
TPU-MLIR Quick Start

Release 1.20-1-gcc08e8b8b

SOPHGO

Jul 03, 2025

Table of contents

1 TPU-MLIR Introduction	4
2 Environment Setup	6
2.1 Basic Environment Configuration	6
2.2 Install tpu_mlir	7
2.3 Install the dependency of tpu_mlir	7
3 Compile the ONNX model	9
3.1 Install tpu_mlir	9
3.2 Prepare working directory	10
3.3 ONNX to MLIR	10
3.4 MLIR to F16 bmodel	12
3.5 MLIR to INT8 bmodel	13
3.5.1 Calibration table generation	13
3.5.2 Compile to INT8 symmetric quantized model	13
3.6 Effect comparison	14
3.7 Model performance test	15
3.7.1 Install the libsophon	15
3.7.2 Check the performance of BModel	15
4 Compile the Torch Model	17
4.1 Install tpu_mlir	17
4.2 Prepare working directory	17
4.3 TORCH to MLIR	18
4.4 MLIR to F16 bmodel	18
4.5 MLIR to INT8 bmodel	19
4.5.1 Calibration table generation	19
4.5.2 Compile to INT8 symmetric quantized model	19
4.6 Effect comparison	19
5 Compile the Caffe model	21
5.1 Install tpu_mlir	21
5.2 Prepare working directory	21
5.3 Caffe to MLIR	22
5.4 MLIR to F32 bmodel	22
5.5 MLIR to INT8 bmodel	23
5.5.1 Calibration table generation	23
5.5.2 Compile to INT8 symmetric quantized model	23

6 Compile the TFLite model	24
6.1 Install tpu_mlir	24
6.2 Prepare working directory	24
6.3 TFLite to MLIR	25
6.4 MLIR to bmodel	25
7 Quantization and optimization	26
7.1 TPU-MLIR Full Int8 Symmetric Quantization	26
7.1.1 run_calibration Process Introduction	27
7.1.2 run_calibration Parameter Introduction	27
7.1.3 The Use of run_calibration Parameters Introduction	30
7.2 Overview of TPU-MLIR Mixed Precision Quantization	33
7.3 pattern-match	33
7.3.1 YOLO Series Automatic Mixed Precision Method	33
7.3.2 Transformer Series Automatic Mixed Precision Method	34
7.4 1. search_qtable	34
7.4.1 Install tpu_mlir	34
7.4.2 Prepare working directory	34
7.4.3 Accuracy test of float anf int8 models	35
7.4.4 To Mix Precision Model	36
7.5 2. run_sensitive_layer	42
7.5.1 Install tpu_mlir	42
7.5.2 Prepare working directory	44
7.5.3 Accuracy test of float anf int8 models	44
7.5.4 To Mix Precision Model	46
7.6 3. fp_forward	50
8 Use Tensor Computing Processor for Preprocessing	53
8.1 Model Deployment Example	54
8.1.1 Deploy to BM168x	54
8.1.2 Deploy to CV18xx	55
9 Use Tensor Computing Processor for Postprocessing	56
9.1 Add detection post_process(YOLOv5s)	56
9.1.1 Prepare working directory	56
9.1.2 ONNX to MLIR	57
9.1.3 MLIR to Bmodel	58
9.1.4 Bmodel Verification	59
9.2 Add segmentation post_processing(YOLOv8s_seg)	59
9.2.1 Prepare working directory	59
9.2.2 ONNX to MLIR	59
9.2.3 MLIR to Bmodel	61
9.2.4 Bmodel Verification	61
10 Compile LLM Model	62
10.1 Overview	62
10.2 Command-Line Arguments	62
10.3 Example Usage	63

11 Appendix.01: Reference for converting model to ONNX format	65
11.1 PyTorch model to ONNX	65
11.1.1 Step 0: Create a working directory	65
11.1.2 Step 1: Build and save the model	66
11.1.3 Step 2: Export ONNX model	66
11.2 TensorFlow model to ONNX	67
11.2.1 Step 0: Create a working directory	67
11.2.2 Step 1: Prepare and convert the model	67
11.3 PaddlePaddle model to ONNX	68
11.3.1 Step 0: Install openssl-1.1.1o	68
11.3.2 Step 1: Create a working directory	68
11.3.3 Step 2: Prepare the model	68
11.3.4 Step 3: Convert the model	69
12 Appendix.02: CV18xx Guidance	70
12.1 Compile yolov5 model	70
12.1.1 Install tpu_mlir	70
12.1.2 Prepare working directory	70
12.1.3 ONNX to MLIR	71
12.1.4 MLIR to BF16 Model	71
12.1.5 MLIR to INT8 Model	72
12.1.6 Result Comparison	72
12.2 Merge cvimodel Files	74
12.2.1 Step 0: generate the cvimodel for batch 1	74
12.2.2 Step 1: generate the cvimodel for batch 2	75
12.2.3 Step 2: merge the cvimodel of batch 1 and batch 2	75
12.2.4 Step 3: use the cvimodel through the runtime interface	75
12.2.5 Overview:	76
12.3 Compile and Run the Runtime Sample	76
12.3.1 1) Run the provided pre-build samples	77
12.3.2 2) Cross-compile samples	79
12.3.3 3) Run samples in docker environment	82
12.4 FAQ	83
12.4.1 Model transformation FAQ	83
12.4.2 Model performance evaluation FAQ	86
12.4.3 Common problems of model deployment	87
12.4.4 Others	88
13 Appendix.03: BM168x Guidance	90
13.1 Merge bmodel Files	90
13.1.1 Step 0: generate the bmodel for batch 1	90
13.1.2 Step 1: generate the bmodel for batch 2	91
13.1.3 Step 2: merge the bmodel of batch 1 and batch 2	92
13.1.4 Overview:	92
14 Appendix.04: Model-zoo test	93
14.1 Test Notification	93

14.2	Configure the system environment	93
14.3	Get the model-zoo model	94
14.4	Prepare the runtime environment	94
14.5	Configure SoC device	95
14.6	Prepare dataset	96
14.6.1	ImageNet	96
14.6.2	COCO (optional)	96
14.6.3	Vid4 (optional)	96
14.7	Prepare the toolchain compilation environment	97
14.8	Model performance and accuracy testing process	97
14.8.1	Compile the model	97
14.8.2	Performance test	98
14.8.3	Precision test	99
15	Appendix.05: TPU Profile Tool Guidance	101
15.1	Compile Bmodel	101
15.2	Generate Raw Profile Data	102
15.3	Visualize Profile Data	102
16	Appendix.06: TDB Guidance	104
16.1	Preparatory work	104
16.2	Start TDB	105
16.3	TDB command summary	105
16.4	TDB usage process	106
16.5	TDB function description	107
16.5.1	next feature	107
16.5.2	breakpoint feature	107
16.5.3	info feature	108
16.5.4	print feature	109
16.5.5	watchpoint feature	110
16.5.6	py feature	111
16.6	BModel Disassembler	112
16.7	BModel Checker	112
17	Appendix.07: Supported Operations	118
17.1	List of operators currently supported by TPU-MLIR	118

TABLE OF CONTENTS



SOPHON

Legal Notices

Copyright © SOPHGO 2025. All rights reserved.

No part or all of the contents of this document may be copied, reproduced or transmitted in any form by any organization or individual without the written permission of the Company.

Attention

All products, services or features, etc. you purchased is subject to SOPHGO's business contracts and terms. All or part of the products, services or features described in this document may not be covered by your purchase or use. Unless otherwise agreed in the contract, SOPHGO makes no representations or warranties (including express and implied) regarding the contents of this document. The contents of this document may be updated from time to time due to product version upgrades or other reasons. Unless otherwise agreed, this document is intended as a guide only. All statements, information and recommendations in this document do not constitute any warranty, express or implied.

Technical Support

Address

Building 1, Zhongguancun Integrated Circuit Design Park (ICPARK), No. 9
Fenghao East Road, Haidian District, Beijing

Postcode

100094

URL

<https://www.sophgo.com/>

Email

sales@sophgo.com

Tel

010-57590723

TABLE OF CONTENTS

Release Record

TABLE OF CONTENTS

Version	Release date	Explanation
v1.19.0	2025.05.30	Support AWQ and GPTQ models; Deconv3D F16, F32 bugfix
v1.18.0	2025.05.01	YOLO series adds automatic mixed precision setting; Added SmoothQuant option for run_calibration; New one-click compilation script for LLM
v1.17.0	2025.04.03	Significant improvement in LLM model compilation speed; TPULang supports PPL operator integration; Fixed random error issue with Trilu bf16 on Mars3
v1.16.0	2025.03.03	TPULang ROI_Extractor support; Einsum supports abcde,abfce->abcdfe pattern; LLMC adds Vila model support
v1.15.0	2025.02.05	Added LLMC quantization support; Address boundary check in codegen; Fixed several comparison issues
v1.14.0	2025.01.02	Added post-processing fusion for yolov8/v11; Support for Conv3D stride > 15; Improved FAttention accuracy
v1.13.0	2024.12.02	Streamlined Release package; Performance optimization for MaxPoolWithMask training operator; Added support for large RoPE operators
v1.12.0	2024.11.06	tpuv7-runtime cmodel integration; BM1690 multi-core LayerGroup optimization; Support for PPL backend operator development
v1.11.0	2024.09.27	Added soc mode for BM1688 tdb; bmodel supports fine-grained merging; Fixed several performance degradation issues
v1.10.0	2024.08.15	Added yolov10 support; New quantization tuning section; Optimized tpu-perf log output
v1.9.0	2024.07.16	BM1690 added 40 model regression tests; New quantization algorithms: octav, aciq_guas and aciq_laplace
v1.8.0	2024.05.30	BM1690 supports multi-core MatMul operator; TPULang supports input/output order specification; tpuperf removes patchelf dependency
v1.7.0	2024.05.15	CV186X dual-core changed to single-core; BM1690 testing process aligned with BM1684X; Support for gemma/llama/qwen models
v1.6.0	2024.02.23	Added Pypi release format; Support for user-defined Global operators; Added CV186X processor platform support
v1.5.0	2023.11.03	More Global Layer support for multi-core parallel processing
v1.4.0	2023.09.27	System dependencies upgraded to Ubuntu22.04; Added BM1684 Winograd support
v1.3.0	2023.07.27	Copyright © 2023 SORICO. All rights reserved. This document contains confidential information. It is the property of SORICO and is intended for internal use only. It may not be reproduced, distributed, or disclosed without the express written consent of SORICO.
v1.2.0	2023.06.14	Adjusted mixed quantization examples
v1.1.0	2023.05.26	Added post-processing using intelligent deep learning techniques for improved accuracy

CHAPTER 1

TPU-MLIR Introduction

TPU-MLIR is the Tensor Computing Processor compiler project for Deep Learning processors. This project provides a complete toolchain, which can convert pre-trained neural networks under different frameworks into binary files bmodel that can be efficiently run on tensor computing processors. The code has been open-sourced to github: <https://github.com/sophgo/tpu-mlir> .

The overall architecture of TPU-MLIR is shown in the figure ([TPU-MLIR overall architecture](#)).

The current directly supported frameworks are pytorch, onnx, tflite and caffe. Models from other frameworks need to be converted to onnx models. The method of converting models from other frameworks to onnx can be found on the onnx official website: <https://github.com/onnx/tutorials>.

To convert a model, firstly you need to execute it in the specified docker. With the required environment, conversion work can be done in two steps, converting the original model to mlir file by `model_transform` and converting the mlir file to bmodel/cvimodel by `model_deploy`. To obtain an INT8 model, you need to call `run_calibration` to generate a quantization table and pass it to `model_deploy`. This article mainly introduces the process of this model conversion.

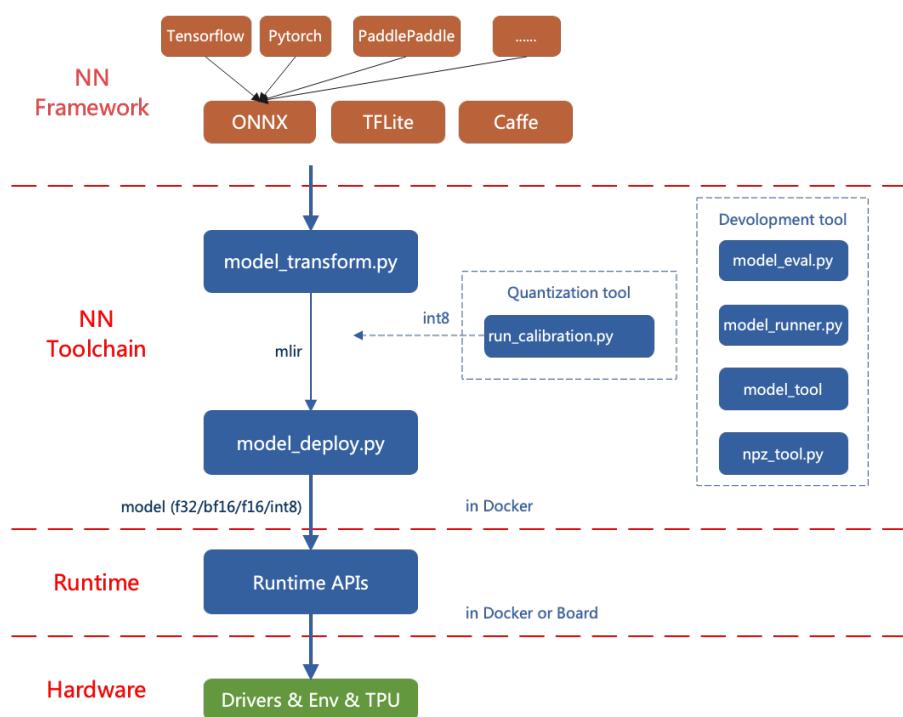


Fig. 1.1: TPU-MLIR overall architecture

CHAPTER 2

Environment Setup

First check whether the current system environment meets ubuntu 22.04 and python 3.10. If not, please proceed to the next section Basic Environment Configuration; if satisfied, jump directly to Install tpu_mlir.

2.1 Basic Environment Configuration

If you do not meet the above system environment, you need to use Docker, download the required image file from DockerHub https://hub.docker.com/r/sophgo/tpuc_dev, or use the following command to pull the image directly:

```
1 $ docker pull sophgo/tpuc_dev:v3.4
```

If the pulling fails, you can download the required image file from the official website development materials <https://developer.sophgo.com/site/index/material/86/all.html>, or use the following command to download and load the image:

```
1 $ wget https://sophon-assets.sophon.cn/sophon-prod-s3/drive/25/04/15/16/tpuc_dev_v3.4.tar.gz  
2 $ docker load -i tpuc_dev_v3.4.tar.gz
```

If you are using docker for the first time, you can execute the following commands to install and configure it (only for the first time):

```
1 $ sudo apt install docker.io  
2 $ sudo systemctl start docker  
3 $ sudo systemctl enable docker  
4 $ sudo groupadd docker
```

(continues on next page)

(continued from previous page)

```
5 $ sudo usermod -aG docker $USER  
6 $ newgrp docker
```

If you download the image file, make sure the image file is in the current directory, and then create a container in the current directory as follows:

```
# use --privileged to get root permission, if you don't need root permission, please remove this F  
→ parameter  
$ docker run --privileged --name myname -v $PWD:/workspace -it sophgo/tpuc_dev:v3.4
```

where `myname` is the name of the container, which can be customized; `$PWD` is the current directory, synchronized with the container's `/workspace` directory.

Subsequent chapters assume that the user is already in the `/workspace` directory inside docker.

2.2 Install tpu_mlir

Currently supported 2 methods to install, which are online and offline installation.

Online installation

Download and install directly from pypi, the latest version will be installed by default:

```
$ pip install tpu_mlir
```

Offline installation

Download `tpu_mlir-*py3-none-any.whl` from [Assets](#) on Github, then install with pip:

```
$ pip install tpu_mlir-*py3-none-any.whl
```

2.3 Install the dependency of tpu_mlir

`tpu_mlir` requires different dependencies when processing models of different frameworks, Both the online and offline installation methods require additional dependencies to be installed.

Online installation

For model files generated by `onnx` or `torch` when online installation, use the following command to install additional dependency environments:

```
# install onnx dependency  
$ pip install tpu_mlir[onnx]  
# install torch dependency  
$ pip install tpu_mlir[torch]
```

There are 5 config currently supported:

onnx, torch, tensorflow, caffe, paddle

You can install multiple dependency config in one command, or use all to install all dependencies:

```
# install onnx, torch, caffe dependency at the same time
$ pip install tpu_mlir[onnx,torch,caffe]
# install all dependency
$ pip install tpu_mlir[all]
```

Offline installation

Similarly, the offline installation method allows you to install additional dependencies using the following command:

```
# install onnx dependency
$ pip install tpu_mlir-* -py3-none-any.whl[onnx]
# install all dependency
$ pip install tpu_mlir-* -py3-none-any.whl[all]
```

CHAPTER 3

Compile the ONNX model

This chapter takes `yolov5s.onnx` as an example to introduce how to compile and transfer an onnx model to run on the BM1684X platform.

The model is from the official website of yolov5: <https://github.com/ultralytics/yolov5/releases/download/v6.0/yolov5s.onnx>

This chapter requires the `tpu_mlir` python package.

platform	file name	info
cv183x/cv182x/cv181x/cv180x	xxx.cvimodel	please refer to the CV18xx Guidance
Other	xxx.bmodel	please refer to the following

3.1 Install `tpu_mlir`

Go to the Docker container and execute the following command to install `tpu_mlir`:

```
$ pip install tpu_mlir[onnx]  
# or  
$ pip install tpu_mlir-*-py3-none-any.whl[onnx]
```

3.2 Prepare working directory

Please download `tpu-mlir-resource.tar` from `Assets` and unzip it, and rename the folder to `tpu_mlir_resource` :

```
$ tar -xvf tpu-mlir-resource.tar  
$ mv regression/ tpu-mlir-resource/
```

Create a `model_yolov5s` directory, and put both model files and image files into the `model_yolov5s` directory.

The operation is as follows:

```
1 $ mkdir model_yolov5s && cd model_yolov5s  
2 $ wget https://github.com/ultralytics/yolov5/releases/download/v6.0/yolov5s.onnx  
3 $ cp -rf tpu_mlir_resource/dataset/COCO2017 .  
4 $ cp -rf tpu_mlir_resource/image .  
5 $ mkdir workspace && cd workspace
```

3.3 ONNX to MLIR

If the input is image, we need to know the preprocessing of the model before transferring it. If the model uses preprocessed npz files as input, no preprocessing needs to be considered. The preprocessing process is formulated as follows (x represents the input):

$$y = (x - \text{mean}) \times \text{scale}$$

The image of the official yolov5 is rgb. Each value will be multiplied by 1/255, respectively corresponding to 0.0,0.0,0.0 and 0.0039216,0.0039216,0.0039216 when it is converted into mean and scale.

