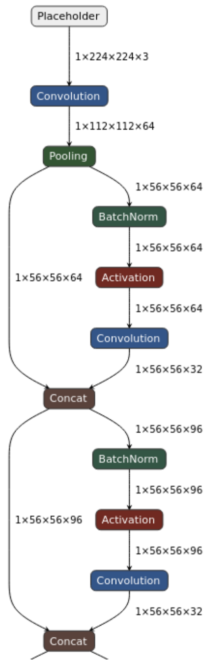
安谋科技 arm CHINA

More About Model Quantization

❑ Measure Quantization Accuracy Drop

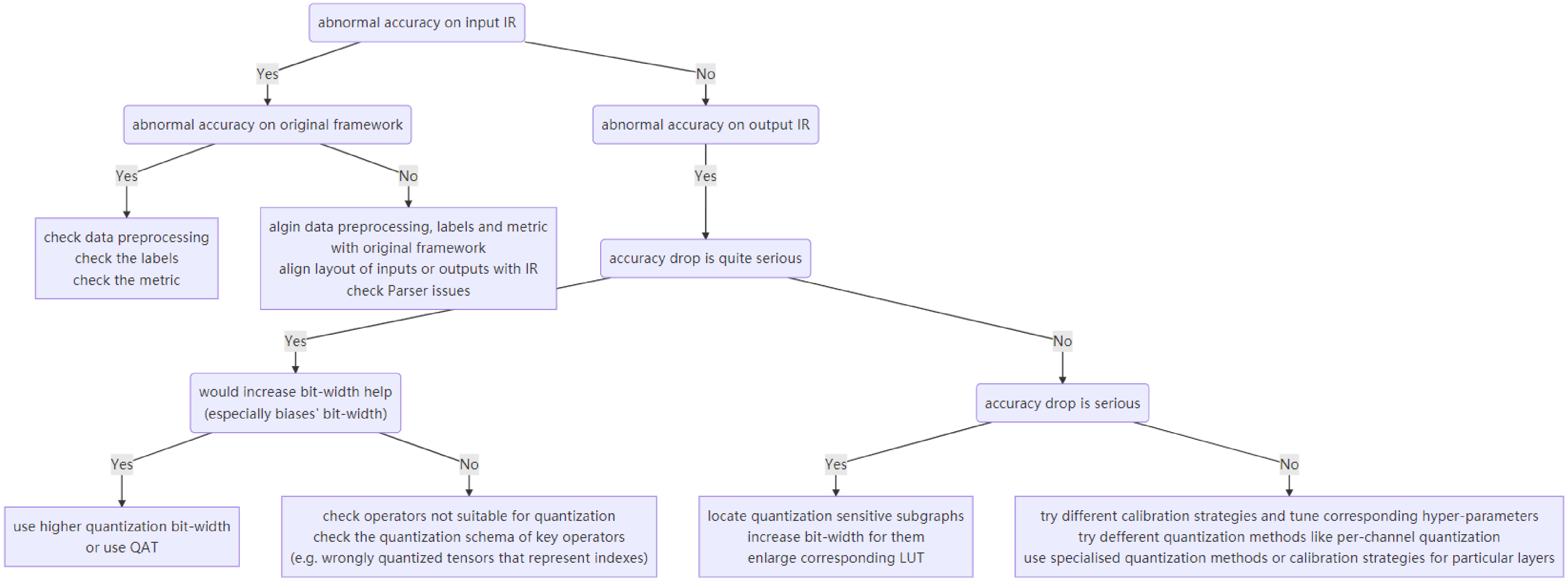
- ✓ The best way is to implement the corresponding metric plugin for golden criterion
- ✓ OPT will log the per-layer cosine similarity information in the output JSON file. You can analyze it (with our CompassStudio GUI)
  - ❖ Cosine distance is an approximate measure, and the value itself has limited meaning
  - ❖ Empirically you should pay more attention to the subgraphs that have sudden and sequential lower values
  - ❖ Input data is taken from the calibration dataset if provided otherwise from the validation dataset, and the batch_size is decided as `batch_size=min(len(xdataset), max(1, batch_size_in_input_IR))`