The model conversion command is as follows:

```
$ model_transform \  
  --model_name yolov5s \  
  --model_def ./yolov5s.onnx \  
  --input_shapes [[1,3,640,640]] \  
  --mean 0.0,0.0,0.0 \  
  --scale 0.0039216,0.0039216,0.0039216 \  
  --keep_aspect_ratio \  
  --pixel_format rgb \  
  --output_names 350,498,646 \  
  --test_input ./image/dog.jpg \  
  --test_result yolov5s_top_outputs.npz \  
  --mlir yolov5s.mlir
```

The main parameters of `model_transform` are described as follows (for a complete introduction, please refer to the user interface chapter of the TPU-MLIR Technical Reference Manual):

Table 3.1: Function of model_transform parameters

Name	Required?	Explanation
model_name	Y	Model name
model_def	Y	Model definition file (e.g., .onnx , .tflite or .prototxt files)
input_shapes	N	Shape of the inputs, such as [[1,3,640,640]] (a two-dimensional array), which can support multiple inputs
input_types	N	Type of the inputs, such int32; separate by ',' for multi inputs; float32 as default
resize_dims	N	The size of the original image to be adjusted to. If not specified, it will be resized to the input size of the model
keep_aspect_ratio	N	When the size of test_input is different from input_shapes, whether to keep the aspect ratio when resizing, the default is false; when set, the insufficient part will be padded with 0
mean	N	The mean of each channel of the image. The default is 0.0,0.0,0.0
scale	N	The scale of each channel of the image. The default is 1.0,1.0,1.0
pixel_format	N	Image type, can be rgb, bgr, gray or rgbd. The default is bgr
channel_format	N	Channel type, can be nhwc or nchw for image input, otherwise it is none. The default is nchw
output_names	N	The names of the output. Use the output of the model if not specified, otherwise use the specified names as the output
test_input	N	The input file for validation, which can be an jpg, npy or npz file. No validation will be carried out if it is not specified
test_result	N	Output file to save verification result with suffix .npz
excepts	N	Names of network layers that need to be excluded from validation. Separated by comma
mlir	Y	Specify the output mlir file name and path, with the suffix .mlir

After converting to an mlir file, a \${model_name}_in_f32.npz file will be generated, which is the input file for the subsequent models.

3.4 MLIR to F16 bmodel

To convert the mlir file to the f16 bmodel, we need to run:

```
$ model_deploy \
  --mlir yolov5s.mlir \
  --quantize F16 \
  --processor bm1684x \
  --test_input yolov5s_in_f32.npz \
  --test_reference yolov5s_top_outputs.npz \
  --tolerance 0.99,0.99 \
  --model yolov5s_1684x_f16.bmodel
```

The main parameters of `model_deploy` are as follows (for a complete introduction, please refer to the user interface chapter of the TPU-MLIR Technical Reference Manual):

Table 3.2: Function of `model_deploy` parameters

Name	Required?	Explanation
mlir	Y	Mlir file
quantize	Y	Quantization type (e.g., F32/F16/BF16/INT8), the quantization types supported by different processors are shown in the table below.
processor	Y	The platform that the model will use. Support bm1690/bm1688/bm1684x/bm1684/cv186x/cv183x/cv182x/cv181x/cv180x
calibration_table	N	The calibration table path. Required when it is INT8/F8E4M3 quantization
tolerance	N	Tolerance for the minimum similarity between MLIR quantized and MLIR fp32 inference results
test_input	N	The input file for validation, which can be an jpg, npy or npz file. No validation will be carried out if it is not specified
test_reference	N	Reference data for validating mlir tolerance (in npz for- mat). It is the result of each operator
compare_all	N	Compare all tensors, if set.
excepts	N	Names of network layers that need to be excluded from validation. Separated by comma
op_divide	N	cv183x/cv182x/cv181x/cv180x only, Try to split the larger op into multiple smaller op to achieve the pur- pose of ion memory saving, suitable for a few specific models
model	Y	Name of output model file (including path)
num_core	N	When the target is selected as bm1688, it is used to select the number of tpu cores for parallel computing, and the default setting is 1 tpu core
skip_validation	N	Skip bmodel correctness verification to boost deploy- ment efficiency; bmodel verification is on by default

Table 3.3: Quantization types supported by different processors

Processor	Supported quantize
BM1688	F32/F16/BF16/INT8/INT4
BM1684X	F32/F16/BF16/INT8
BM1684	F32/INT8
CV186X	F32/F16/BF16/INT8/INT4
CV183X/CV182X/CV181X/CV180X	BF16/INT8
BM1690	F32/F16/BF16/INT8/F8E4M3/F8E5M2

After compilation, a file named `yolov5s_1684x_f16.bmodel` is generated.

3.5 MLIR to INT8 bmodel

3.5.1 Calibration table generation

Before converting to the INT8 model, you need to run calibration to get the calibration table. The number of input data is about 100 to 1000 according to the situation.

Then use the calibration table to generate a symmetric or asymmetric bmodel. It is generally not recommended to use the asymmetric one if the symmetric one already meets the requirements, because the performance of the asymmetric model will be slightly worse than the symmetric model.

Here is an example of the existing 100 images from COCO2017 to perform calibration:

```
$ run_calibration yolov5s.mlir \
  --dataset ..//COCO2017 \
  --input_num 100 \
  -o yolov5s_cali_table
```

After running the command above, a file named `yolov5s_cali_table` will be generated, which is used as the input file for subsequent compilation of the INT8 model.

3.5.2 Compile to INT8 symmetric quantized model

Execute the following command to convert to the INT8 symmetric quantized model:

```
$ model_deploy \
  --mlir yolov5s.mlir \
  --quantize INT8 \
  --calibration_table yolov5s_cali_table \
  --processor bm1684x \
  --test_input yolov5s_in_f32.npz \
```

(continues on next page)

(continued from previous page)

```
--test_reference yolov5s_top_outputs.npz \
--tolerance 0.85,0.45 \
--model yolov5s_1684x_int8_sym.bmodel
```

After compilation, a file named `yolov5s_1684x_int8_sym.bmodel` is generated.

3.6 Effect comparison

In `tpu_mlir` package, there are `yolov5` use cases written in python, using the `detect_yolov5` command to detect objects in images.

This command corresponds to the source code path `{package/path/to/tpu_mlir}/python/samples/detect_yolov5.py`.

It can be learned how the model is used by reading the code. Firstly, preprocess to get the model's input, then do inference to get the output, and finally do post-processing.

Use the following codes to validate the inference results of `onnx/f16/int8` respectively.

The `onnx` model is run as follows to get `dog_onnx.jpg`:

```
$ detect_yolov5 \
--input ..../image/dog.jpg \
--model ..../yolov5s.onnx \
--output dog_onnx.jpg
```

The `f16` bmodel is run as follows to get `dog_f16.jpg` :

```
$ detect_yolov5 \
--input ..../image/dog.jpg \
--model yolov5s_1684x_f16.bmodel \
--output dog_f16.jpg
```

The `int8` symmetric bmodel is run as follows to get `dog_int8_sym.jpg`:

```
$ detect_yolov5 \
--input ..../image/dog.jpg \
--model yolov5s_1684x_int8_sym.bmodel \
--output dog_int8_sym.jpg
```

The result images are compared as shown in the figure ([Comparison of TPU-MLIR for YOLOv5s' compilation effect](#)).

Due to different operating environments, the final performance will be somewhat different from [Fig. 3.1](#).

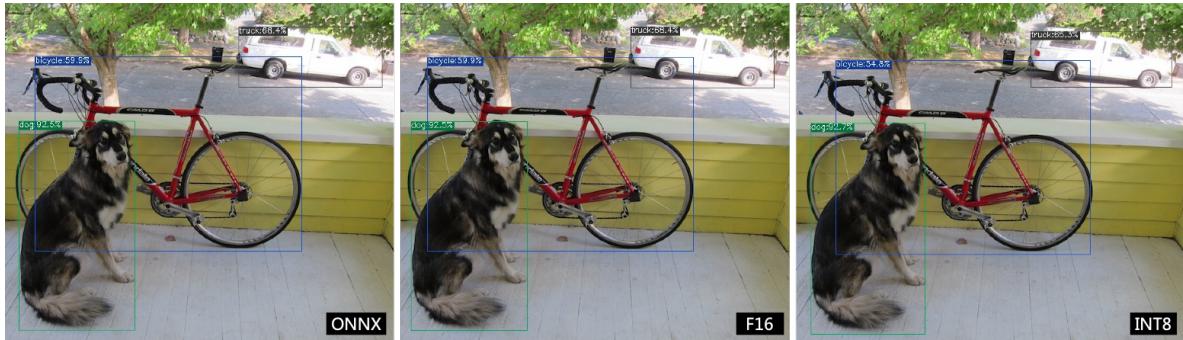


Fig. 3.1: Comparison of TPU-MLIR for YOLOv5s' compilation effect

3.7 Model performance test

The following operations need to be performed outside of Docker,

3.7.1 Install the libsophon

Please refer to the libsophon manual to install libsophon.

3.7.2 Check the performance of BModel

After installing libsophon, you can use `bmrt_test` to test the accuracy and performance of the bmodel. You can choose a suitable model by estimating the maximum fps of the model based on the output of `bmrt_test`.

```
# Test the bmodel compiled above
# --bmodel parameter followed by bmodel file,
$ cd path/to/model_yolov5s/workspace
$ bmrt_test --bmodel yolov5s_1684x_f16.bmodel
$ bmrt_test --bmodel yolov5s_1684x_int8_sym.bmodel
```

Take the output of the last command as an example (the log is partially truncated here):

```
1 [BMRT][load_bmodel:983] INFO:pre net num: 0, load net num: 1
2 [BMRT][show_net_info:1358] INFO: #####
3 [BMRT][show_net_info:1359] INFO: NetName: yolov5s, Index=0
4 [BMRT][show_net_info:1361] INFO: ---- stage 0 ----
5 [BMRT][show_net_info:1369] INFO: Input 0) 'images' shape=[ 1 3 640 640 ] dtype=FLOAT32
6 [BMRT][show_net_info:1378] INFO: Output 0) '350_Transpose_f32' shape=[ 1 3 80 80 85 ] ...
7 [BMRT][show_net_info:1378] INFO: Output 1) '498_Transpose_f32' shape=[ 1 3 40 40 85 ] ...
8 [BMRT][show_net_info:1378] INFO: Output 2) '646_Transpose_f32' shape=[ 1 3 20 20 85 ] ...
9 [BMRT][show_net_info:1381] INFO: #####
10 [BMRT][bmrt_test:770] INFO:==> running network #0, name: yolov5s, loop: 0
11 [BMRT][bmrt_test:834] INFO:reading input #0, bytesize=4915200
```

(continues on next page)

(continued from previous page)

```
12 [BMRT][print_array:702] INFO: --> input_data: < 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
13 [BMRT][bmrt_test:982] INFO:reading output #0, bytesize=6528000
14 [BMRT][print_array:702] INFO: --> output ref_data: < 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
15 [BMRT][bmrt_test:982] INFO:reading output #1, bytesize=1632000
16 [BMRT][print_array:702] INFO: --> output ref_data: < 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
17 [BMRT][bmrt_test:982] INFO:reading output #2, bytesize=408000
18 [BMRT][print_array:702] INFO: --> output ref_data: < 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
19 [BMRT][bmrt_test:1014] INFO:net[yolov5s] stage[0], launch total time is 4122 us (npu 4009 us, F
→normal 113 us)
20 [BMRT][bmrt_test:1017] INFO:+++ The network[yolov5s] stage[0] output_data +++
21 [BMRT][print_array:702] INFO:output data #0 shape: [1 3 80 80 85 ] < 0.301003 ...
22 [BMRT][print_array:702] INFO:output data #1 shape: [1 3 40 40 85 ] < 0 0.228689 ...
23 [BMRT][print_array:702] INFO:output data #2 shape: [1 3 20 20 85 ] < 1.00135 ...
24 [BMRT][bmrt_test:1058] INFO:load input time(s): 0.008914
25 [BMRT][bmrt_test:1059] INFO:calculate time(s): 0.004132
26 [BMRT][bmrt_test:1060] INFO:get output time(s): 0.012603
27 [BMRT][bmrt_test:1061] INFO:compare time(s): 0.006514
```

The following information can be learned from the output above:

1. Lines 05-08: the input and output information of bmodel
2. Line 19: running time on the Tensor Computing Processor, of which the processor takes 4009us and the non-accelerated part takes 113us. The time of the latter mainly refers to the waiting time of calling at HOST
3. Line 24: the time to load data into the NPU's DDR
4. Line 25: the total time of Line 19
5. Line 26: the output data retrieval time

CHAPTER 4

Compile the Torch Model

This chapter takes `yolov5s.pt` as an example to introduce how to compile and transfer an pytorch model to run on the BM1684X platform.

This chapter requires the `tpu_mlir` python package.

4.1 Install tpu_mlir

Go to the Docker container and execute the following command to install `tpu_mlir`:

```
$ pip install tpu_mlir[torch]
# or
$ pip install tpu_mlir-*~py3-none-any.whl[torch]
```

4.2 Prepare working directory

Please download `tpu-mlir-resource.tar` from [Assets](#) and unzip it, and rename the folder to `tpu_mlir_resource` :

```
$ tar -xvf tpu-mlir-resource.tar
$ mv regression/ tpu-mlir-resource/
```

Create a `model_yolov5s_pt` directory, and put both model files and image files into the `model_yolov5s_pt` directory.

The operation is as follows:

```
1 $ mkdir model_yolov5s_pt && cd model_yolov5s_pt
2 $ wget -O yolov5s.pt "https://github.com/sophgo/tpu-mlir/raw/master/regression/model/yolov5s.
3   ↪pt"
4 $ cp -rf tpu_mlir_resource/dataset/COCO2017 .
5 $ cp -rf tpu_mlir_resource/image .
5 $ mkdir workspace && cd workspace
```

4.3 TORCH to MLIR

The model in this example has a RGB input with mean and scale of 0.0,0.0,0.0 and 0.0039216, 0.0039216,0.0039216 respectively.

The model conversion command:

```
$ model_transform \
  --model_name yolov5s_pt \
  --model_def ./yolov5s.pt \
  --input_shapes [[1,3,640,640]] \
  --mean 0.0,0.0,0.0 \
  --scale 0.0039216,0.0039216,0.0039216 \
  --keep_aspect_ratio \
  --pixel_format rgb \
  --test_input ./image/dog.jpg \
  --test_result yolov5s_pt_top_outputs.npz \
  --mlir yolov5s_pt.mlir
```

After converting to mlir file, a \${model_name}_in_f32.npz file will be generated, which is the input file of the model. It is worth noting that we only support static models, and the model needs to call `torch.jit.trace()` to generate a static model before compilation.

4.4 MLIR to F16 bmodel

Convert the mlir file to the bmodel of f16, the operation method is as follows:

```
$ model_deploy \
  --mlir yolov5s_pt.mlir \
  --quantize F16 \
  --processor bm1684x \
  --test_input yolov5s_pt_in_f32.npz \
  --test_reference yolov5s_pt_top_outputs.npz \
  --tolerance 0.99,0.99 \
  --model yolov5s_pt_1684x_f16.bmodel
```

After compilation, a file named `yolov5s_pt_1684x_f16.bmodel` will be generated.

4.5 MLIR to INT8 bmodel

4.5.1 Calibration table generation

Before converting to the INT8 model, you need to run calibration to get the calibration table. Here is an example of the existing 100 images from COCO2017 to perform calibration:

```
$ run_calibration yolov5s_pt.mlir \
--dataset ./COCO2017 \
--input_num 100 \
-o yolov5s_pt_cali_table
```

After running the command above, a file named `yolov5s_pt_cali_table` will be generated, which is used as the input file for subsequent compilation of the INT8 model.

4.5.2 Compile to INT8 symmetric quantized model

Execute the following command to convert to the INT8 symmetric quantized model:

```
$ model_deploy \
--mlir yolov5s_pt.mlir \
--quantize INT8 \
--calibration_table yolov5s_pt_cali_table \
--processor bm1684x \
--test_input yolov5s_pt_in_f32.npz \
--test_reference yolov5s_pt_top_outputs.npz \
--tolerance 0.85,0.45 \
--model yolov5s_pt_1684x_int8_sym.bmodel
```

After compilation, a file named `yolov5s_pt_1684x_int8_sym.bmodel` will be generated.

4.6 Effect comparison

Use the command `detect_yolov5` path to perform object detection on the image.

Use the following codes to verify the execution results of pytorch/ f16/ int8 respectively.

The pytorch model is run as follows to get `dog_torch.jpg`:

```
$ detect_yolov5 \
--input ./image/dog.jpg \
--model ./yolov5s.pt \
--output dog_torch.jpg
```

The f16 bmodel is run as follows to get `dog_f16.jpg` :

```
$ detect_yolov5 \
--input ..../image/dog.jpg \
--model yolov5s_pt_1684x_f16.bmodel \
--output dog_f16.jpg
```

The int8 asymmetric bmodel is run as follows to get `dog_int8_sym.jpg` :

```
$ detect_yolov5 \
--input ..../image/dog.jpg \
--model yolov5s_pt_1684x_int8_sym.bmodel \
--output dog_int8_sym.jpg
```

The result images are compared as shown in the figure ([Comparison of TPU-MLIR for YOLOv5s compilation effect](#)).



Fig. 4.1: Comparison of TPU-MLIR for YOLOv5s compilation effect

Due to different operating environments, the final performance will be somewhat different from Fig. 4.1.

CHAPTER 5

Compile the Caffe model

This chapter takes `mobilenet_v2_deploy.prototxt` and `mobilenet_v2.caffemodel` as examples to introduce how to compile and transfer a caffe model to run on the BM1684X platform.

This chapter requires the `tpu_mlir` python package.

5.1 Install tpu_mlir

Go to the Docker container and execute the following command to install `tpu_mlir`:

```
$ pip install tpu_mlir[caffe]  
# or  
$ pip install tpu_mlir-* -py3-none-any.whl[caffe]
```

5.2 Prepare working directory

Please download `tpu-mlir-resource.tar` from [Assets](#) and unzip it, and rename the folder to `tpu_mlir_resource` :

```
$ tar -xvf tpu-mlir-resource.tar  
$ mv regression/ tpu-mlir-resource/
```

Create a `mobilenet_v2` directory, and put both model files and image files into the `mobilenet_v2` directory.

The operation is as follows:

```
1 $ mkdir mobilenet_v2 && cd mobilenet_v2
2 $ wget https://raw.githubusercontent.com/shicai/MobileNet-Caffe/master/mobilenet_v2_deploy.
   ↪prototxt
3 $ wget https://github.com/shicai/MobileNet-Caffe/raw/master/mobilenet_v2.caffemodel
4 $ cp -rf tpu_mlir_resource/dataset/ILSVRC2012 .
5 $ cp -rf tpu_mlir_resource/image .
6 $ mkdir workspace && cd workspace
```

5.3 Caffe to MLIR

The model in this example has a BGR input with mean and scale of 103.94, 116.78, 123.68 and 0.017, 0.017, 0.017 respectively.

The model conversion command:

```
$ model_transform \
  --model_name mobilenet_v2 \
  --model_def ./mobilenet_v2_deploy.prototxt \
  --model_data ./mobilenet_v2.caffemodel \
  --input_shapes [[1,3,224,224]] \
  --resize_dims=256,256 \
  --mean 103.94,116.78,123.68 \
  --scale 0.017,0.017,0.017 \
  --pixel_format bgr \
  --test_input ./image/cat.jpg \
  --test_result mobilenet_v2_top_outputs.npz \
  --mlir mobilenet_v2.mlir
```

After converting to mlir file, a \${model_name}_in_f32.npz file will be generated, which is the input file of the model.

5.4 MLIR to F32 bmodel

Convert the mlir file to the bmodel of f32, the operation method is as follows:

```
$ model_deploy \
  --mlir mobilenet_v2.mlir \
  --quantize F32 \
  --processor bm1684x \
  --test_input mobilenet_v2_in_f32.npz \
  --test_reference mobilenet_v2_top_outputs.npz \
  --tolerance 0.99,0.99 \
  --model mobilenet_v2_1684x_f32.bmodel
```

After compilation, a file named mobilenet_v2_1684x_f32.bmodel is generated.

5.5 MLIR to INT8 bmodel

5.5.1 Calibration table generation

Before converting to the INT8 model, you need to run calibration to get the calibration table. The number of input data is about 100 to 1000 according to the situation.

Then use the calibration table to generate a symmetric or asymmetric bmodel. It is generally not recommended to use the asymmetric one if the symmetric one already meets the requirements, because the performance of the asymmetric model will be slightly worse than the symmetric model.

Here is an example of the existing 100 images from ILSVRC2012 to perform calibration:

```
$ run_calibration mobilenet_v2.mlir \
--dataset .../ILSVRC2012 \
--input_num 100 \
-o mobilenet_v2_cali_table
```

After running the command above, a file named `mobilenet_v2_cali_table` will be generated, which is used as the input file for subsequent compilation of the INT8 model.

5.5.2 Compile to INT8 symmetric quantized model

Execute the following command to convert to the INT8 symmetric quantized model:

```
$ model_deploy \
--mlir mobilenet_v2.mlir \
--quantize INT8 \
--calibration_table mobilenet_v2_cali_table \
--processor bm1684x \
--test_input mobilenet_v2_in_f32.npz \
--test_reference mobilenet_v2_top_outputs.npz \
--tolerance 0.96,0.70 \
--model mobilenet_v2_1684x_int8_sym.bmodel
```

After compilation, a file named `mobilenet_v2_1684x_int8_sym.bmodel` is generated.

CHAPTER 6

Compile the TFLite model

This chapter takes the `lite-model_mobilebert_int8_1.tflite` model as an example to introduce how to compile and transfer a TFLite model to run on the BM1684X platform.

This chapter requires the `tpu_mlir` python package.

6.1 Install tpu_mlir

Go to the Docker container and execute the following command to install `tpu_mlir`:

```
$ pip install tpu_mlir[tensorflow]
# or
$ pip install tpu_mlir-*py3-none-any.whl[tensorflow]
```

6.2 Prepare working directory

Please download `tpu-mlir-resource.tar` from [Assets](#) and unzip it, and rename the folder to `tpu_mlir_resource` :

```
$ tar -xvf tpu-mlir-resource.tar
$ mv regression/ tpu-mlir-resource/
```

Create a `mobilebert_tf` directory, note that it is the same level as `tpu-mlir`, and put the test image file into the `mobilebert_tf` directory.

The operation is as follows:

```
1 $ mkdir mobilebert_tf && cd mobilebert_tf  
2 $ wget -O lite-model_mobilebert_int8_1.tflite https://storage.googleapis.com/tfhub-lite-models/  
→ ̄ree/lite-model/mobilebert/int8/1.tflite  
3 $ cp -rf tpu_mlir_resource/npz_input/squad_data.npz .  
4 $ mkdir workspace && cd workspace
```

6.3 TFLite to MLIR

The model conversion command:

```
$ model_transform \  
--model_name mobilebert_tf \  
--mlir mobilebert_tf.mlir \  
--model_def ./lite-model_mobilebert_int8_1.tflite \  
--test_input ./squad_data.npz \  
--test_result mobilebert_tf_top_outputs.npz \  
--input_shapes [[1,384],[1,384],[1,384]] \  
--channel_format none
```

After converting to mlir file, a `mobilebert_tf_in_f32.npz` file will be generated, which is the input file of the model.

6.4 MLIR to bmodel

This model is a tflite asymmetric quantized model, which can be converted into a bmodel according to the following parameters:

```
$ model_deploy \  
--mlir mobilebert_tf.mlir \  
--quantize INT8 \  
--processor bm1684x \  
--test_input mobilebert_tf_in_f32.npz \  
--test_reference mobilebert_tf_top_outputs.npz \  
--model mobilebert_tf_bm1684x_int8.bmodel
```

Once compiled, a file named `mobilebert_tf_bm1684x_int8.bmodel` is generated.

CHAPTER 7

Quantization and optimization

In deploying neuron network, the accuracy and throughput (inference speed) are critical targets. To achieve high accuracy and high speed, for some networks, mix precision inference is essential.

The mixed-precision method of TPU-MLIR is searching layers in neural network that are not suitable for low-bit quantization to generate a quantize table, which is used to specify these layers to use higher-bit quantization in the model_deploy stage. This chapter will first introduce the current full int8 symmetric quantization of TPU-MLIR, and then explain how to use the existing quantize table automatic generation tools in TPU-MLIR.

7.1 TPU-MLIR Full Int8 Symmetric Quantization

TPU-MLIR adopts full int8 symmetric quantization by default, where full int8 means that all operators, except for those that the compiler defaults to floating-point operations (such as `layernorm`), are quantized to int8. This section introduces how to use the TPU-MLIR full int8 symmetric quantization tool.

After generating the corresponding MLIR file for the model using the `model_transform` command as instructed in the previous tutorial, if you want to perform int8 symmetric quantization on the model, you also need to generate a calibration table `cali_table` using the `run_calibration` command. How to use the parameters of the `run_calibration` command for different types of models to ensure the generated quantized model has good accuracy will be provided in detailed guidance below.

7.1.1 run_calibration Process Introduction

The quantization part of the following figure (:ref:calibration) shows the overall process of the current `run_calibration`, which includes the automatic mixed-precision module `search_qtable`, the automatic calibration method selection module `search_threshold`, cross-layer weight equalization module `weight_equalization`, and bias correction module `bias_correction`, etc. In the following sections, we will provide the usage details of the above methods based on actual situations.

7.1.2 run_calibration Parameter Introduction

The table below provides an introduction to the parameters of the `run_calibration` command.

Table 7.1: `run_calibration.py` parameter

parameter	description
<code>mlir_file</code>	mlir file
<code>sq</code>	open SmoothQuant
<code>we</code>	open <code>weight_equalization</code>
<code>bc</code>	open <code>bias_correction</code>
<code>dataset</code>	calibration dataset
<code>data_list</code>	sample list
<code>input_num</code>	number of calibration sample
<code>inference_num</code>	the number of images required for the inference process with <code>search_qtable</code> and <code>search_threshold</code> is set to 30 by default
<code>bc_inference_num</code>	the number of images required for the inference process with bias correction is set to 30 by default
<code>tune_list</code>	the list of sample used for tuning
<code>tune_num</code>	the number of images for tuning
<code>histogram_bin_num</code>	specify the number of histogram bins for KLD calculation, default is 2048
<code>expected_cos</code>	expect the similarity between the mixed-precision model output of <code>search_qtable</code> and the floating-point model output, with a value range of [0,1], default is 0.99
<code>min_layer_cos</code>	the lower bound of similarity between the quantized output and the floating-point output of the layer in <code>bias_correction</code> ; compensation is required for the layer when it falls below this threshold, with a value range of [0,1], default is 0.99
<code>max_float_layers</code>	set the number of floating-point layers for <code>search_qtable</code> , default is 5
<code>processor</code>	processor type, default is <code>bm1684x</code>

continues on next page

Table 7.1 – continued from previous page

parameter	description
cali_method	select the calibration mode; if this parameter is not added, the default is KLD calibration. “use_percentile9999” uses the 99.99 percentile as the threshold. “use_max” uses the absolute maximum value as the threshold. “use_torch_observer_for_cali” uses torch’s observer for calibration. “use_mse” uses octav for calibration.
fp_type	search_qtable floating-point layer data type
post_process	post-processing path
global_compare_layers	specify the global contrastive layers, for example, layer1,layer2 or layer1:0.3,layer2:0.7.
search	specify the search type, which includes search_qtable, search_threshold, and false. The default is false, which means search is not enabled
transformer	whether it is a transformer model, in search_qtable, if it is a transformer model, a specific acceleration strategy can be assigned, default is False
quantize_method_list	the calibration method used for searching in search_qtable, default is MSE, with selectable range being MSE, KL, MAX, Percentile9999
benchmark_method	specify the similarity calculation method of search_threshold, with the default being cosine similarity (cos)
kurtosis_analysis	Specify the generation of the kurtosis of the activation values for each layer
part_quantize	Specify partial quantization of the model. The calibration table (cali_table) will be automatically generated alongside the quantization table (qtable). Available modes include N_mode, H_mode, or custom_mode, with H_mode generally delivering higher accuracy
custom_operator	Specify the operators to be quantized, which should be used in conjunction with the aforementioned custom_mode
part_asymmetric	When symmetric quantization is enabled, if specific subnets in the model match a defined pattern, the corresponding operators will automatically switch to asymmetric quantization
mix_mode	Specify the mixed-precision types for the search_qtable. Currently supported options are 8_16 and 4_8
cluster	Specify that a clustering algorithm is used to detect sensitive layers during the search_qtable process
quantize_table_o	the mixed-precision quantization table from search_qtable cali_table output path
debug_cmd	debug command
debug_log	log output level

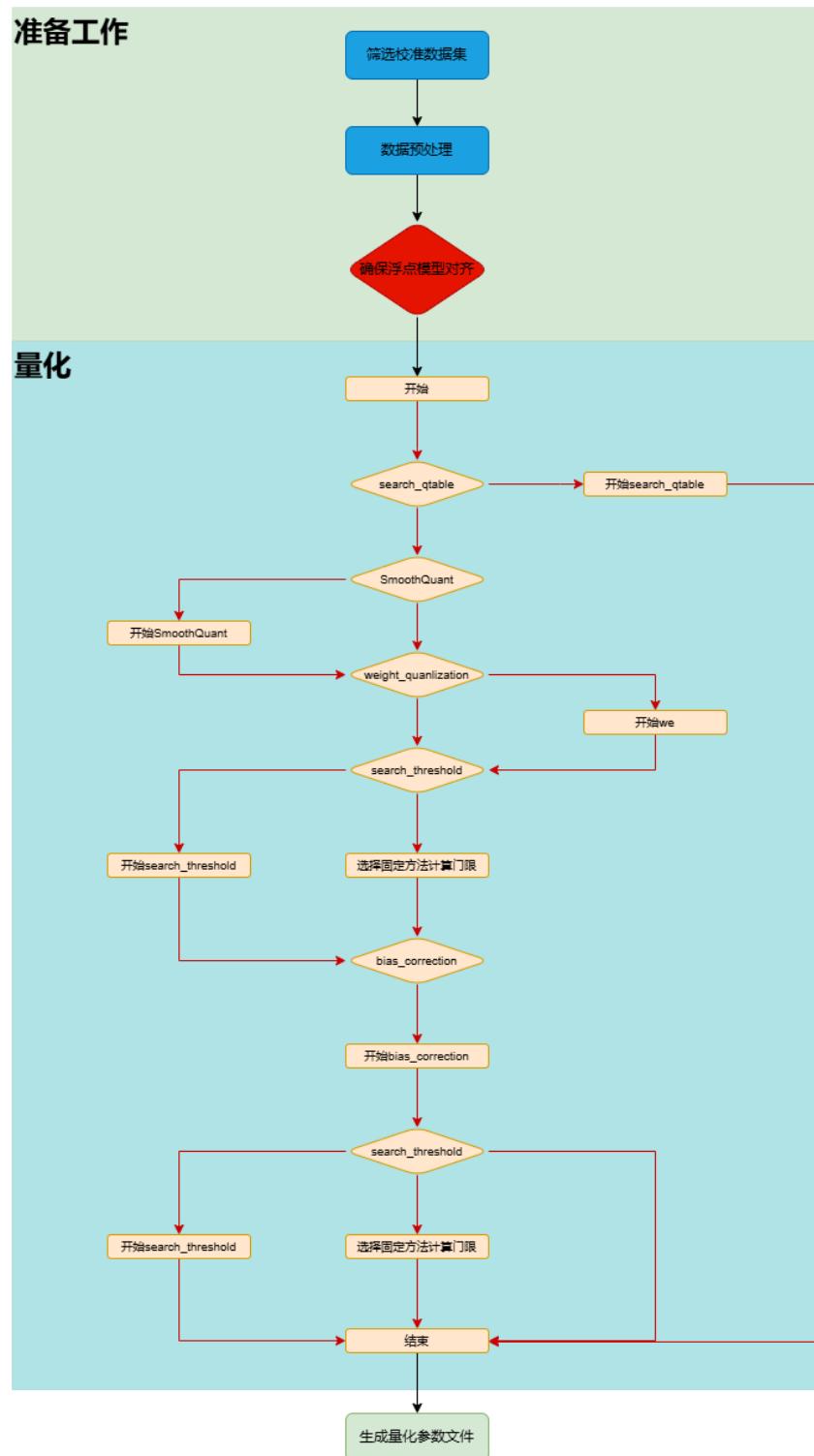


Fig. 7.1: run_calibration process

7.1.3 The Use of run_calibration Parameters Introduction

Based on the user's needs and their understanding of the model itself and quantization, we have provided targeted ways to use the `run_calibration` parameters in different situations.

Table 7.2: applicable scenarios for `run_calibration` parameter

sce-nario	description	quant-i-zation speed	cali-bration method	recommended method
case1	initial model quantization	insensi-tive	unclear	<code>search_threshold</code>
case2	initial model quantization	/	clear	<code>cali_method</code> directly selects the corresponding calibration method
case3	initial model quantization	sensi-tive	unclear	the <code>cali_method</code> selects a fixed calibration method; for details on choosing a specific calibration method, refer to the subsequent sections
case4	after model quantization, the accuracy on the bm1684 processor does not meet the requirements	/	/	<code>open_sq</code> , <code>we</code> and <code>bc</code> methods

case 1: When you perform the initial quantization on your model, which is the first time you use the `run_calibration` command, you may not be clear about the calibration method that is best suited for your current model and you may not be sensitive to the quantization speed. In this case, it is recommended to use the `search_threshold` method. This method can automatically select the calibration method that is most suitable for your current model and output the calibration table `cali_table` generated by this method to the output path you specify. It will also generate a log file `Search_Threshold`, which records the quantization information for different calibration methods. The specific operation is as follows:

```
$ run_calibration mlir.file \
--dataset data_path \
--input_num 100 \
--processor bm1684x \
--search search_threshold \
--inference_num 30 \
-o cali_table
```

Notes: 1. At this point, it is necessary to select the processor parameter, which corresponds to the processor platform on which the model is intended to be deployed. The current default is `bm1684x`. 2. `inference_num` corresponds to the number of inference data required for the `search_threshold` process (this data will be extracted from the dataset you provide). The

larger the `inference_num`, the more accurate the `search_threshold` result, but the longer the quantization time required. Here, the default for `inference_num` is set to 30, which can be customized according to the actual situation.

case2: When quantizing your model for the first time, you already know which calibration method is suitable for the model. At this point, you can directly choose a fixed calibration method based on the `cali_method` parameter. The specific operation is as follows:

```
$ run_calibration mlir.file \
  --dataset data_path \
  --input_num 100 \
  --cali_method use_mse \
  -o cali_table
```

Notes:1.when the `cali_method` parameter is not added, the default KLD calibration method will be used. 2.currently, the `cali_method` supports five options, including `use_mse`, `use_max`, `use_percentile9999`, `use_aciq_gauss` and `use_aciq_laplace`.

case3: When you are sensitive to quantization time and wish to generate the calibration table `cali_table` as quickly as possible, but you are unsure how to choose a calibration method, it is recommended to select a fixed calibration method based on the `cali_method` parameter. In comparison to the quantization speed of TPU-MLIR V1.8, the V1.9 version shows a 100% speed improvement for individual calibration methods, resulting in an average time reduction of around 50%. The acceleration effect is significant. In the V1.9 version, `use_mse` is the fastest calibration method on average. When selecting a calibration method, you can consider the following empirical conclusions:

1.For non-transformer models without attention structure, `use_mse` is a suitable calibration method. Here is a specific operation guide:

```
$ run_calibration mlir.file \
  --dataset data_path \
  --input_num 100 \
  --cali_method use_mse \
  -o cali_table
```

You can also choose the default KLD calibration method. Here is a specific operation guide:

```
$ run_calibration mlir.file \
  --dataset data_path \
  --input_num 100 \
  -o cali_table
```

If neither of the above two methods meets the accuracy requirements, you may need to consider adopting a mixed precision strategy or a hybrid threshold method. More detailed information on these approaches can be found in the subsequent section.

2.When your model is a transformer model that includes an attention structure, you can choose the `use_mse` calibration method. If the `use_mse` calibration method does not produce satisfactory results, you can then consider trying the `use_max` calibration method. Here is a specific operation guide:

```
$ run_calibration mlir.file \
--dataset data_path \
--input_num 100 \
--cali_method use_max \
-o cali_table
```

If the `use_max` method also fails to meet the requirements, at this point, you may need to adopt a mixed precision strategy. You can then try the mixed precision methods that will be introduced later.

Apart from the overall selection rules mentioned above, here are some specific details for choosing calibration methods:1.If your model is a YOLO series object detection model, it is recommended to use the default KLD calibration method.2.If your model is a multi-output classification model, it is also recommended to use the default KLD calibration method.

case4: When your model is deployed on the bm1684 processor and the full int8 quantized model obtained through the methods mentioned above has poor accuracy, you can try enabling SmoothQuant (`sq`), cross-layer weight equalization (`we`) and bias correction (`bc`). To do this, simply add the `sq`, `we` and `bc` parameters to the original command. If you have used `search_threshold` for searching, the operations for adding `sq`, `we` and `bc` are as follows:

```
$ run_calibration mlir.file \
--sq \
--we \
--bc \
--dataset data_path \
--input_num 100 \
--processor bm1684 \
--search search_threshold \
--inference_num 30 \
--bc_inference_num 100 \
-o cali_table
```

If you choose a fixed calibration method using `cali_method` , for example, using `use_mse` , to add the `we` and `bc` methods, the specific operation is as follows:

```
$ run_calibration mlir.file \
--sq \
--we \
--bc \
--dataset data_path \
--input_num 100 \
--processor bm1684 \
--cali_method use_mse \
--bc_inference_num 100 \
-o cali_table
```

If you are using the default KLD calibration method, simply remove the `cali_method` parameter.

Notes:1.Make sure to specify the processor parameter as bm1684. 2.The `bc_inference_num`

parameter is the number of data samples required when using the `bc` quantization method (these samples will be extracted from the dataset you provide), so the number of images should not be too few. 3.The `sq`, `we` and `bc` methods can be used independently. If you choose only the `we` method, simply omit the `sq` and `bc` parameters in the operation. 4. Shape calculation ops will be found and set as float in `model_name_shape_ops` qtable saved in the current directory, the content of this file can be merged by hand with following mix-precision setting files.

7.2 Overview of TPU-MLIR Mixed Precision Quantization

TPU-MLIR provides model mixed precision quantization methods, with its core step being the acquisition of a `quantize_table` ,hereafter referred to as `qtable` that records operator names and their quantization types.