More About Model Quantization

❑ Locate Potential Problems

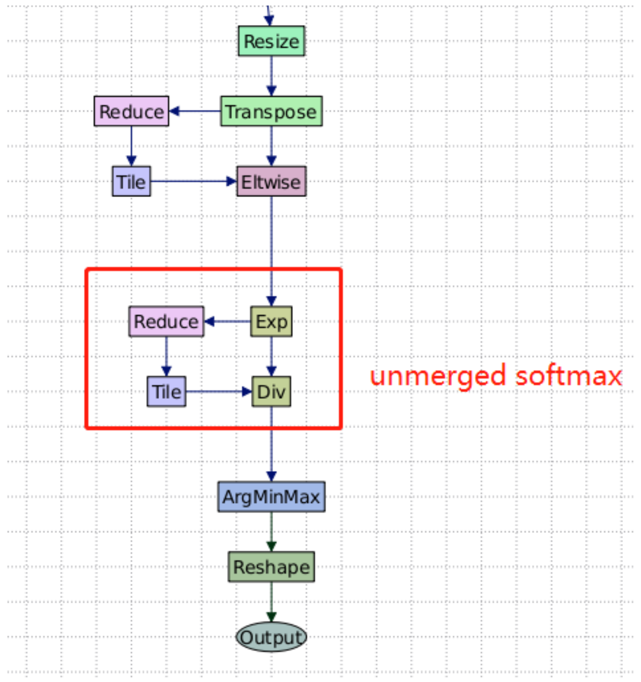

abnormal accuracy on input IR
Yes
No
abnormal accuracy on original framework
abnormal accuracy on output IR
Yes
No
Yes
check data preprocessing
check the labels
check the metric
algin data preprocessing, labels and metric
with original framework
align layout of inputs or outputs with IR
check Parser issues
accuracy drop is quite serious
Yes
No
would increase bit-width help
(especially biases' bit-width)
accuracy drop is serious
Yes
No
Yes
No
use higher quantization bit-width
or use QAT
check operators not suitable for quantization
check the quantization schema of key operators
(e.g. wrongly quantized tensors that represent indexes)
locate quantization sensitive subgraphs
increase bit-width for them
enlarge corresponding LUT
try different calibration strategies and tune corresponding hyper-parameters
try defferent quantization methods like per-channel quantization
use specialised quantization methods or calibration strategies for particular layers

More About Model Quantization: Other Tips

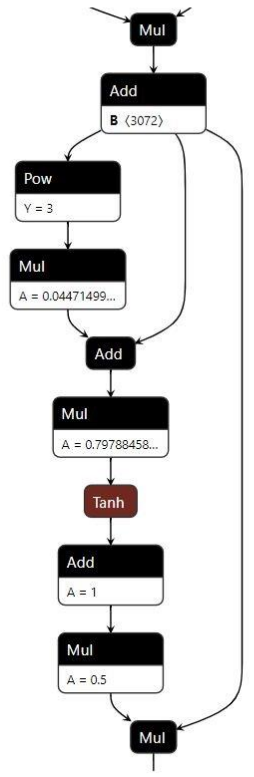
- The most valuable thing is to get the correct dataset and metric ready first
- Visualization and/or auralization of input/output samples (generated by the original framework, input IR and output IR) and comparing them are often helpful
- Suggested tuning paradigm for urgent tasks: get satisfying accuracy first, then lowering costs
  - Use 16bit quantization (also remember the use of ` lut_items_in_bits `)
  - Use mixed precision (maybe with using ` mixed_precision_auto_search`, or manually setting with hints from using ` fake_quant_operators_for_debug `)
  - Try 8bit per-channel quantization for weights and/or asymmetric quantization for activations
  - Additionally tuning with a global calibration strategy (`adaround` is often cost-effective)
- Pay attention to infinite or equivalent extreme values in model (usually cause abnormal scales)
  - you can exclude these values from statistic through configuring the `trim_infinity_before_statistic` field