TPU-MLIR provides two paths for obtaining the `qtable`: For typical models, TPU-MLIR provides an experience-based `pattern-match` method. For special or atypical models, PU-MLIR provides three mixed precision quantization methods: `search_qtable` , `run_sensitive_layer` and `fp_forward` The following four section will provide detailed introductions to these four mixed precision methods.

7.3 pattern-match

The `pattern-match` method is integrated into `run_calibration` and does not require explicit parameter specification. Currently, there are two type of models for which experience `qtable` is provided: one is the YOLO series, and the other is the Transformer series (e.g., BERT). After obtaining the `cali_table` , if the model matches an existing pattern, a `qtable` will be generated in the `path/to/cali_table/` folder.

7.3.1 YOLO Series Automatic Mixed Precision Method

Currently `pattern-match` method supported YOLO models include YOLOV5, V6, V7, V8, V9, V10, V11, and V12.

YOLO series models are classic and widely used. When exporting models through official support, post-processing branches with significantly different numerical values are often merged for output, leading to large accuracy loss when quantizing the model to full INT8. Due to the similar structural features of YOLO series models (i.e., a three-level maxpool structure), `pattern-match` automatically identifies whether the model belongs to the YOLO series. If so, operators in the post-processing part will further be recognized and set as float in `qtable`. This `qtable` can be manually merged with the following hybrid precision configurations for use in `model_deploy`. Example of YOLOv8 model output:

¹ `['top.MaxPool', 'top.MaxPool', 'top.MaxPool', 'top.Concat']` (Name: `yolo_block`) is a subset of the F
→main list. Count: 1

(continues on next page)

(continued from previous page)

- 2 The [yolov6_8_9_11_12] post-processing pattern matches this model. Block count: 1
- 3 The [yolov6_8_9_11_12] post-processing pattern is: ['top.Sub', 'top.Add', 'top.Add', 'top.Sub',
→'top.MulConst', 'top.Concat', 'top.Mul', 'top.Concat']
- 4 The qtable has been generated in: path/to/calib_table/qtable !!!

7.3.2 Transformer Series Automatic Mixed Precision Method

Currently pattern-match method supported Transformer series models include BERT, EVA, DeiT, Swin, CSWin, ViT, and DETR.

If the above modules are identified, SiLU, GELU and LayerNorm after Add operators will be set as non-quantized. For ViT, MatMul after Softmax/GELU operators will be identified. For EVA, MatMul after SiLU→Mul and Add operators will be identified. For Swin, Permute before Reshape→LayerNorm, Add and Depth2Space operators will be identified. For DETR, all operators except Conv, Scale, Reshape, and MatMul after LayerNorm/Reshape will be set as non-quantized. These operators are set as non-quantized to generate the qtable.

7.4 1. search_qtable

search_qtable is a mixed precision feature integrated into run_calibration. When full int8 quantization precision does not meet the requirements, mixed precision method are needed, meaning that some operators are set to perform floating-point operations. search_qtable is an optimized version of run_sensitive_layer. In contrast to run_sensitive_layer, search_qtable is faster and supports more customizable parameters. This section takes mobilenet-v2 as example to introduce how to use search_qtable.

This section requires the tpu_mlir python package.

7.4.1 Install tpu_mlir

```
$ pip install tpu_mlir[all]
# or
$ pip install tpu_mlir-*py3-none-any.whl[all]
```

7.4.2 Prepare working directory

Please download tpu-mlir-resource.tar from Assets and unzip it, and rename the folder to tpu_mlir_resource :

```
$ tar -xvf tpu-mlir-resource.tar
$ mv regression/ tpu-mlir-resource/
```

Create a mobilenet-v2 directory, and put both model files and image files into the mobilenet-v2 directory.

The operation is as follows:

```
1 $ mkdir mobilenet-v2 && cd mobilenet-v2
2 $ wget https://github.com/sophgo/tpu-mlir/releases/download/v1.4-beta.0/mobilenet_v2.pt
3 $ cp -rf tpu_mlir_resource/dataset/ILSVRC2012 .
4 $ mkdir workspace && cd workspace
```

7.4.3 Accuracy test of float and int8 models

Step 1: To F32 mlir

```
$ model_transform \
--model_name mobilenet_v2 \
--model_def ./mobilenet_v2.pt \
--input_shapes [[1,3,224,224]] \
--resize_dims 256,256 \
--mean 123.675,116.28,103.53 \
--scale 0.0171,0.0175,0.0174 \
--pixel_format rgb \
--mlir mobilenet_v2.mlir
```

Step 2: Gen calibartion table

Here, we use the use_mse method for calibration.

```
$ run_calibration.py mobilenet_v2.mlir \
--dataset ./ILSVRC2012 \
--input_num 100 \
--cali_method use_mse \
-o mobilenet_v2_cali_table
```

Step 3: To F32 bmodel

```
$ model_deploy \
--mlir mobilenet_v2.mlir \
--quantize F32 \
--processor bm1684 \
--model mobilenet_v2_1684_f32.bmodel
```

Step 4: To INT8 model

```
$ model_deploy \
  --mlir mobilenet_v2.mlir \
  --quantize INT8 \
  --processor bm1684 \
  --calibration_table mobilenet_v2_cali_table \
  --model mobilenet_v2_bm1684_int8_sym.bmodel
```

Step 5: Accuracy test

`classify_mobilenet_v2` is a python program, to run mobilenet-v2 model.

Test the fp32 model:

```
$ classify_mobilenet_v2.py \
  --model_def mobilenet_v2_bm1684_f32.bmodel \
  --input ..../ILSVRC2012/n02090379_7346.JPEG \
  --output mobilenet_v2_fp32_bmodel.jpeg \
  --category_file ..../ILSVRC2012/synset_words.txt
```

The classification information is displayed on the output image. The right label sleeping bag ranks first.

Test the INT8 model:

```
$ classify_mobilenet_v2.py \
  --model_def mobilenet_v2_bm1684_int8_sym.bmodel \
  --input ..../ILSVRC2012/n02090379_7346.JPEG \
  --output mobilenet_v2_INT8_sym_bmodel.jpeg \
  --category_file ..../ILSVRC2012/synset_words.txt
```

The classification information is displayed on the output image. The right label sleeping bag ranks second.

7.4.4 To Mix Precision Model

After int8 conversion, do these commands as below.

Step 1: Execute the `search_qtable` command

The `search_qtable` feature is currently integrated into the `run_calibration` process. Therefore, to use it, you only need to add the relevant parameters to the `run_calibration` command. The parameters related to `search_qtable` in `run_calibration` are explained as follows:



Fig. 7.2: Execution Performance of classify_mobilenet_v2 in FP32



Fig. 7.3: Execution Performance of classify_movenet_v2 in INT8

Table 7.3: search_qtable parameters

Name	Re- quired?	Explanation
(None) dataset	Y N	mlir file Directory of input samples. Images, npz or npy files are placed in this directory
data_list	N	The sample list (cannot be used together with “dataset”)
processor	Y	The platform that the model will use. Support bm1690, bm1688, bm1684x, bm1684, cv186x, cv183x, cv182x, cv181x, cv180x
fp_type	N	Specifies the type of float used for mixing precision. Support auto,F16,F32,BF16. Default is auto, indicating that it is automatically selected by program
input_num	N	The number of samples used for calibration
inference_num	N	The number of samples used for inference, default 30
max_float_layers	N	The number of layers set to float, default 5
tune_list	N	The sample list for tune threshold
tune_num	N	The number of samples for tune threshold, default 5
post_process	N	The user defined post process program path, default None
expected_cos	N	Specify the minimum cos value for the expected final output layer of the network. The default is 0.99
debug_cmd	N	Specifies a debug command string for development. It is empty by default
global_compare_layers	N	global compare layers, for example: layer1,layer2 or layer1:0.3,layer2:0.7
search	Yes	Specify the search type, which includes search_qtable, search_threshold, or false. You need to select search_qtable
transformer	N	Is it a transformer model? In search_qtable, if it is a transformer model, a specific acceleration strategy can be assigned. The default is False
quantize_method_list	N	the calibration method used for searching in search_qtable, default is MSE, with selectable range being MSE, KL, MAX, Percentile9999
quantize_table	Yes	qtable output path
calibration_table	Yes	cali_table output path

`search_qtable` supports user defined post process programs `post_process_func.py`. It can be placed in the current project directory or in another location, if it is placed in another location, you need to specify the full path of the file in the `post_process`. The post process function must be named `PostProcess`, the input data is the output of the network and the output data is the post-processing result. Create the `post_process_func.py` file with the following sample contents:

```
def PostProcess(data):
    print("in post process")
    return data
```

`search_qtable` can customize the calibration method with mixed thresholds, controlled by the parameter `quantize_method_list`. By default, only the MSE calibration method is used for the search. If you want to use a mixed search with KLD and MSE, set the parameter `quantize_method_list` to `KL,MSE`. `search_qtable` has an acceleration strategy for transformer models. If the model is a transformer model with an attention structure, you can set the parameter `transformer` to `True`. Use `search_qtable` to search for layers with significant loss. Note that it is recommended to use bad cases for the search.

In this example, 100 images are used for quantization, 30 images are used for inference, and a mixed search using KLD and MSE calibration methods is performed. Execute the command as follows:

```
$ run_calibration.py mobilenet_v2.mlir \
--dataset .../ILSVRC2012 \
--input_num 100 \
--inference_num 30 \
--expected_cos 0.99 \
--quantize_method_list KL,MSE \
--search search_qtable \
--transformer False \
--processor bm1684 \
--post_process post_process_func.py \
--quantize_table mobilenet_v2_qtable \
--calibration_table mobilenet_v2_cali_table \
```

The final output after execution is printed as follows:

```
the layer input3.1 is 0 sensitive layer, loss is 0.004858517758037473, type is top.Conv
the layer input5.1 is 1 sensitive layer, loss is 0.002798812150635266, type is top.Scale
the layer input11.1 is 2 sensitive layer, loss is 0.0015642610676610547, type is top.Conv
the layer input13.1 is 3 sensitive layer, loss is 0.0009357141882855302, type is top.Scale
the layer input6.1 is 4 sensitive layer, loss is 0.0009211346574943269, type is top.Conv
the layer input2.1 is 5 sensitive layer, loss is 0.0007767164275293004, type is top.Scale
the layer input0.1 is 6 sensitive layer, loss is 0.0006842551513905892, type is top.Conv
the layer input128.1 is 7 sensitive layer, loss is 0.0003780628201499603, type is top.Conv
.....
run result:
int8 outputs_cos:0.986809 old
mix model outputs_cos:0.993372
Output mix quantization table to mobilenet_v2_qtable
total time:667.644282579422
success search qtable
```

Above, `int8 outputs_cos` represents the cosine similarity between network outputs of `int8` model and `float` model; `mix model outputs_cos` represents the cosine similarity between network outputs of `mix model` and `float` model; `total time` represents the search time is 667 seconds. In addition, this program generates a quantization table `mobilenet_v2_qtable`, the

context is as below:

```
# op_name quantize_mode
input3.1 F32
input5.1 F32
input11.1 F32
input13.1 F32
input6.1 F32
```

In the table, the first column represents the corresponding layer, and the second column represents the type. Supported types are F32/F16/BF16/INT8. `search_qtable` will determine the number of mixed precision layers in the qtable based on the user-defined `expected_cos` parameter value. For example, if the `expected_cos` parameter value is equal to 0.99, the number of mixed precision layers in the qtable corresponds to the minimum number of mixed precision layers required to achieve that level of model output comparison. Of course, the number of mixed precision layers in the table will be limited based on the number of model operators. If the minimum number of mixed precision layers exceeds the limitation, only the limited quantity of mixed precision layers will be taken. Additionally, a log file `Search_Qtable` will be generated with the following content:

```
1 INFO:root:quantize_method_list =['KL', 'MSE']
2 INFO:root:run float mode: mobilenet_v2.mlir
3 INFO:root:run int8 mode: mobilenet_v2.mlir
4 INFO:root:all_int8_cos=0.9868090914371674
5 INFO:root:run int8 mode: mobilenet_v2.mlir
6 INFO:root:layer name check pass !
7 INFO:root:all layer number: 117
8 INFO:root:all layer number no float: 116
9 INFO:root:transformer model: False, all search layer number: 116
10 INFO:root:Global metrics layer is : None
11 INFO:root:start to handle layer: input0.1, type: top.Conv
12 INFO:root:adjust layer input0.1 th, with method KL, and threshlod 9.442267236793155
13 INFO:root:run int8 mode: mobilenet_v2.mlir
14 INFO:root:outputs_cos_loss = 0.0006842551513905892
15 INFO:root:adjust layer input0.1 th, with method MSE, and threshlod 9.7417731
16 INFO:root:run int8 mode: mobilenet_v2.mlir
17 INFO:root:outputs_cos_loss = 0.0007242344141149548
18 INFO:root:layer input0.1, layer type is top.Conv, best_th = 9.442267236793155, best_method =F
  ↳KL, best_cos_loss = 0.0006842551513905892
19 .....
```

The log file first provides the custom parameters, including the calibration method used for the mixed threshold `quantize_method_list`, the number of ops to be searched all search layer number and whether it is a transformer model or not. Then, it records the threshold obtained for each op under the given calibration methods (in this case, MSE and KL) and provides the loss of similarity (1 - cosine similarity) between the mixed-precision model using only the corresponding threshold for that operation in int8 computation and the original float model. It also includes the loss information of each operation output on the screen side and the cosine similarity between the final mixed-precision model and the original float model. Users can use the qtable output by the program, or modify the qtable based on the loss information, and then

generate the mixed-precision model. After Search_Qtable is finished, the optimal threshold will be updated to a new quantization table new_cali_table.txt , stored in the current project directory, which needs to be called when generating the mixed-precision model.

Step 2: Gen mix precision model

```
$ model_deploy \
--mlir mobilenet_v2.mlir \
--quantize INT8 \
--processor bm1684 \
--calibration_table new_cali_table.txt \
--quantize_table mobilenet_v2_qtable \
--model mobilenet_v2_bm1684_int8_mix.bmodel
```

Step 3: Test accuracy of mix model

```
$ classify_mobilenet_v2 \
--model_def mobilenet_v2_bm1684_int8_mix.bmodel \
--input .../ILSVRC2012/n02090379_7346.JPEG \
--output mobilenet_v2_INT8_mix_bmodel_1.jpeg \
--category_file .../ILSVRC2012/synset_words.txt
```

The classification information is displayed on the output image. The right label sleeping bag ranks first.

7.5 2. run_sensitive_layer

This section takes mobilenet-v2 as example to introduce how to use sensitive layer search.

This section requires the tpu_mlir python package.

7.5.1 Install tpu_mlir

```
$ pip install tpu_mlir[all]
# or
$ pip install tpu_mlir-* -py3-none-any.whl[all]
```



Fig. 7.4: Execution Performance of classify_mobilenet_v2 in the Mixed Precision Model

7.5.2 Prepare working directory

Please download `tpu-mlir-resource.tar` from `Assets` and unzip it, and rename the folder to `tpu_mlir_resource` :

```
$ tar -xvf tpu-mlir-resource.tar  
$ mv regression/ tpu-mlir-resource/
```

Create a `mobilenet-v2` directory, and put both model files and image files into the `mobilenet-v2` directory.

The operation is as follows:

```
1 $ mkdir mobilenet-v2 && cd mobilenet-v2  
2 $ wget https://github.com/sophgo/tpu-mlir/releases/download/v1.4-beta.0/mobilenet_v2.pt  
3 $ cp -rf tpu_mlir_resource/dataset/ILSVRC2012 .  
4 $ mkdir workspace && cd workspace
```

7.5.3 Accuracy test of float anf int8 models

Step 1: To F32 mlir

```
$ model_transform \  
  --model_name mobilenet_v2 \  
  --model_def ./mobilenet_v2.pt \  
  --input_shapes [[1,3,224,224]] \  
  --resize_dims 256,256 \  
  --mean 123.675,116.28,103.53 \  
  --scale 0.0171,0.0175,0.0174 \  
  --pixel_format rgb \  
  --mlir mobilenet_v2.mlir
```

Step 2: Gen calibartion table

```
$ run_calibration mobilenet_v2.mlir \  
  --dataset ./ILSVRC2012 \  
  --input_num 100 \  
  -o mobilenet_v2_cali_table
```

Step 3: To F32 bmodel

```
$ model_deploy \
--mlir mobilenet_v2.mlir \
--quantize F32 \
--processor bm1684 \
--model mobilenet_v2_1684_f32.bmodel
```

Step 4: To INT8 model

```
$ model_deploy \
--mlir mobilenet_v2.mlir \
--quantize INT8 \
--processor bm1684 \
--calibration_table mobilenet_v2_cali_table \
--model mobilenet_v2_bm1684_int8_sym.bmodel
```

Step 5: Accuracy test

`classify_mobilenet_v2` is a python program, to run mobilenet-v2 model.

Test the fp32 model:

```
$ classify_mobilenet_v2 \
--model_def mobilenet_v2_bm1684_f32.bmodel \
--input .../ILSVRC2012/n01440764_9572.JPEG \
--output mobilenet_v2_fp32_bmodel.jpeg \
--category_file .../ILSVRC2012/synset_words.txt
```

The classification information is displayed on the output image. The right label tench, Tinca tinca ranks first.

```
Top-5
n01440764 tench, Tinca tinca
n02536864 coho, cohoe, coho salmon, blue jack, silver salmon, Oncorhynchus kisutch
n02422106 hartebeest
n02749479 assault rifle, assault gun
n02916936 bulletproof vest
```

Test the INT8 model:

```
$ classify_mobilenet_v2 \
--model_def mobilenet_v2_bm1684_int8_sym.bmodel \
--input .../ILSVRC2012/n01440764_9572.JPEG \
--output mobilenet_v2_INT8_sym_bmodel.jpeg \
--category_file .../ILSVRC2012/synset_words.txt
```

The right label tench, Tinca tinca ranks first.

```
Top-5
n01440764 tench, Tinca tinca
n02749479 assault  file, assau
n02536864 coho, cohoe, coho
n02916936 bulletproof vest
n04336792 stretcher
```

7.5.4 To Mix Precision Model

After int8 conversion, do these commands as below.

Step 1: Search sensitive layers

Use `run_sensitive_layer` and `bad cases` to search sensitive layers, parameters as below:

Table 7.4: run_sensitive_layer parameters

Name	Re- quired?	Explanation
(None) dataset	Y N	mlir file Directory of input samples. Images, npz or npy files are placed in this directory
data_list	N	The sample list (cannot be used together with “dataset”)
calibration_table processor	Y Y	Name of calibration table file The platform that the model will use. Support bm1690, bm1688, bm1684x, bm1684, cv186x, cv183x, cv182x, cv181x, cv180x.
fp_type	N	Specifies the type of float used for mixing precision. Support auto,F16,F32,BF16. Default is auto, indicating that it is automatically selected by program
input_num	N	The number of samples used for calibration, default 10
inference_num	N	The number of samples used for inference, default 10
max_float_layers	N	The number of layers set to float, default 5
tune_list	N	The sample list for tune threshold
tune_num	N	The number of samples for tune threshold, default 5
histogram_bin_num	N	The number of bins used in kld calibration, default 2048
post_process	N	The user defined post process program path, default None
expected_cos	N	Specify the minimum cos value for the expected final output layer of the network. The default is 0.99. The smaller the value, the more layers may be set to floating-point
debug_cmd	N	Specifies a debug command string for development. It is empty by default
o	Y	output quantization table
global_compare_layers	N	global compare layers, for example: layer1,layer2 or layer1:0.3,layer2:0.7
fp_type	N	float type of mix precision

Sensitive layer program supports user defined post process programs `post_process_func.py`. It can be placed in the current project directory or in another location, if it is placed in another location, you need to specify the full path of the file in the `post_process`. The post process function must be named `PostProcess`, the input data is the output of the network and the output data is the post-processing result. Create the `post_process_func.py` file with the following sample contents:

```
def PostProcess(data):
    print("in post process")
    return data
```

In this example, 100 images are used for calibration and 30 images are used for inference, and

the command is as follows:

The operation is as follows:

```
$ run_sensitive_layer mobilenet_v2.mlir \
--dataset ./ILSVRC2012 \
--input_num 100 \
--inference_num 30 \
--calibration_table mobilenet_v2_cali_table \
--processor bm1684 \
--post_process post_process_func.py \
-o mobilenet_v2_qtable
```

The final output after execution is printed as follows:

```
the layer input3.1 is 0 sensitive layer, loss is 0.008808857469573828, type is top.Conv
the layer input11.1 is 1 sensitive layer, loss is 0.0016958347875666302, type is top.Conv
the layer input128.1 is 2 sensitive layer, loss is 0.0015641432811860367, type is top.Conv
the layer input130.1 is 3 sensitive layer, loss is 0.0014325751094084183, type is top.Scale
the layer input127.1 is 4 sensitive layer, loss is 0.0011817314259702227, type is top.Add
the layer input13.1 is 5 sensitive layer, loss is 0.001018420214596527, type is top.Scale
the layer 787 is 6 sensitive layer, loss is 0.0008603856180608993, type is top.Scale
the layer input2.1 is 7 sensitive layer, loss is 0.0007558935451825732, type is top.Scale
the layer input119.1 is 8 sensitive layer, loss is 0.000727441637624282, type is top.Add
the layer input0.1 is 9 sensitive layer, loss is 0.0007138056757098887, type is top.Conv
the layer input110.1 is 10 sensitive layer, loss is 0.000662179506136229, type is top.Conv
.....
run result:
int8 outputs_cos:0.978847 old
mix model outputs_cos:0.989741
Output mix quantization table to mobilenet_v2_qtable
total time:402.15848112106323
success sensitive layer search
```

Above, int8 outputs_cos represents the cosine similarity between network outputs of int8 model and float model; mix model outputs_cos represents the cosine similarity between network outputs of mix model and float model; total time represents the search time is 402 seconds. In addition, this program generates a quantization table mobilenet_v2_qtable, the context is as below:

```
# op_name quantize_mode
input3.1 F32
input11.1 F32
input128.1 F32
input130.1 F32
input127.1 F32
```

The first column in the table is layer name, and the second one is quantization type. Also a log file named SensitiveLayerSearch is generated, its context is as blow:

```
1 INFO:root:start to handle layer: input3.1, type: top.Conv
2 INFO:root:adjust layer input3.1 th, with method MAX, and threshlod 5.5119305
(continues on next page)
```

(continued from previous page)

```

3 INFO:root:run int8 mode: mobilenet_v2.mlir
4 INFO:root:outputs_cos_los = 0.014830573787862011
5 INFO:root:adjust layer input3.1 th, with method Percentile9999, and threshlod 4.1202815
6 INFO:root:run int8 mode: mobilenet_v2.mlir
7 INFO:root:outputs_cos_los = 0.011843443367980822
8 INFO:root:adjust layer input3.1 th, with method KL, and threshlod 2.6186381997094728
9 INFO:root:run int8 mode: mobilenet_v2.mlir
10 INFO:root:outputs_cos_los = 0.008808857469573828
11 INFO:root:layer input3.1, layer type is top.Conv, best_th = 2.6186381997094728, best_method = F
   ↳ KL, best_cos_loss = 0.008808857469573828

```

The log file records the threshold obtained for each operation under different quantization methods (MAX/Percentile9999/KL) and provides the loss of similarity (1 - cosine similarity) between the mixed-precision model using only the corresponding threshold for that operation in int8 computation and the original float model. It also includes the loss information of each operation output on the screen side and the cosine similarity between the final mixed-precision model and the original float model. Users can use the qtable output by the program, or modify the qtable based on the loss information, and then generate the mixed-precision model. After the search for sensitive layers is finished, the optimal threshold will be updated to a new quantization table ‘new_cali_table.txt’ , stored in the current project directory, which needs to be called when generating the mixed-precision model. In this case, based on the output loss information, it was observed that the loss of input3.1 is much higher than that of other operations, which can be set to FP32 only in the qtable.

Step 2: Gen mix precision model

```

$ model_deploy \
  --mlir mobilenet_v2.mlir \
  --quantize INT8 \
  --processor bm1684 \
  --calibration_table new_cali_table.txt \
  --quantize_table mobilenet_v2_qtable \
  --model mobilenet_v2_bm1684_int8_mix.bmodel

```

Step 3: Test accuracy of mix model

```

$ classify_mobilenet_v2 \
  --model_def mobilenet_v2_bm1684_mix.bmodel \
  --input ../../ILSVRC2012/n01440764_9572.JPEG \
  --output mobilenet_v2_INT8_sym_bmodel.jpeg \
  --category_file ../../ILSVRC2012/synset_words.txt

```

The classification results are as follows. The right label tench, Tinca tinca ranks first again.

```
Top-5
n01440764 tench, Tinca tinca
n02749479 assault rifle, assault gun
n02916936 bulletproof vest
n02536864 coho, coho salmon, blue jack, silver salmon, Oncorhynchus kisutch
n04090263 rifle
```

7.6 3. fp_forward

For specific neural networks, some layers may not be suitable for quantization due to significant differences in data distribution. The “Local Non-Quantization” allows you to add certain layers before, after, or between other layers to a mixed-precision table. These layers will not be quantized when generating a mixed-precision model.

In this section, we will continue using the example of the YOLOv5s network mentioned in Chapter 3 and demonstrate how to use the Local Non-Quantization to quickly generate a mix-precision model.

The process of generating FP32 and INT8 models is the same as in Chapter 3. Here, we focus on generating mix-precision model and the accuracy testing.

For YOLO series models, the last three convolutional layers often have significantly different data distributions, and adding them manually to the mixed-precision table can improve accuracy. With the Local Non-Quantization feature, you can search for the corresponding layers from the Top MLIR file generated by `model_transform` and quickly add them to the mixed-precision table using the following command:

```
$ fp_forward \
  yolov5s.mlir \
  --quantize INT8 \
  --processor bm1684x \
  --fpfwd_outputs 474_Conv,326_Conv,622_Conv \
  -o yolov5s_qtable
```

Opening the file “`yolov5s_qtable`” will reveal that the relevant layers have been added to the `qtable`.

Generating the Mixed-Precision Model

```
$ model_deploy \
  --mlir yolov5s.mlir \
  --quantize INT8 \
  --calibration_table yolov5s_cali_table \
  --quantize_table yolov5s_qtable \
  --processor bm1684x \
  --test_input yolov5s_in_f32.npz \
  --test_reference yolov5s_top_outputs.npz \
  --tolerance 0.85,0.45 \
  --model yolov5s_1684x_mix.bmodel
```

Validating the Accuracy of FP32 and Mixed-Precision Models In the model-zoo, there is a program called “yolo” used for accuracy validation of object detection models. You can use the “harness” field in the mlir.config.yaml file to invoke “yolo” as follows:

Modify the relevant fields as follows:

```
$ dataset:  
  imagedir: $(coco2017_val_set)  
  anno: $(coco2017_anno)/instances_val2017.json  
  
harness:  
  type: yolo  
  args:  
    - name: FP32  
      bmodel: $(workdir)/$(name)_bm1684_f32.bmodel  
    - name: INT8  
      bmodel: $(workdir)/$(name)_bm1684_int8_sym.bmodel  
    - name: mix  
      bmodel: $(workdir)/$(name)_bm1684_mix.bmodel
```

Switch to the top-level directory of model-zoo and use tpu_perf.precision_benchmark for accuracy testing, as shown in the following command:

```
$ python3 -m tpu_perf.precision_benchmark yolov5s_path --mlir --target BM1684X --devices 0
```

The accuracy test results will be stored in output/yolo.csv:

mAP for the FP32 model: mAP for the mixed-precision model using the default mixed-precision table:

Performance Testing

mAP for the mixed-precision model using the manually added mixed-precision table:

Parameter Description

Table 7.5: fp_forward parameters

Name	Re- quired?	Explanation
(None)	Y	mlir file
processor	Y	The platform that the model will use. Support bm1690, bm1688, bm1684x, bm1684, cv186x, cv183x, cv182x, cv181x, cv180x.
fpfwd_inputs	N	Specify layers (including this layer) to skip quantization before them. Multiple inputs are separated by commas.
fpfwd_outputs	N	Specify layers (including this layer) to skip quantization after them. Multiple inputs are separated by commas.
fpfwd_blocks	N	Specify the start and end layers between which quantization will be skipped. Start and end layers are separated by colon, and multiple blocks are separated by space.
fp_type	N	Specifies the type of float used for mixing precision. Support auto,F16,F32,BF16. Default is auto, indicating that it is automatically selected by program
o	Y	output quantization table

CHAPTER 8

Use Tensor Computing Processor for Preprocessing

At present, the two main series of processors supported by TPU-MLIR are BM168x (except BM1684) and CV18xx. Both of them support common image preprocessing fusion. The developer can pass the preprocessing arguments during the compilation process, and the compiler will directly insert the corresponding preprocessing operators into the generated model. The generated bmodel or cvimodel can directly use the unpreprocessed image as input and use TPU to do the preprocessing.

Table 8.1: Supported Preprocessing Type

Preprocessing Type	BM168x	CV18xx
Crop	True	True
Normalization	True	True
NHWC to NCHW	True	True
BGR/ RGB Conversion	True	True

The image cropping will first adjust the image to the size specified by the “–resize_dims” argument of the `model_transform` tool, and then crop it to the size of the model input. The normalization supports directly converting unpreprocessed image data.

To integrate preprocessing into the model, you need to specify the “–fuse_preprocess” argument when using the `model_deploy` tool, and the `test_input` should be an image of the original format (i.e., jpg, jpeg and png format). There will be a preprocessed npz file of input named `${model_name} _in_ori.npz` generated. In addition, there is a “–customization_format” argument to specify the original image format input to the model. The supported image formats are described as follows (BM1684 not included):

Table 8.2: Types of customization_format and Description

customization_format	Description	BM168x	CV18xx
None	same with model format, do nothing, as default	True	True
RGB_PLANAR	rgb color order and nchw tensor format	True	True
RGB_PACKED	rgb color order and nhwc tensor format	True	True
BGR_PLANAR	bgr color order and nchw tensor format	True	True
BGR_PACKED	bgr color order and nhwc tensor format	True	True
GRAYSCALE	one color channel only and nchw tensor format	True	True
YUV420_PLANAR	yuv420 planner format, from vpss input	True	True
YUV_NV21	NV21 format of yuv420, from vpss input	True	True
YUV_NV12	NV12 format of yuv420, from vpss input	True	True
RGBA_PLANAR	rgba format and nchw tensor format	False	True

Note that the input data shape of the YUV format in the BM168X model is (n, resize_dim_h, resize_dim_w), `resize_dim_h, resize_dim_w` is the `resize_dim` argument of the `model_transform` stage.

When the order of the color channels in the `customization_format` is different from the model input, a channel conversion operation will be performed. If the `customization_format` argument is not specified, the corresponding `customization_format` will be automatically set according to the `pixel_format` and `channel_format` arguments defined when using the `model_transform` tool.

8.1 Model Deployment Example

Take the mobilenet_v2 model as an example, use the `model_transform` tool to generate the original mlir, and the `run_calibration` tool to generate the calibration table (refer to the chapter “Compiling the Caffe Model” for more details).

8.1.1 Deploy to BM168x

The command to generate the preprocess-fused symmetric INT8 quantized bmodel model is as follows:

```
$ model_deploy \
--mlir mobilenet_v2.mlir \
--quantize INT8 \
--calibration_table mobilenet_v2_cali_table \
--processor bm1684x \
--test_input ../image/cat.jpg \
--test_reference mobilenet_v2_top_outputs.npz \
--tolerance 0.96,0.70 \
```

(continues on next page)

(continued from previous page)

```
--fuse_preprocess \
--model mobilenet_v2_bm1684x_int8_sym_fuse_preprocess.bmodel
```

8.1.2 Deploy to CV18xx

The command to generate the preprocess-fused symmetric INT8 quantized cvimodel model are as follows:

```
$ model_deploy \
--mlir mobilenet_v2.mlir \
--quantize INT8 \
--calibration_table mobilenet_v2_cali_table \
--processor cv183x \
--test_input ./image/cat.jpg \
--test_reference mobilenet_v2_top_outputs.npz \
--tolerance 0.96,0.70 \
--fuse_preprocess \
--customization_format RGB_PLANAR \
--model mobilenet_v2_cv183x_int8_sym_fuse_preprocess.cvimodel
```

vpss input

When the input data comes from the video post-processing module VPSS provided by CV18xx (for details on how to use VPSS for preprocessing, please refer to “CV18xx Media Software Development Reference”), data alignment is required (e.g., 32-bit aligned width), fuse_preprocess and aligned_input need to be set at the same time. The command to generate the preprocessed-fused cvimodel model is as follows:

```
$ model_deploy \
--mlir mobilenet_v2.mlir \
--quantize INT8 \
--calibration_table mobilenet_v2_cali_table \
--processor cv183x \
--test_input ./image/cat.jpg \
--test_reference mobilenet_v2_top_outputs.npz \
--tolerance 0.96,0.70 \
--fuse_preprocess \
--customization_format RGB_PLANAR \
--aligned_input \
--model mobilenet_v2_cv183x_int8_sym_fuse_preprocess_aligned.cvimodel
```

In the above command, aligned_input specifies the alignment that the model input needs to do.

Note that with vpss as input, runtime can use CVI_NN_SetTensorPhysicalAddr to reduce memory data copy.

CHAPTER 9

Use Tensor Computing Processor for Postprocessing

Currently, TPU-MLIR supports integrating the post-processing of YOLO series and SSD network models into the model. The processors currently supporting this function include BM1684X, BM1688, CV186X and BM1690. This chapter will take the conversion of YOLOv5s and YOLOv8s_seg to F16 model as an example to introduce how this function is used.

This chapter requires the tpu_mlir python package.

Go to the Docker container and execute the following command to install tpu_mlir:

```
$ pip install tpu_mlir[onnx]  
# or  
$ pip install tpu_mlir-*py3-none-any.whl[onnx]
```

Please download tpu-mlir-resource.tar from Assets and unzip it, and rename the folder to tpu_mlir_resource :

```
$ tar -xvf tpu-mlir-resource.tar  
$ mv regression/ tpu-mlir-resource/
```

9.1 Add detection post_process(YOLOv5s)

9.1.1 Prepare working directory

Create a model_yolov5s directory, and put both model files and image files into the model_yolov5s directory.

The operation is as follows:

```

1 $ mkdir yolov5s_<code>onnx</code> && <code>cd</code> yolov5s_<code>onnx</code>
2 $ wget https://github.com/ultralytics/yolov5/releases/download/v6.0/yolov5s.onnx
3 $ cp -rf tpu_mlir_resource/dataset/COCO2017 .
4 $ cp -rf tpu_mlir_resource/image .
5 $ mkdir workspace && <code>cd</code> workspace

```

9.1.2 ONNX to MLIR

The model conversion command is as follows:

```

$ model_transform \
--model_name yolov5s \
--model_def ./yolov5s.onnx \
--input_shapes [[1,3,640,640]] \
--mean 0.0,0.0,0.0 \
--scale 0.0039216,0.0039216,0.0039216 \
--keep_aspect_ratio \
--pixel_format rgb \
--output_names 326,474,622 \
--add_postprocess yolov5 \
--test_input ./image/dog.jpg \
--test_result yolov5s_top_outputs.npz \
--mlir yolov5s.mlir

```

There are two points to note here. The first is that the `--add_postprocess` argument needs to be included in the command. The second is that the specified `--output_names` should correspond to the final convolution operation.

The generated `yolov5s.mlir` file finally has a `top.YoloDetection` inserted at the end as follows:

```

1 %260 = "top.Weight"() : () -> tensor<255x512x1x1xf32> loc(#loc261)
2 %261 = "top.Weight"() : () -> tensor<255xf32> loc(#loc262)
3 %262 = "top.Conv"(%253, %260, %261) {dilations = [1, 1], do_relu = false, group = 1 : i64, F
  → kernel_shape = [1, 1], pads = [0, 0, 0, 0], relu_limit = -1.000000e+00 : f64, strides = [1, 1]} : F
  → (tensor<1x512x6x32xf32>, tensor<255x512x1x1xf32>, tensor<255xf32>) -> tensor
  → <1x255x6x32xf32> loc(#loc263)
4 %263 = "top.YoloDetection"(%256, %259, %262) {anchors = [10, 13, 16, 30, 33, 23, 30, 61, 62, 45,
  → 59, 119, 116, 90, 156, 198, 373, 326], class_num = 80 : i64, keep_topk = 200 : i64, net_input
  → h = 640 : i64, net_input_w = 640 : i64, nms_threshold = 5.000000e-01 : f64, num_boxes = 3F
  → : i64, obj_threshold = 0.6999999999999996 : f64, version = "yolov5"} : (tensor
  → <1x255x24x128xf32>, tensor<1x255x12x64xf32>, tensor<1x255x6x32xf32>) -> tensor
  → <1x1x200x7xf32> loc(#loc264)
5 return %263 : tensor<1x1x200x7xf32> loc(#loc)

```

Here you can see that `top.YoloDetection` includes parameters such as anchors, `num_boxes`, and so on. If the post-processing is not standard YOLO, and needs to be changed to other parameters, these parameters in the MLIR file can be directly modified. Also, the output has been changed to one, with the shape of `1x1x200x7`, where 200 represents the maximum number of detection boxes. When there are multiple batches, its value will change to `batchx200`. The 7 elements respectively represent [`batch_number`, `class_id`, `score`, `center_x`, `center_y`, `width`,

height]. The coordinates are relative to the width and length of the model input, it's 640x640 in this example, with the following values:

```
[0., 16., 0.924488, 184.21094, 401.21973, 149.66412, 268.50336 ]
```

9.1.3 MLIR to Bmodel

To convert the MLIR file to an F16 bmodel, proceed as follows:

```
$ model_deploy \
--mlir yolov5s.mlir \
--quantize F16 \
--processor bm1684x \
--fuse_preprocess \
--test_input yolov5s_in_f32.npz \
--test_reference yolov5s_top_outputs.npz \
--model yolov5s_1684x_f16.bmodel
```

Here, the --fuse_preprocess parameter is added in order to integrate the preprocessing into the model as well. In this way, the converted model is a model that includes post-processing. The model information can be viewed with `model_tool` as follows:

```
$ model_tool --info yolov5s_1684x_f16.bmodel
```

```
1 bmodel version: B.2.2
2 processor: BM1684X
3 create time: Wed Jan 3 07:29:14 2024
4
5 kernel_module name: libbm1684x_kernel_module.so
6 kernel_module size: 2677600
7 =====
8 net 0: [yolov5s] static
9 -----
10 stage 0:
11 subnet number: 2
12 input: images_raw, [1, 3, 640, 640], uint8, scale: 1, zero_point: 0
13 output: yolo_post, [1, 1, 200, 7], float32, scale: 1, zero_point: 0
14
15 device mem size: 31238060 (coeff: 14757888, instruct: 124844, runtime: 16355328)
16 host mem size: 0 (coeff: 0, runtime: 0)
```

Here, [1, 1, 200, 7] is the maximum shape, and the actual output varies depending on the number of detected boxes.