More About Model Quantization: Other Tips

- ❖ Should always know better about the model itself before debugging its quantization issues
- ❖ Pay attention to unmerged sequential nonlinear one-variable functions (especially `exp`, `log` or `pow`) or norm like operators, and try to merge them with the *Compass Parser*

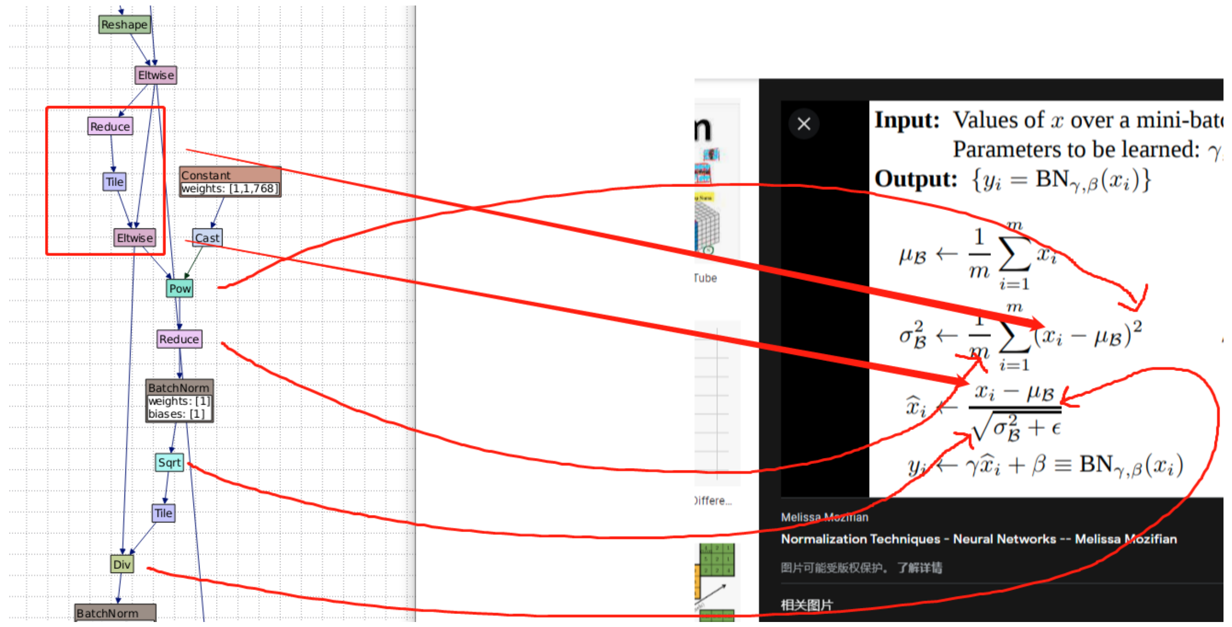


unmerged softmax



$$x\Phi(x) \approx \frac{1}{2}x\left[1+\tanh\left(\sqrt{\frac{2}{\pi}}\left(x+0.044715x^3\right)\right)\right]$$