9.1.4 Bmodel Verification

In tpu_mlir package, there are yolov5 use cases written in python, using the `detect_yolov5` command to detect objects in images. This command corresponds to the source code path `{package/path/to/tpu_mlir}/python/samples/detect_yolov5.py`. It is used for object detection in images. By reading this code, you can understand how the final output result is transformed into bounding boxes.

The command execution is as follows:

```
$ detect_yolov5 \
--input ..../image/dog.jpg \
--model yolov5s_1684x_f16.bmodel \
--net_input_dims 640,640 \
--fuse_preprocess \
--fuse_postprocess \
--output dog_out.jpg
```

9.2 Add segmentation post_processing(YOLOv8s_seg)

9.2.1 Prepare working directory

Create a `model_yolov8s_seg` directory, export ONNX model file by the official model, and put image files into the `model_yolov8s_seg` directory.

The operation is as follows:

```
1 $ mkdir yolov8s_seg_onnx && cd yolov8s_seg_onnx
2 $ python -c "import torch; from ultralytics import YOLO; model = YOLO('yolov8s-seg.pt'); model.
   ↪export(format='onnx')"
3 $ cp -rf tpu_mlir_resource/dataset/COCO2017 .
4 $ cp -rf tpu_mlir_resource/image .
5 $ mkdir workspace && cd workspace
```

9.2.2 ONNX to MLIR

The model conversion command is as follows:

```
$ model_transform \
--model_name yolov8s_seg \
--model_def ..../yolov8s-seg.onnx \
--input_shapes [[1,3,640,640]] \
--mean 0.0,0.0,0.0 \
--scale 0.0039216,0.0039216,0.0039216 \
--keep_aspect_ratio \
--pixel_format rgb \
--add_postprocess yolov8_seg \
```

(continues on next page)

(continued from previous page)

```
--test_input ./image/dog.jpg \
--test_result yolov8s_seg_top_outputs.npz \
--mlir yolov8s_seg.mlir
```

The generated `yolov8s_seg.mlir` file has several operators named `yolo_seg_post*` inserted at the end, which are used for post-processing operations such as coordinate transformation, NMS, and matrix multiplication.

```

1 %429 = "top.Sigmoid"(%428) {bias = 0.000000e+00 : f64, log = false, round_mode =
  → "HalfAwayFromZero", scale = 1.000000e+00 : f64} : (tensor<8400x25600xf32>) -> tensor
  → <8400x25600xf32> loc(#loc431)
2 %430 = "top.Reshape"(%429) {flatten_start_dim = -1 : i64} : (tensor<8400x25600xf32>) -> [F]
  → tensor<8400x160x160xf32> loc(#loc432)
3 %431 = "top.Slice"(%430, %0, %0, %0) {axes = [], ends = [100, 160, 160], hasparamConvert_
  → axes = [], offset = [0, 0, 0], steps = [1, 1, 1]} : (tensor<8400x160x160xf32>, none, none, none) ->
  → tensor<100x160x160xf32> loc(#loc433)
4 %432 = "top.Slice"(%425, %0, %0, %0) {axes = [], ends = [100, 6], hasparamConvert_ axes = [], [F]
  → offset = [0, 0], steps = [1, 1]} : (tensor<8400x38xf32>, none, none, none) -> tensor<100x6xf32>
  → loc(#loc434)
5 return %431, %432 : tensor<100x160x160xf32>, tensor<100x6xf32> loc(#loc)
```

The model has 2 outputs, which `masks_uncrop_uncompare` is the raw segmentation mask with a shape of `100x160x160`, where 100 represents the maximum number of detection boxes, and `160x160` corresponds to the pixel size of the final feature map.

The `seg_out` represents the detection boxes with a shape of `100x6`, where 100 indicates the maximum number of detection boxes, and the 6 elements respectively represent `[x_left, y_up, x_right, y_bottom, score, class_id]`. Currently, post-processing for segmentation models with multiple batches is not supported, so batch information is not included. The coordinates are relative to the width and length of the model input, it's `640x640` in this example, with the following values:

```
[-74.06776, 263.67566, 74.06777, 531.1172, 0.9523437, 16.]
```

From the raw mask to the final mask output, a resize operation is required to scale it back to the original image size. Then, the excess parts of the mask are cropped based on the detection boxes in `seg_out`. Finally, threshold filtering is applied to the mask to obtain the full-resolution mask. The processing code can be referenced in the workflow of the source file located at `{package/path/to/tpu_mlir}/python/samples/segment_yolo.py`.

9.2.3 MLIR to Bmodel

To convert the MLIR file to an F16 bmodel, proceed as follows:

```
$ fp_forward.py yolov8s_seg.mlir \
--fpfwd_outputs yolo_seg_post_mulconst3 \
--chip bm1684x \
--fp_type F32 \
-o yolov8s_seg_qtable

$ model_deploy \
--mlir yolov8s_seg.mlir \
--quantize F16 \
--processor bm1684x \
--fuse_preprocess \
--quantize_table yolov8s_seg_qtable \
--model yolov8s_seg_1684x_f16.bmodel
```

Here, the --fuse_preprocess parameter is added in order to integrate the preprocessing into the model as well. The yolov8s_seg_qtable is used because post-processing applies an offset operation to the boxes, multiplying their coordinates by a large integer. FP16 can introduce inaccuracies with such operations, leading to excessive masks and reduced post-processing performance. To address this, mixed precision with FP32 is used for these operations.

9.2.4 Bmodel Verification

In tpu_mlir package, there are yolov8s_seg use cases written in python, using the segment_yolo command to detect objects in images. This command corresponds to the source code path {package/path/to/tpu_mlir}/python/samples/segment_yolo.py. It is used for instance segmentation in images. By reading this code, you can understand how the final output result is transformed into masks and bounding boxes.

The command execution is as follows:

```
$ segment_yolo \
--input ../image/dog.jpg \
--model yolov8s_seg_1684x_f16.bmodel \
--net_input_dims 640,640 \
--fuse_preprocess \
--fuse_postprocess \
--output dog_out.jpg
```

CHAPTER 10

Compile LLM Model

10.1 Overview

`llm_convert.py` is a tool for converting large language models (LLM) into the bmodel format. It converts the original model weights to the bmodel format, enabling efficient inference on chip platforms such as BM1684X, BM1688, and CV186AH. Currently supported LLM types include qwen2 and llama, for example, Qwen2-7B-Instruct, Llama-2-7b-chat-hf, etc.

10.2 Command-Line Arguments

Below is an explanation of the command-line arguments supported by this tool:

- `-m, --model_path` (string, required) Specify the path to the original model weights. For example: `./Qwen2-7B-Instruct`
- `-s, --seq_length` (integer, required) Specify the sequence length to be used during the conversion.
- `-q, --quantize` (string, required) Specify the quantization type for the bmodel. You must choose from the following options:
 - `bf16`
 - `w8bf16`
 - `w4bf16`
 - `f16`
 - `w8f16`

- w4f16
- -g, --q_group_size (integer, default: 64) When using the W4A16 quantization mode, this sets the group size for quantization.
- -c, --chip (string, default: bm1684x) Specify the chip platform for generating the bmodel. Supported options are:
 - bm1684x
 - bm1688
 - cv186ah
- --num_device (integer, default: 1) Specify the number of devices for bmodel deployment.
- --num_core (integer, default: 0) Specify the number of cores to be used for bmodel deployment, where 0 indicates using the maximum number of cores.
- --symmetric Set this flag to use symmetric quantization.
- --embedding_disk Set this flag to export the word_embedding as a binary file and run inference on the CPU.
- --max_pixels (integer) For multimodal models such as qwen2.5vl, it is used to specify the maximum pixel dimensions. For example, you can specify 672,896, indicating an image of 672x896; or it can be 602112, representing the maximum number of pixels.
- -o, --out_dir (string, default: ./tmp) Specify the output directory for the generated bmodel files.

10.3 Example Usage

Assume you need to convert a large model located at `/workspace/Qwen2-7B-Instruct` into a bmodel for the `bm1684x` platform, using a sequence length of 384 and the `w4bf16` quantization type. Additionally, set the group size to 128 and store the output files in the directory `qwen2_7b`. You can execute the following command:

First, download Qwen2-7B-Instruct locally from Hugging Face, then run:

```
llm_convert.py -m /workspace/Qwen2-7B-Instruct -s 384 -q w4bf16 -g 128 -c bm1684x -o qwen2_
→ 7b
```

Note: If you encounter an error indicating that `transformers` is not found, you will need to install it. The command is as follows (apply a similar approach for other pip packages):

```
pip3 install transformers --upgrade
```

Also supports AWQ and GPTQ models, such as Qwen2.5-0.5B-Instruct-AWQ and Qwen2.5-0.5B-Instruct-GPTQ-Int4. The conversion command is as follows:

```
llm_convert.py -m /workspace/Qwen2.5-0.5B-Instruct-AWQ -s 384 -q w4bf16 -c bm1684x -o qwen2.  
→ 5_0.5b  
llm_convert.py -m /workspace/Qwen2.5-0.5B-Instruct-GPTQ-Int4 -s 384 -q w4bf16 -c bm1684x -  
→ o qwen2.5_0.5b
```

CHAPTER 11

Appendix.01: Reference for converting model to ONNX format

This chapter provides a reference for how to convert PyTorch, TensorFlow and PaddlePaddle models to ONNX format. You can also refer to the model conversion tutorial provided by ONNX official repository: <https://github.com/onnx/tutorials>. All the operations in this chapter are carried out in the Docker container. For the specific environment configuration method, please refer to the content of Chapter 2.

11.1 PyTorch model to ONNX

This section takes a self-built simple PyTorch model as an example to perform onnx conversion.

11.1.1 Step 0: Create a working directory

Create and enter the torch_model directory using the command line.

```
1 $ mkdir torch_model  
2 $ cd torch_model
```

11.1.2 Step 1: Build and save the model

Create a script named `simple_net.py` in this directory and run it. The specific content of the script is as follows:

```
1 #!/usr/bin/env python3
2 import torch
3
4 # Build a simple nn model
5 class SimpleModel(torch.nn.Module):
6
7     def __init__(self):
8         super(SimpleModel, self).__init__()
9         self.m1 = torch.nn.Conv2d(3, 8, 3, 1, 0)
10        self.m2 = torch.nn.Conv2d(8, 8, 3, 1, 1)
11
12    def forward(self, x):
13        y0 = self.m1(x)
14        y1 = self.m2(y0)
15        y2 = y0 + y1
16        return y2
17
18 # Create a SimpleModel and save its weight in the current directory
19 model = SimpleModel()
20 torch.save(model.state_dict(), "weight.pth")
```

The run command as follows:

```
$ python simple_net.py
```

After running the script, we will get a `weight.pth` weight file in the current directory.

11.1.3 Step 2: Export ONNX model

Create another script named `export_onnx.py` in the same directory and run it. The specific content of the script is as follows:

```
1 #!/usr/bin/env python3
2 import torch
3 from simple_net import SimpleModel
4
5 # Load the pretrained model and export it as onnx
6 model = SimpleModel()
7 model.eval()
8 checkpoint = torch.load("weight.pth", map_location="cpu")
9 model.load_state_dict(checkpoint)
10
11 # Prepare input tensor
12 input = torch.randn(1, 3, 16, 16, requires_grad=True)
13
```

(continues on next page)

(continued from previous page)

```
14 # Export the torch model as onnx
15 torch.onnx.export(model,
16     input,
17     'model.onnx', # name of the exported onnx model
18     opset_version=13,
19     export_params=True,
20     do_constant_folding=True)
```

After running the script, we can get the onnx model named `model.onnx` in the current directory.

11.2 TensorFlow model to ONNX

In this section, we use the `mobilenet_v1_0.25_224` model provided in the TensorFlow official repository as a conversion example.

11.2.1 Step 0: Create a working directory

Create and enter the `tf_model` directory using the command line.

```
1 $ mkdir tf_model
2 $ cd tf_model
```

11.2.2 Step 1: Prepare and convert the model

Download the model with the following commands and use the `tf2onnx` tool to export it as an ONNX model:

```
1 $ wget -nc http://download.tensorflow.org/models/mobilenet_v1_2018_08_02/mobilenet_v1_0.
2   ↪25_224.tgz
3 # tar to get "*.pb" model def file
4 $ tar xzf mobilenet_v1_0.25_224.tgz
5 $ python -m tf2onnx.convert --graphdef mobilenet_v1_0.25_224_frozen.pb \
6     --output mnet_25.onnx --inputs input:0 \
7     --inputs-as-nchw input:0 \
    --outputs MobilenetV1/Predictions/Reshape_1:0
```

After running all commands, we can get the onnx model named `mnet_25.onnx` in the current directory.

11.3 PaddlePaddle model to ONNX

This section uses the SqueezeNet1_1 model provided in the official PaddlePaddle repository as a conversion example. This section requires additional installation of openssl-1.1.1o (ubuntu 22.04 provides openssl-3.0.2 by default).

11.3.1 Step 0: Install openssl-1.1.1o

```
1 wget http://nz2.archive.ubuntu.com/ubuntu/pool/main/o/openssl/libssl1.1_1.1.1f-1ubuntu2.19_
→amd64.deb
2 sudo dpkg -i libssl1.1_1.1.1f-1ubuntu2.19_amd64.deb
```

If the link is expired, check <http://nz2.archive.ubuntu.com/ubuntu/pool/main/o/openssl/?C=M;O=D> for a valid one.

11.3.2 Step 1: Create a working directory

Create and enter the pp_model directory using the command line.

```
1 $ mkdir pp_model
2 $ cd pp_model
```

11.3.3 Step 2: Prepare the model

Download the model with the following commands:

```
1 $ wget https://bj.bcebos.com/paddlehub/fastdeploy/SqueezeNet1_1_infer.tgz
2 $ tar xzf SqueezeNet1_1_infer.tgz
3 $ cd SqueezeNet1_1_infer
```

In addition, use the paddle_infer_shape.py script from the PaddlePaddle project to perform shape inference on the model. The input shape is set to [1,3,224,224] in NCHW format here:

```
1 $ wget https://raw.githubusercontent.com/jiangjiajun/PaddleUtils/main/paddle/paddle_infer_
→shape.py
2 $ python paddle_infer_shape.py --model_dir . \
3   --model_filename inference.pdmodel \
4   --params_filename inference.pdiparams \
5   --save_dir new_model \
6   --input_shape_dict="{'inputs':[1,3,224,224]}"
```

After running all commands, we will be in the SqueezeNet1_1_infer directory, and the new_model directory will be generated in that directory.

11.3.4 Step 3: Convert the model

Install the `paddle2onnx` tool through the following commands, and use this tool to convert the PaddlePaddle model to the ONNX format:

```
1 $ pip install paddle2onnx
2 $ paddle2onnx --model_dir new_model \
3     --model_filename inference.pdmodel \
4     --params_filename inference.pdiparams \
5     --opset_version 13 \
6     --save_file squeezeonnet1_1.onnx
```

After running all the above commands we will get an onnx model named `squeezeonnet1_1.onnx`

CHAPTER 12

Appendix.02: CV18xx Guidance

CV18xx series processor currently supports ONNX and Caffe models but not TFLite models. In terms of quantization, CV18xx supports BF16 and symmetric INT8 format. This chapter takes the CV183X as an example to introduce the compilation and runtime sample of the CV18xx series processor.

12.1 Compile yolov5 model

12.1.1 Install tpu_mlir

Go to the Docker container and execute the following command to install tpu_mlir:

```
$ pip install tpu_mlir[all]
# or
$ pip install tpu_mlir-* -py3-none-any.whl[all]
```

12.1.2 Prepare working directory

Please download tpu-mlir-resource.tar from Assets and unzip it, and rename the folder to tpu_mlir_resource :

```
$ tar -xvf tpu-mlir-resource.tar
$ mv regression/ tpu-mlir-resource/
```

Create a model_yolov5s directory, and put both model files and image files into the model_yolov5s directory.

The operation is as follows:

```

1 $ mkdir model_yolov5s && cd model_yolov5s
2 $ wget https://github.com/ultralytics/yolov5/releases/download/v6.0/yolov5s.onnx
3 $ cp -rf tpu_mlir_resource/dataset/COCO2017 .
4 $ cp -rf tpu_mlir_resource/image .
5 $ mkdir workspace && cd workspace

```

12.1.3 ONNX to MLIR

If the input is an image, we need to learn the preprocessing of the model before conversion. If the model uses the preprocessed npz file as input, there is no need to consider preprocessing. The preprocessing process is expressed as follows (x stands for input):

$$y = (x - \text{mean}) \times \text{scale}$$

The input of yolov5 on the official website is rgb image, each value of it will be multiplied by 1/255, and converted into mean and scale corresponding to 0.0,0.0,0.0 and 0.0039216,0.0039216,0.0039216.

The model conversion command is as follows:

```

$ model_transform \
--model_name yolov5s \
--model_def ./yolov5s.onnx \
--input_shapes [[1,3,640,640]] \
--mean 0.0,0.0,0.0 \
--scale 0.0039216,0.0039216,0.0039216 \
--keep_aspect_ratio \
--pixel_format rgb \
--output_names 326,474,622 \
--test_input ./image/dog.jpg \
--test_result yolov5s_top_outputs.npz \
--mlir yolov5s.mlir

```

For the argument description of `model_transform`, refer to the section [The main parameters of `model_transform`](#).

12.1.4 MLIR to BF16 Model

Convert the mlir file to the cvimodel of bf16, the operation is as follows:

```

$ model_deploy \
--mlir yolov5s.mlir \
--quantize BF16 \
--processor cv183x \
--test_input yolov5s_in_f32.npz \
--test_reference yolov5s_top_outputs.npz \
--model yolov5s_cv183x_bf16.cvimodel

```

For the argument description of `model_deploy`, refer to the section [The main parameters of `model_deploy`](#).

12.1.5 MLIR to INT8 Model

Before converting to the INT8 model, you need to do calibration to get the calibration table. The number of input data depends on the situation but is normally around 100 to 1000. Then use the calibration table to generate INT8 symmetric cvimodel.

Here we use the 100 images from COCO2017 as an example to perform calibration:

```
$ run_calibration yolov5s.mlir \
--dataset ./COCO2017 \
--input_num 100 \
-o yolov5s_cali_table
```

After the operation is completed, a file named `${model_name}_cali_table` will be generated, which is used as the input of the following compilation work.

To convert to symmetric INT8 cvimodel model, execute the following command:

```
$ model_deploy \
--mlir yolov5s.mlir \
--quantize INT8 \
--calibration_table yolov5s_cali_table \
--processor cv183x \
--test_input yolov5s_in_f32.npz \
--test_reference yolov5s_top_outputs.npz \
--tolerance 0.85,0.45 \
--model yolov5s_cv183x_int8_sym.cvimodel
```

After compiling, a file named `${model_name}_cv183x_int8_sym.cvimodel` will be generated.