unmerged GELU



Input: Values of x over a mini-bat
Parameters to be learned: γ

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$$

Melissa Mozifian

Normalization Techniques - Neural Networks -- Melissa Mozifian

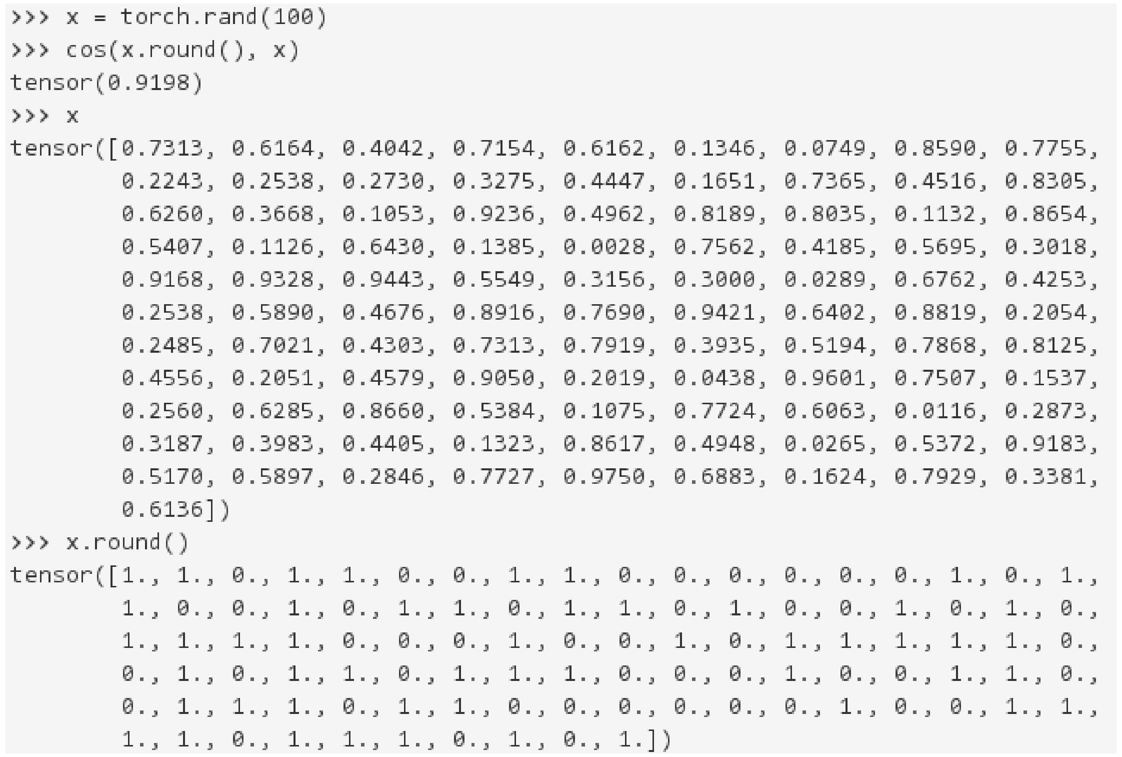
图片可能受版权保护。 了解详情

相关图片

安谋科技
arm CHINA

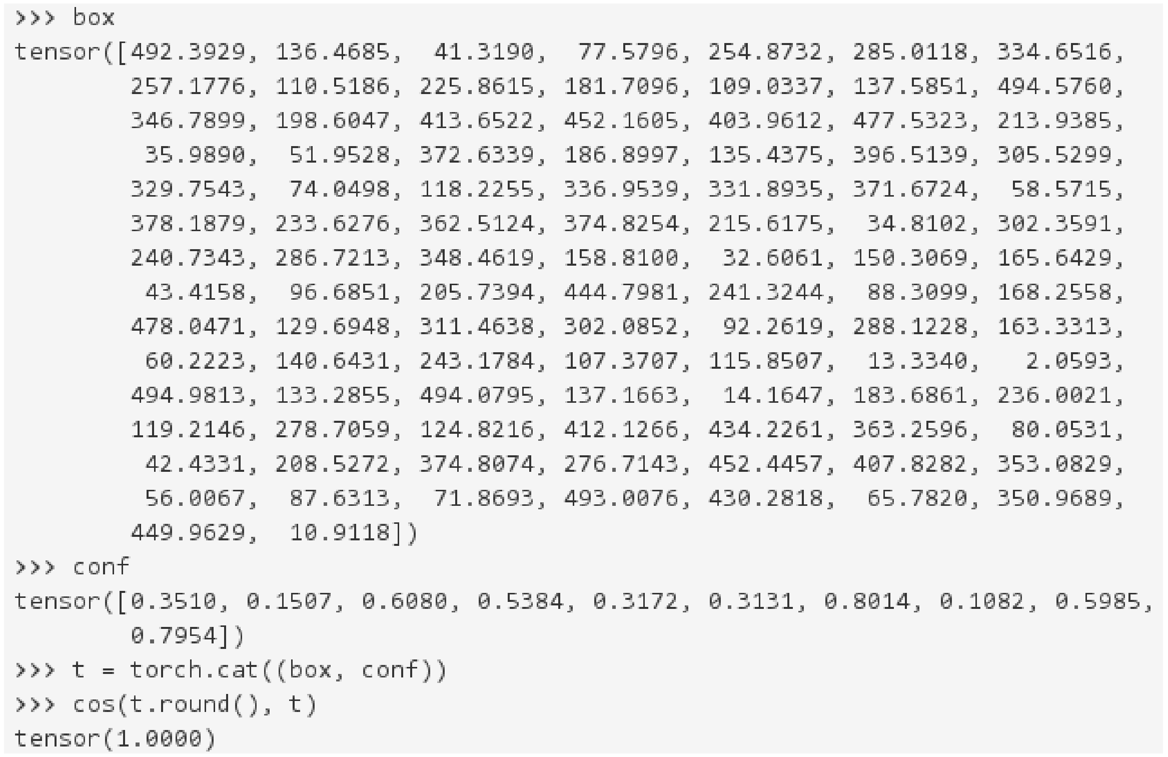
More About Model Quantization: Other Tips

❖ Cosine similarity is not always suitable and reliable



```
>>> x = torch.rand(100)
>>> cos(x.round(), x)
tensor(0.9198)
>>> x
tensor([0.7313, 0.6164, 0.4042, 0.7154, 0.6162, 0.1346, 0.0749, 0.8590, 0.7755,
        0.2243, 0.2538, 0.2730, 0.3275, 0.4447, 0.1651, 0.7365, 0.4516, 0.8305,
        0.6260, 0.3668, 0.1053, 0.9236, 0.4962, 0.8189, 0.8035, 0.1132, 0.8654,
        0.5407, 0.1126, 0.6430, 0.1385, 0.0028, 0.7562, 0.4185, 0.5695, 0.3018,
        0.9168, 0.9328, 0.9443, 0.5549, 0.3156, 0.3000, 0.0289, 0.6762, 0.4253,
        0.2538, 0.5890, 0.4676, 0.8916, 0.7690, 0.9421, 0.6402, 0.8819, 0.2054,
        0.2485, 0.7021, 0.4303, 0.7313, 0.7919, 0.3935, 0.5194, 0.7868, 0.8125,
        0.4556, 0.2051, 0.4579, 0.9050, 0.2019, 0.0438, 0.9601, 0.7507, 0.1537,
        0.2560, 0.6285, 0.8660, 0.5384, 0.1075, 0.7724, 0.6063, 0.0116, 0.2873,
        0.3187, 0.3983, 0.4405, 0.1323, 0.8617, 0.4948, 0.0265, 0.5372, 0.9183,
        0.5170, 0.5897, 0.2846, 0.7727, 0.9750, 0.6883, 0.1624, 0.7929, 0.3381,
        0.6136])
>>> x.round()
tensor([1., 1., 0., 1., 1., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 1., 0., 1.,
        1., 0., 0., 1., 0., 1., 1., 0., 1., 1., 0., 1., 0., 0., 1., 0., 1., 0.,
        1., 1., 1., 1., 0., 0., 0., 1., 0., 0., 1., 0., 1., 1., 1., 1., 1., 0.,
        0., 1., 0., 1., 1., 0., 1., 1., 1., 0., 0., 0., 1., 0., 0., 1., 1., 0.,
        0., 1., 1., 1., 0., 1., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 1., 1.,
        1., 1., 0., 1., 1., 1., 0., 1., 0., 1.])
```