12.1.6 Result Comparison

The onnx model is run as follows to get `dog_onnx.jpg`:

```
$ detect_yolov5 \
--input ./image/dog.jpg \
--model ./yolov5s.onnx \
--output dog_onnx.jpg
```

The FP32 mlir model is run as follows to get `dog_mlir.jpg`:

```
$ detect_yolov5 \
--input ./image/dog.jpg \
--model yolov5s.mlir \
--output dog_mlir.jpg
```

The BF16 cvimodel is run as follows to get dog_bf16.jpg:

```
$ detect_yolov5 \
--input ..../image/dog.jpg \
--model yolov5s_cv183x_bf16.cvimodel \
--output dog_bf16.jpg
```

The INT8 cvimodel is run as follows to get dog_int8.jpg:

```
$ detect_yolov5 \
--input ..../image/dog.jpg \
--model yolov5s_cv183x_int8_sym.cvimodel \
--output dog_int8.jpg
```

The comparison of the four images is shown in Fig. 12.1, due to the different operating environments, the final effect and accuracy will be slightly different from Fig. 12.1.

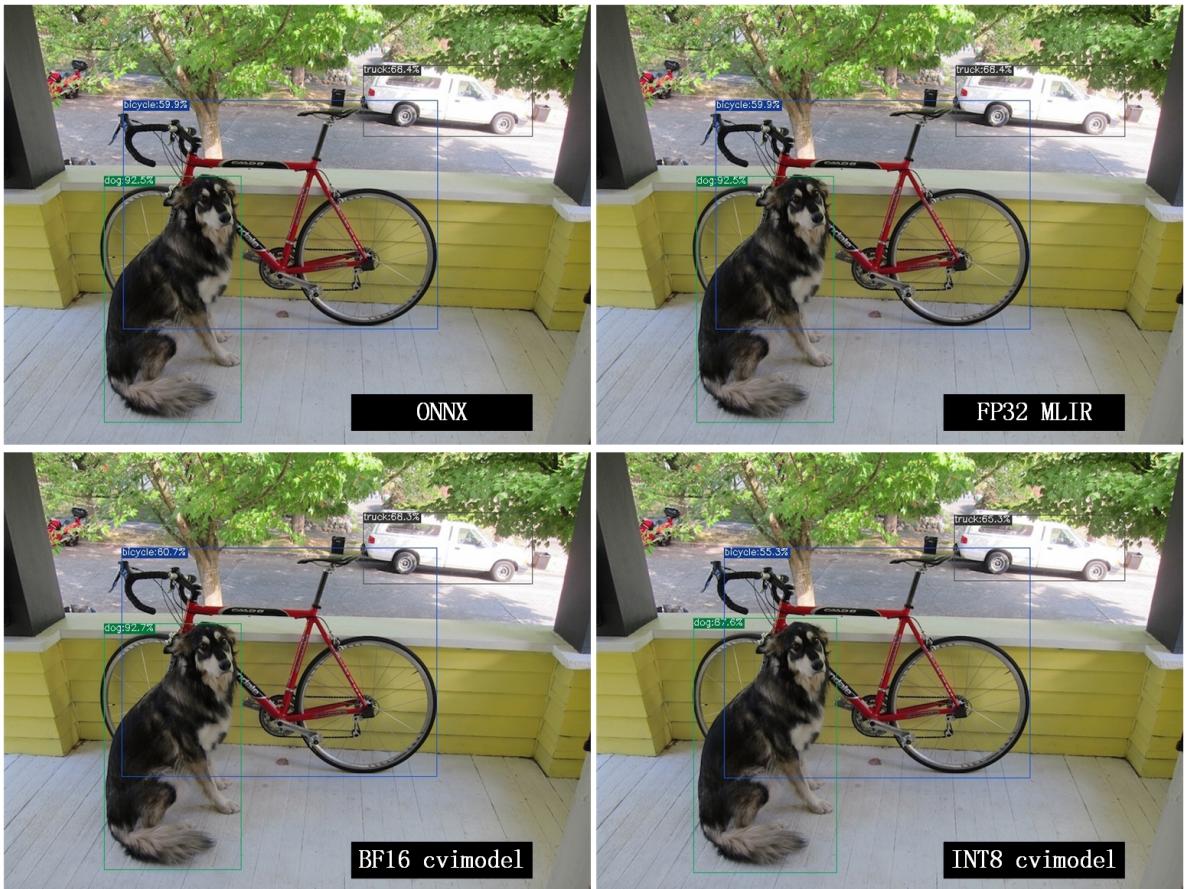


Fig. 12.1: Comparing the results of different models

The above tutorial introduces the process of TPU-MLIR deploying the ONNX model to the CV18xx series processors. For the conversion process of the Caffe model, please refer to the chapter “Compiling the Caffe Model” . You only need to replace the processors name with the specific CV18xx processors.

12.2 Merge cvimodel Files

For the same model, independent cvimodel files can be generated according to the input batch size and resolution(different H and W). However, in order to save storage, you can merge these related cvimodel files into one cvimodel file and share its weight part. The steps are as follows:

12.2.1 Step 0: generate the cvimodel for batch 1

Please refer to the previous section to create a new workspace directory and convert yolov5s to the mlir fp32 model by model_transform

Attention :

- 1.Use the same workspace directory for the cvimodels that need to be merged, and do not share the workspace with other cvimodels that do not need to be merged.
 - 2.In Step 0, Step 1, --merge_weight is required
-

```
$ model_transform \
--model_name yolov5s \
--model_def ./yolov5s.onnx \
--input_shapes [[1,3,640,640]] \
--mean 0.0,0.0,0.0 \
--scale 0.0039216,0.0039216,0.0039216 \
--keep_aspect_ratio \
--pixel_format rgb \
--output_names 326,474,622 \
--test_input ./image/dog.jpg \
--test_result yolov5s_top_outputs.npz \
--mlir yolov5s_bs1.mlir
```

Use the yolov5s_cali_table generated in preceding sections, or generate calibration table by run_calibration.

```
# Add --merge_weight
$ model_deploy \
--mlir yolov5s_bs1.mlir \
--quantize INT8 \
--calibration_table yolov5s_cali_table \
--processor cv183x \
--test_input yolov5s_in_f32.npz \
--test_reference yolov5s_top_outputs.npz \
--tolerance 0.85,0.45 \
--merge_weight \
--model yolov5s_cv183x_int8_sym_bs1.cvimodel
```

12.2.2 Step 1: generate the cvimodel for batch 2

Generate mlir fp32 file in the same workspace:

```
$ model_transform \
--model_name yolov5s \
--model_def ./yolov5s.onnx \
--input_shapes [[2,3,640,640]] \
--mean 0.0,0.0,0.0 \
--scale 0.0039216,0.0039216,0.0039216 \
--keep_aspect_ratio \
--pixel_format rgb \
--output_names 326,474,622 \
--test_input ./image/dog.jpg \
--test_result yolov5s_top_outputs.npz \
--mlir yolov5s_bs2.mlir
```

```
# Add --merge_weight
$ model_deploy \
--mlir yolov5s_bs2.mlir \
--quantize INT8 \
--calibration_table yolov5s_cali_table \
--processor cv183x \
--test_input yolov5s_in_f32.npz \
--test_reference yolov5s_top_outputs.npz \
--tolerance 0.85,0.45 \
--merge_weight \
--model yolov5s_cv183x_int8_sym_bs2.cvimodel
```

12.2.3 Step 2: merge the cvimodel of batch 1 and batch 2

Use model_tool to mrege two cvimodel files:

```
model_tool \
--combine \
yolov5s_cv183x_int8_sym_bs1.cvimodel \
yolov5s_cv183x_int8_sym_bs2.cvimodel \
-o yolov5s_cv183x_int8_sym_bs1_bs2.cvimodel
```

12.2.4 Step 3: use the cvimodel through the runtime interface

Use model_tool to check the program id of bs1 and bs2.:

```
model_tool --info yolov5s_cv183x_int8_sym_bs1_bs2.cvimodel
```

At runtime, you can run different batch program in the following ways:

```

CVI_MODEL_HANDLE bs1_handle;
CVI_RC ret = CVI_NN_RegisterModel("yolov5s_cv183x_int8_sym_bs1_bs2.cvimodel", &bs1_
→handle);
assert(ret == CVI_RC_SUCCESS);
// choice batch 1 program
CVI_NN_SetConfig(bs1_handle, OPTION_PROGRAM_INDEX, 0);
CVI_NN_GetInputOutputTensors(bs1_handle, ...);
....
```



```

CVI_MODEL_HANDLE bs2_handle;
// Reuse loaded cvimodel
CVI_RC ret = CVI_NN_CloneModel(bs1_handle, &bs2_handle);
assert(ret == CVI_RC_SUCCESS);
// choice batch 2 program
CVI_NN_SetConfig(bs2_handle, OPTION_PROGRAM_INDEX, 1);
CVI_NN_GetInputOutputTensors(bs2_handle, ...);
....
```



```
// clean up bs1_handle and bs2_handle
CVI_NN_CleanupModel(bs1_handle);
CVI_NN_CleanupModel(bs2_handle);
```

12.2.5 Overview:

Using the above command, you can merge either the same models or different models

The main steps are:

1. When generating a cvimodel through model_deploy, add the -merge_weight parameter.
2. The work directory of the model to be merged must be the same, and do not clean up any intermediate files before merging the models(Reuse the previous model's weight is implemented through the intermediate file _weight_map.csv).
3. Use model_tool to merge cvimodels.

12.3 Compile and Run the Runtime Sample

This part introduces how to compile and run the runtime samples, include how to cross-compile samples for EVB board and how to compile and run samples in docker. The following 4 samples are included:

- Sample-1 : classifier (mobilenet_v2)
- Sample-2 : classifier_bf16 (mobilenet_v2)
- Sample-3 : classifier fused preprocess (mobilenet_v2)
- Sample-4 : classifier multiple batch (mobilenet_v2)

12.3.1 1) Run the provided pre-build samples

The following files are required:

- cvitek_tpu_sdk_[cv183x|cv182x|cv182x_uclibc|cv181x_glibc32|cv181x_musl_riscv64_rvv|cv180x_musl_riscv64_rvv].tar.gz
- cvimodel_samples_[cv183x|cv182x|cv181x|cv180x].tar.gz

Select the required files according to the processor type and load them into the EVB file system. Execute them on the Linux console of EVB. Here, we take CV183x as an example.

Unzip the model file (delivered in cvimodel format) and the TPU_SDK used by samples. Enter into the samples directory to execute the test. The process is as follows:

```
#env
tar zxf cvimodel_samples_cv183x.tar.gz
export MODEL_PATH=$PWD/cvimodel_samples
tar zxf cvitek_tpu_sdk_cv183x.tar.gz
export TPU_ROOT=$PWD/cvitek_tpu_sdk
cd cvitek_tpu_sdk && source ./envs_tpu_sdk.sh
# get cvimodel info
cd samples
./bin/cvi_sample_model_info $MODEL_PATH/mobilenet_v2.cvimodel

#####
# sample-1 : classifier
#####
./bin/cvi_sample_classifier \
$MODEL_PATH/mobilenet_v2.cvimodel \
./data/cat.jpg \
./data/synset_words.txt

# TOP_K[5]:
# 0.326172, idx 282, n02123159 tiger cat
# 0.326172, idx 285, n02124075 Egyptian cat
# 0.099609, idx 281, n02123045 tabby, tabby cat
# 0.071777, idx 287, n02127052 lynx, catamount
# 0.041504, idx 331, n02326432 hare

#####
# sample-2 : classifier_bf16
#####
./bin/cvi_sample_classifier_bf16 \
$MODEL_PATH/mobilenet_v2_bf16.cvimodel \
./data/cat.jpg \
./data/synset_words.txt

# TOP_K[5]:
# 0.314453, idx 285, n02124075 Egyptian cat
# 0.040039, idx 331, n02326432 hare
# 0.018677, idx 330, n02325366 wood rabbit, cottontail, cottontail rabbit
# 0.010986, idx 463, n02909870 bucket, pail
# 0.010986, idx 852, n04409515 tennis ball
```

(continues on next page)

(continued from previous page)

```
#####
# sample-3 : classifier fused preprocess
#####
./bin/cvi_sample_classifier_fused_preprocess \
    $MODEL_PATH/mobilenet_v2_fused_preprocess.cvimodel \
    ./data/cat.jpg \
    ./data/synset_words.txt

# TOP_K[5]:
# 0.326172, idx 282, n02123159 tiger cat
# 0.326172, idx 285, n02124075 Egyptian cat
# 0.099609, idx 281, n02123045 tabby, tabby cat
# 0.071777, idx 287, n02127052 lynx, catamount
# 0.041504, idx 331, n02326432 hare

#####
# sample-4 : classifier multiple batch
#####
./bin/cvi_sample_classifier_multi_batch \
    $MODEL_PATH/mobilenet_v2_bs1_bs4.cvimodel \
    ./data/cat.jpg \
    ./data/synset_words.txt

# TOP_K[5]:
# 0.326172, idx 282, n02123159 tiger cat
# 0.326172, idx 285, n02124075 Egyptian cat
# 0.099609, idx 281, n02123045 tabby, tabby cat
# 0.071777, idx 287, n02127052 lynx, catamount
# 0.041504, idx 331, n02326432 hare
```

At the same time, the script is provided as a reference, and the execution effect is the same as that of direct operation, as follows:

```
./run_classifier.sh
./run_classifier_bf16.sh
./run_classifier_fused_preprocess.sh
./run_classifier_multi_batch.sh
```

There are more samples can be referred in the cvitek_tpu_sdk/samples/samples_extra:

```
./bin/cvi_sample_detector_yolo_v3_fused_preprocess \
    $MODEL_PATH/yolo_v3_416_fused_preprocess_with_detection.cvimodel \
    ./data/dog.jpg \
    yolo_v3_out.jpg

./bin/cvi_sample_detector_yolo_v5_fused_preprocess \
    $MODEL_PATH/yolov5s_fused_preprocess.cvimodel \
    ./data/dog.jpg \
```

(continues on next page)

(continued from previous page)

```

yolo_v5_out.jpg

./bin/cvi_sample_detector_yolox_s \
$MODEL_PATH/yolox_s.cvimodel \
./data/dog.jpg \
yolox_s_out.jpg

./bin/cvi_sample_alpha_pose_fused_preprocess \
$MODEL_PATH/yolo_v3_416_fused_preprocess_with_detection.cvimodel \
$MODEL_PATH/alpha_pose_fused_preprocess.cvimodel \
./data/pose_demo_2.jpg \
alpha_pose_out.jpg

./bin/cvi_sample_fd_fr_fused_preprocess \
$MODEL_PATH/retinaface_mnet25_600_fused_preprocess_with_detection.cvimodel \
$MODEL_PATH/arcface_res50_fused_preprocess.cvimodel \
./data/obama1.jpg \
./data/obama2.jpg

```

12.3.2 2) Cross-compile samples

The source code is given in the released packages. You can cross-compile the samples' source code in the docker environment and run them on EVB board according to the following instructions.

The following files are required in this part:

- cvitek_tpu_sdk_[cv183x|cv182x|cv182x_uclibc|cv181x_glibc32|cv181x_musl_riscv64_rvv|cv180x_n
- cvitek_tpu_samples.tar.gz

aarch 64-bit (such as cv183x aarch64-bit platform)

Prepare TPU sdk:

```

tar zxf host-tools.tar.gz
tar zxf cvitek_tpu_sdk_cv183x.tar.gz
export PATH=$PWD/host-tools/gcc/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin:
→$PATH
export TPU_SDK_PATH=$PWD/cvitek_tpu_sdk
cd cvitek_tpu_sdk && source ./envs_tpu_sdk.sh && cd ..

```

Compile samples and install them into “install_samples” directory:

```

tar zxf cvitek_tpu_samples.tar.gz
cd cvitek_tpu_samples
mkdir build_soc
cd build_soc

```

(continues on next page)

(continued from previous page)

```

cmake -G Ninja \
-DCMAKE_BUILD_TYPE=RELEASE \
-DCMAKE_C_FLAGS_RELEASE=-O3 \
-DCMAKE_CXX_FLAGS_RELEASE=-O3 \
-DCMAKE_TOOLCHAIN_FILE=$TPU_SDK_PATH/cmake/toolchain-aarch64-linux.cmake \
-DTPU_SDK_PATH=$TPU_SDK_PATH \
-DOPENCV_PATH=$TPU_SDK_PATH/opencv \
-DCMAKE_INSTALL_PREFIX=../install_samples \
..
cmake --build . --target install

```

arm 32-bit (such as 32-bit cv183x/cv182x platform)

Prepare TPU sdk:

```

tar zxf host-tools.tar.gz
tar zxf cvitek_tpu_sdk_cv182x.tar.gz
export TPU_SDK_PATH=$PWD/cvitek_tpu_sdk
export PATH=$PWD/host-tools/gcc/gcc-linaro-6.3.1-2017.05-x86_64_arm-linux-gnueabihf/bin:
→$PATH
cd cvitek_tpu_sdk && source ./envs_tpu_sdk.sh && cd ..

```

If docker version < 1.7, please update 32-bit system library(just once):

```

dpkg --add-architecture i386
apt-get update
apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386

```

Compile samples and install them into `install_samples` directory:

```

tar zxf cvitek_tpu_samples.tar.gz
cd cvitek_tpu_samples
mkdir build_soc
cd build_soc
cmake -G Ninja \
-DCMAKE_BUILD_TYPE=RELEASE \
-DCMAKE_C_FLAGS_RELEASE=-O3 \
-DCMAKE_CXX_FLAGS_RELEASE=-O3 \
-DCMAKE_TOOLCHAIN_FILE=$TPU_SDK_PATH/cmake/toolchain-linux-gnueabihf.
→cmake \
-DTPU_SDK_PATH=$TPU_SDK_PATH \
-DOPENCV_PATH=$TPU_SDK_PATH/opencv \
-DCMAKE_INSTALL_PREFIX=../install_samples \
..
cmake --build . --target install

```

uclibc 32-bit platform (such as cv182x uclibc platform)

Prepare TPU sdk:

```
tar zxf host-tools.tar.gz
tar zxf cvitek_tpu_sdk_cv182x_uclibc.tar.gz
export TPU_SDK_PATH=$PWD/cvitek_tpu_sdk
export PATH=$PWD/host-tools/gcc/arm-cvitek-linux-uclibcgnueabihf/bin:$PATH
cd cvitek_tpu_sdk && source ./envs_tpu_sdk.sh && cd ..
```

If docker version < 1.7, please update 32-bit system library(just once):

```
dpkg --add-architecture i386
apt-get update
apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386
```

Compile samples and install them into `install_samples` directory:

```
tar zxf cvitek_tpu_samples.tar.gz
cd cvitek_tpu_samples
mkdir build_soc
cd build_soc
cmake -G Ninja \
-DCMAKE_BUILD_TYPE=RELEASE \
-DCMAKE_C_FLAGS_RELEASE=-O3 \
-DCMAKE_CXX_FLAGS_RELEASE=-O3 \
-DCMAKE_TOOLCHAIN_FILE=$TPU_SDK_PATH/cmake/toolchain-linux-uclibc.cmake \
-DTPU_SDK_PATH=$TPU_SDK_PATH \
-DOPENCV_PATH=$TPU_SDK_PATH/opencv \
-DCMAKE_INSTALL_PREFIX=../install_samples \
..
cmake --build . --target install
```

riscv 64-bit musl platform (such as cv180x/cv181x riscv 64-bit musl platform)

Prepare TPU sdk:

```
tar zxf host-tools.tar.gz
tar zxf cvitek_tpu_sdk_cv181x_musl_riscv64_rvv.tar.gz
export TPU_SDK_PATH=$PWD/cvitek_tpu_sdk
export PATH=$PWD/host-tools/gcc/riscv64-linux-musl-x86_64/bin:$PATH
cd cvitek_tpu_sdk && source ./envs_tpu_sdk.sh && cd ..
```

Compile samples and install them into `install_samples` directory:

```
tar zxf cvitek_tpu_samples.tar.gz
cd cvitek_tpu_samples
mkdir build_soc
cd build_soc
cmake -G Ninja \
```

(continues on next page)

(continued from previous page)

```

-DCMAKE_BUILD_TYPE=RELEASE \
-DCMAKE_C_FLAGS_RELEASE=-O3 \
-DCMAKE_CXX_FLAGS_RELEASE=-O3 \
-DCMAKE_TOOLCHAIN_FILE=$TPU_SDK_PATH/cmake/toolchain-riscv64-linux-musl-
→x86_64.cmake \
-DTPU_SDK_PATH=$TPU_SDK_PATH \
-DOPENCV_PATH=$TPU_SDK_PATH/opencv \
-DCMAKE_INSTALL_PREFIX=../install_samples \
..
cmake --build . --target install

```

riscv 64-bit glibc platform(such as cv180x/cv181x 64-bit glibc platform)

Prepare TPU sdk:

```

tar zxf host-tools.tar.gz
tar zxf cvitek_tpu_sdk_cv181x_glibc_riscv64.tar.gz
export TPU_SDK_PATH=$PWD/cvitek_tpu_sdk
export PATH=$PWD/host-tools/gcc/riscv64-linux-x86_64/bin:$PATH
cd cvitek_tpu_sdk && source ./envs_tpu_sdk.sh && cd ..