Cosine distance is not that close, but the similarity may be good enough



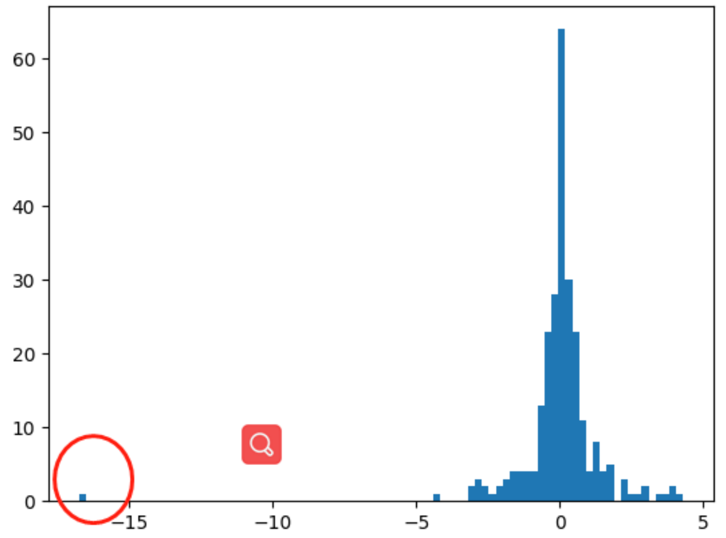
```
>>> box
tensor([492.3929, 136.4685,  41.3190,  77.5796, 254.8732, 285.0118, 334.6516,
        257.1776, 110.5186, 225.8615, 181.7096, 109.0337, 137.5851, 494.5760,
        346.7899, 198.6047, 413.6522, 452.1605, 403.9612, 477.5323, 213.9385,
         35.9890,  51.9528, 372.6339, 186.8997, 135.4375, 396.5139, 305.5299,
        329.7543,  74.0498, 118.2255, 336.9539, 331.8935, 371.6724,  58.5715,
        378.1879, 233.6276, 362.5124, 374.8254, 215.6175,  34.8102, 302.3591,
        240.7343, 286.7213, 348.4619, 158.8100,  32.6061, 150.3069, 165.6429,
         43.4158,  96.6851, 205.7394, 444.7981, 241.3244,  88.3099, 168.2558,
        478.0471, 129.6948, 311.4638, 302.0852,  92.2619, 288.1228, 163.3313,
         60.2223, 140.6431, 243.1784, 107.3707, 115.8507,  13.3340,   2.0593,
        494.9813, 133.2855, 494.0795, 137.1663,  14.1647, 183.6861, 236.0021,
        119.2146, 278.7059, 124.8216, 412.1266, 434.2261, 363.2596,  80.0531,
         42.4331, 208.5272, 374.8074, 276.7143, 452.4457, 407.8282, 353.0829,
         56.0067,  87.6313,  71.8693, 493.0076, 430.2818,  65.7820, 350.9689,
        449.9629,  10.9118])
>>> conf
tensor([0.3510, 0.1507, 0.6080, 0.5384, 0.3172, 0.3131, 0.8014, 0.1082, 0.5985,
        0.7954])
>>> t = torch.cat((box, conf))
>>> cos(t.round(), t)
tensor(1.0000)
```

Cosine distance is perfect, but may hide issues about key components

More About Model Quantization: Other Tips

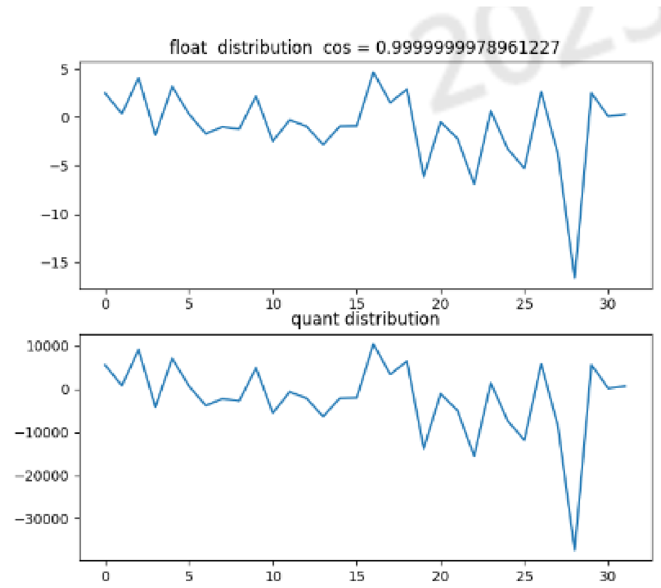
❖ Studying tensor level quantization accuracy drop: float tensor vs quantized/dequantized tensor

histogram

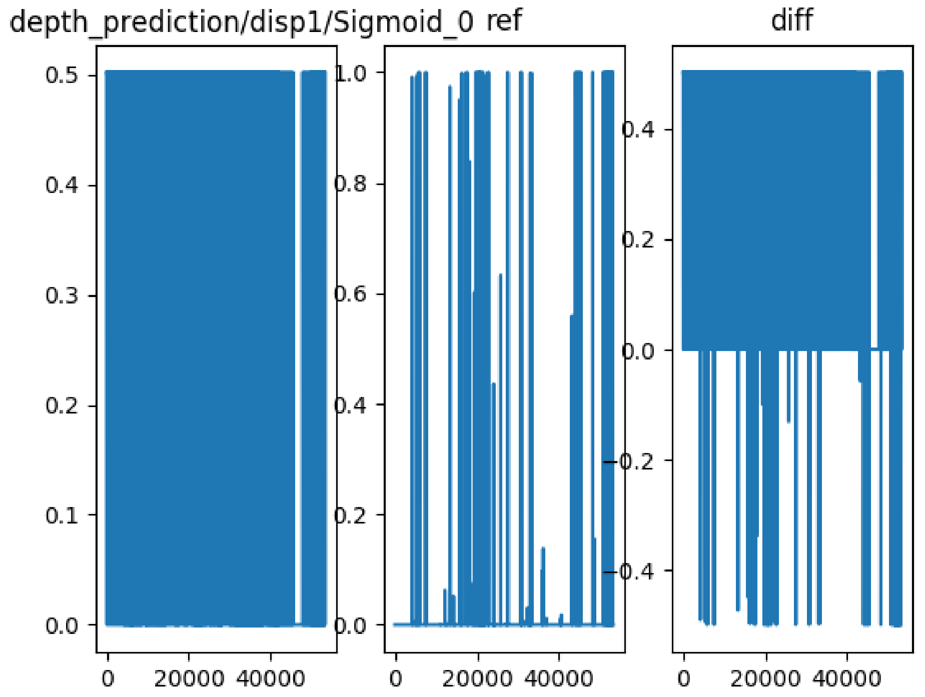


outliers: need to be excluded by calibration

raw value distribution



normal



abnormal: truncation caused by improper scale or bit-width

安谋科技 arm CHINA

Reference

[1] arXiv:2103.13630. A Survey of Quantization Methods for Efficient Neural Network Inference

[2] arXiv:2102.04503. VS-Quant: Per-vector Scaled Quantization for Accurate Low-Precision Neural Network Inference

[3] arXiv:1712.05877. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference

[4] https://on-demand.gputechconf.com/gtc/2017/presentation/s7310-8-bit-inference-with-tensorrt.pdf

[5] arXiv:1810.05723. Post-training 4-bit quantization of convolution networks for rapid-deployment

[6] arXiv:2006.16669. EasyQuant: Post-training Quantization via Scale Optimization

[7] arXiv:2004.10568. Up or Down? Adaptive Rounding for Post-Training Quantization

Q & A

安谋科技
arm CHINA

THANKS

www.armchina.com

安谋科技
arm CHINA