```

Compile samples and install them into install_samples directory:

```

tar zxf cvitek_tpu_samples.tar.gz
cd cvitek_tpu_samples
mkdir build_soc
cd build_soc
cmake -G Ninja \
-DCMAKE_BUILD_TYPE=RELEASE \
-DCMAKE_C_FLAGS_RELEASE=-O3 \
-DCMAKE_CXX_FLAGS_RELEASE=-O3 \
-DCMAKE_TOOLCHAIN_FILE=$TPU_SDK_PATH/cmake/toolchain-riscv64-linux-x86_64.
→cmake \
-DTPU_SDK_PATH=$TPU_SDK_PATH \
-DOPENCV_PATH=$TPU_SDK_PATH/opencv \
-DCMAKE_INSTALL_PREFIX=../install_samples \
..
cmake --build . --target install

```

12.3.3 3) Run samples in docker environment

The following files are required:

- cvitek_tpu_sdk_x86_64.tar.gz
- cvimodel_samples_[cv183x|cv182x|cv181x|cv180x].tar.gz
- cvitek_tpu_samples.tar.gz

Prepare TPU sdk:

```
tar zxf cvitek_tpu_sdk_x86_64.tar.gz
export TPU_SDK_PATH=$PWD/cvitek_tpu_sdk
cd cvitek_tpu_sdk && source ./envs_tpu_sdk.sh && cd ..
```

Compile samples and install them into install_samples directory:

```
tar zxf cvitek_tpu_samples.tar.gz
cd cvitek_tpu_samples
mkdir build
cd build
cmake -G Ninja \
-DCMAKE_BUILD_TYPE=RELEASE \
-DCMAKE_C_FLAGS_RELEASE=-O3 \
-DCMAKE_CXX_FLAGS_RELEASE=-O3 \
-DTPU_SDK_PATH=$TPU_SDK_PATH \
-DCNPY_PATH=$TPU_SDK_PATH/cnpy \
-DOENCV_PATH=$TPU_SDK_PATH/opencv \
-DCMAKE_INSTALL_PREFIX=../install_samples \
..
cmake --build . --target install
```

Run samples:

```
# envs
tar zxf cvimodel_samples_cv183x.tar.gz
export MODEL_PATH=$PWD/cvimodel_samples
source cvitek_mlir/cvitek_envs.sh

# get cvimodel info
cd ../install_samples
./bin/cvi_sample_model_info $MODEL_PATH/mobilenet_v2.cvimodel
```

Other samples are samely to the instructions of running on EVB board.

12.4 FAQ

12.4.1 Model transformation FAQ

1 Related to model transformation

1.1 Whether pytorch,tensorflow, etc. can be converted directly to cvimodel?

pytorch: Supports the .pt model statically via `jit.trace(torch_model.eval(), inputs).save('model_name.pt')`.

tensorflow / others: It is not supported yet and can be supported indirectly through onnx.

1.2 An error occurs when model_transform is executed

model_transform This command convert the onnx,caffe model into the fp32 mlir. The high probability of error here is that there are unsupported operators or incompatible operator attributes, which can be fed back to the tpu team to solve.

1.3 An error occurs when model_deploy is executed

model_deploy This command quantizes fp32 mlir to int8/bf16mlir, and then converts int8/bf16mlir to cvimodel. In the process of conversion, two similarity comparisons will be involved: one is the quantitative comparison between fp32 mlir and int8/bf16mlir, and the other is the similarity comparison between int8/bf16mlir and the final converted cvimodel. If the similarity comparison fails, the following err will occur:

```
[437 Transpose ] SIMILAR [PASSED]
(1, 3, 20, 20, 85) float32
cosine_similarity = 0.999616
euclidean_similarity = 0.972212
snr_similarity = 21.209481
154 compared
152 passed
1 equal, 0 close, 152 similar
1 failed
1 not_equal_1_not_similar
min_similarity = (0.9813582301139832, 0.7978442697003846, 13.49835753440857)
Target: yolo_v5_s_cv183x_int8_sym_tpu_outputs.npz
Reference: yolo_v5_s_top_outputs.npz
npz compare FAILED.
Compare 437_Transpose: 100%
Traceback (most recent call last):
File "/workspace/python/tools/model_deploy.py", line 286, in <module>
    tool.lowering()
File "/workspace/python/tools/model_deploy.py", line 103, in lowering
    tool.validate_tpu_mlir()
File "/workspace/python/tools/model_deploy.py", line 190, in validate_tpu_mlir
    f32_blobs_compare(self.tpu_npz, self.ref_npz, self.tolerance, self.excepts)
File "/workspace/python/utils/mlir_shell.py", line 172, in f32_blobs_compare
    os.system(cmd)
File "/workspace/python/utils/mlir_shell.py", line 50, in _os_system
    raise RuntimeError("!!Error: {} format(cmd_str))".format(cmd_str))
RuntimeError: !!Error: npz_tool.py compare yolo_v5_s_cv183x_int8_sym_tpu_outputs.npz yolo_v5_s_top_outputs.npz -tolerance 0.96,0.80 --except - -vv
```

Solution: The tolerance parameter is incorrect. During the model conversion process, similarity will be calculated for the output of int8/bf16 mlir and fp32 mlir, and tolerance is to limit the minimum value of similarity. If the calculated minimum value of similarity is lower than the corresponding preset tolerance value, the program will stop execution. Consider making adjustments to tolerance. (If the minimum similarity value is too low, please report it to the tpu team.)

1.4 What is the difference between the pixel_format parameter of model_transform and the customization_format parameter of model_deploy?

Channel_order is the input image type of the original model (only gray/rgb planar/bgr planar is supported),customization_format is the input image type of cvimodel, which is determined by the customer and must be used together with fuse_preprocess. (If the input is a YUV image obtained through VPSS or VI, set customization_format to YUV format.) If pixel_format is inconsistent with customization_format, cvimodel will automatically converts the input to the type specified by pixel_format.

1.5 Whether the multi-input model is supported and how to preprocess it?

Models with multiple input images using different preprocessing methods are not supported.

2 Related to model quantization

2.1 run run_calibration raise KeyError: ‘images’

Please check that the path of the data set is correct.

2.2 How to deal with multiple input problems by running quantization?

When running run_calibration, you can store multiple inputs using .npz, or using the –data_list argument, and the multiple inputs in each row of the data_list are separated by “,” .

2.3 Is the input preprocessed when quantization is performed?

Yes, according to the preprocessing parameters stored in the mlir file, the quantization process is preprocessed by loading the preprocessing parameters.

2.4 The program is killed by the system or the memory allocation fails when run calibration

It is necessary to check whether the memory of the host is enough, and the common model requires about 8G memory. If memory is insufficient, try adding the following parameters when running run_calibration to reduce memory requirements.

--tune_num 2 # default is 5

2.5 Does the calibration table support manual modification?

Supported, but it is not recommended.

3 Others

3.1 Does the converted model support encryption?

Not supported for now.

3.2 What is the difference in inference speed between bf16 model and int8 model?

The theoretical difference is about 3-4 times, and there will be differences for different models, which need to be verified in practice.

3.3 Is dynamic shape supported?

Cvimodel does not support dynamic shape. If several shapes are fixed, independent cvimodel files can be generated through the form of shared weights. See [Merge cvimodel Files](#) for details.

12.4.2 Model performance evaluation FAQ

1 Evaluation process

First converted to bf16 model, through the `model_tool --info xxxx.cvimodel` command to obtain the ION memory and the storage space required by the model , and then execute `model_runner` on the EVB board to evaluate the performance, and then evaluate the accuracy in the business scenario according to the provided sample. After the accuracy of the model output meets the expectation, the same evaluation is performed on the int8 model.

2 After quantization, the accuracy does not match the original model, how to debug?

2.1 Ensure `--test_input`, `--test_reference`, `--compare_all` , `--tolerance` parameters are set up correctly.

2.2 Compare the results of the original model and the bf16 model. If the error is large, check whether the pre-processing and post-processing are correct.

2.3 If int8 model accuracy is poor:

- 1) Verify that the data set used by `run_calibration` is the validation set used when training the model;
- 2) A business scenario data set (typically 100-1000 images) can be added for `run_calibration`.

2.4 Confirm the input type of cvimodel:

- 1) If the `--fuse_preprocess` argument is specified, the input type of cvimodel is `uint8`;
- 2) If `--quant_input` is specified,in general,bf16_cvimodel input type is `fp32,int8_cvimodel` input type is `int8`;
- 3) The input type can also be obtained with `model_tool --info xxx.cvimodel`

3 bf16 model speed is relatively slow,int8 model accuracy does not meet expectations how to do?

Try using a mixed-precision quantization method. See mix precision for details.

12.4.3 Common problems of model deployment

1 The The CVI_NN_Forward interface encounters an error after being invoked for many times or is stuck for a long time

There may be driver or hardware issues that need to be reported to the tpu team for resolution.

2 Is the model preprocessing slow?

2.1 Add the --fuse_preprocess parameter when running model_deploy, which will put the preprocessing inside the Tensor Computing Processor for processing.

2.2 If the image is obtained from vpss or vi, you can use --fuse_preprocess, --aligned_input when converting to the model. Then use an interface such as CVI_NN_SetTensorPhysicalAddr to set the input tensor address directly to the physical address of the image, reducing the data copy time.

3 Are floating-point and fixed-point results the same when comparing the inference results of docker and evb ?

Fixed point has no difference, floating point has difference, but the difference can be ignored.

4 Support multi-model inference parallel?

Multithreading is supported, but models are inferred on Tensor Computing Processor in serial.

5 Fill input tensor related interface

CVI_NN_SetTensorPtr : Set the virtual address of input tensor, and the original tensor memory will not be freed. Inference **copies data** from a user-set virtual address to the original tensor memory.

CVI_NN_SetTensorPhysicalAddr : Set the physical address of input tensor, and the original tensor memory will be freed. Inference directly reads data from the newly set physical address, **data copy is not required**. A Frame obtained from VPSS can call this interface by passing in the Frame's first address. Note that model_deploy must be set --fused_preprocess and --aligned_input .

CVI_NN_SetTensorWithVideoFrame : Fill the Input Tensor with the VideoFrame structure. Note The address of VideoFrame is a physical address. If the model is fused preprocess and aligned_input, it is equivalent to CVI_NN_SetTensorPhysicalAddr, otherwise the VideoFrame data will be copied to the Input Tensor.

`CVI_NN_SetTensorWithAlignedFrames` : Support multi-batch, similar to `CVI_NN_SetTensorWithVideoFrame`.

`CVI_NN_FeedTensorWithFrames` : similar to `CVI_NN_SetTensorWithVideoFrame`.

6 How is ion memory allocated after model loading

6.1 Calling `CVI_NN_RegisterModel` allocates ion memory for weight and cmdbuf (you can see the weight and cmdbuf sizes by using `model_tool`).

6.2 Calling `CVI_NN_GetInputOutputTensors` allocates ion memory for tensor(including `private_gmem`, `shared_gmem`, `io_mem`).

6.3 Calling `CVI_NN_CloneModel` can share weight and cmdbuf memory.

6.4 Other interfaces do not apply for ion memory.

6.5 Shared_gmem of different models can be shared (including multithreading), so initializing shared_gmem of the largest model first will saves ion memory.

7 The model inference time becomes longer after loading the business program

Generally, after services are loaded, the `tdma_exe_ms` becomes longer, but the `tiu_exe_ms` remains unchanged. This is because `tdma_exe_ms` takes time to carry data in memory. If the memory bandwidth is insufficient, the `tdma` time will increase.

suggestion:

- 1) vpss/venc optimize chn and reduce resolution
- 2) Reduces memory copy
- 3) Fill input tensor by using copy-free mode

12.4.4 Others

1 In the cv182x/cv181x/cv180x on-board environment, the taz:invalid option -z decompression fails

Decompress the sdk in other linux environments and then use it on the board. windows does not support soft links. Therefore, decompressing the SDK in Windows may cause the soft links to fail and an error may be reported

2 If tensorflow model is pb form of saved_model, how to convert it to pb form of frozen_model

```
import tensorflow as tf
from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input, decode_
↪predictions
import numpy as np
import tf2onnx
import onnxruntime as rt

img_path = "./cat.jpg"
# pb model and variables should in model dir
pb_file_path = "your model dir"
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
# Or set your preprocess here
x = preprocess_input(x)

model = tf.keras.models.load_model(pb_file_path)
preds = model.predict(x)

# different model input shape and name will differently
spec = (tf.TensorSpec((1, 224, 224, 3), tf.float32, name="input"), )
output_path = model.name + ".onnx"

model_proto, _ = tf2onnx.convert.from_keras(model, input_signature=spec, F
↪opset=13, output_path=output_path)
```

CHAPTER 13

Appendix.03: BM168x Guidance

BM168x series processor currently supports ONNX, pytorch, Caffe and TFLite models. This chapter takes the BM1684x processor as an example to introduce merging bmodel files of the BM168x series processors.

13.1 Merge bmodel Files

For the same model, independent bmodel files can be generated according to the input batch size and resolution(different H and W). However, in order to save storage, you can merge these related bmodel files into one bmodel file and share its weight part. The steps are as follows:

13.1.1 Step 0: generate the bmodel for batch 1

Please refer to the previous section to create a new workspace directory and convert yolov5s to the mlir fp32 model by model_transform

Attention :

- 1.Use the same workspace directory for the bmodels that need to be merged, and do not share the workspace with other bmodels that do not need to be merged.
 - 2.In Step 0, Step 1, -merge_weight is required
-

```
$ model_transform \
--model_name yolov5s \
--model_def ./yolov5s.onnx \
--input_shapes [[1,3,640,640]] \
--mean 0.0,0.0,0.0 \
--scale 0.0039216,0.0039216,0.0039216 \
--keep_aspect_ratio \
--pixel_format rgb \
--output_names 350,498,646 \
--test_input ./image/dog.jpg \
--test_result yolov5s_top_outputs.npz \
--mlir yolov5s_bs1.mlir
```

Use the yolov5s_cali_table generated in preceding sections, or generate calibration table by run_calibration.

```
# Add --merge_weight
$ model_deploy \
--mlir yolov5s_bs1.mlir \
--quantize INT8 \
--calibration_table yolov5s_cali_table \
--processor bm1684x \
--test_input yolov5s_in_f32.npz \
--test_reference yolov5s_top_outputs.npz \
--tolerance 0.85,0.45 \
--merge_weight \
--model yolov5s_bm1684x_int8_sym_bs1.bmodel
```

13.1.2 Step 1: generate the bmodel for batch 2

Generate mlir fp32 file in the same workspace:

```
$ model_transform \
--model_name yolov5s \
--model_def ./yolov5s.onnx \
--input_shapes [[2,3,640,640]] \
--mean 0.0,0.0,0.0 \
--scale 0.0039216,0.0039216,0.0039216 \
--keep_aspect_ratio \
--pixel_format rgb \
--output_names 350,498,646 \
--test_input ./image/dog.jpg \
--test_result yolov5s_top_outputs.npz \
--mlir yolov5s_bs2.mlir
```

```
# Add --merge_weight
$ model_deploy \
--mlir yolov5s_bs2.mlir \
--quantize INT8 \
```

(continues on next page)

(continued from previous page)

```
--calibration_table yolov5s_cali_table \
--processor bm1684x \
--test_input yolov5s_in_f32.npz \
--test_reference yolov5s_top_outputs.npz \
--tolerance 0.85,0.45 \
--merge_weight \
--model yolov5s_bm1684x_int8_sym_bs2.bmodel
```

13.1.3 Step 2: merge the bmodel of batch 1 and batch 2

Use model_tool to merge two bmodel files:

```
model_tool \
--combine \
yolov5s_bm1684x_int8_sym_bs1.bmodel \
yolov5s_bm1684x_int8_sym_bs2.bmodel \
-o yolov5s_bm1684x_int8_sym_bs1_bs2.bmodel
```

13.1.4 Overview:

Using the above command, you can merge either the same models or different models

The main steps are:

1. When generating a bmodel through model_deploy, add the -merge_weight parameter.
2. The work directory of the model to be merged must be the same, and do not clean up any intermediate files before merging the models(Reuse the previous model's weight is implemented through the intermediate file _weight_map.csv).
3. Use model_tool to merge bmodels.

CHAPTER 14

Appendix.04: Model-zoo test

14.1 Test Notification

If the test time exceeds the following time limit, it will be considered abnormal:

- Compilation test: 48 hours
- Performance test: 24 hours
- Accuracy test: 24 hours (currently only BM1684X PCIE needs to perform accuracy test)

14.2 Configure the system environment

If you are using Docker for the first time, use the methods in [Environment Setup](#) to install and configure Docker. At the same time, `git-lfs` will be used in this chapter. If you use `git-lfs` for the first time, you need execute the following commands for installation and configuration in the user's own system (not in Docker container).

```
$ curl -s https://packagecloud.io/install/repositories/github/git-lfs/script.deb.sh | sudo bash  
$ sudo apt-get install git-lfs
```

14.3 Get the model-zoo model

In your working directory, get the model-zoo test package from the SDK package provided by SOPHGO, then create and set up model-zoo as follows:

```
$ mkdir -p model-zoo  
$ tar -xvf path/to/model-zoo_<date>.tar.bz2 --strip-components=1 -C model-zoo
```

The directory structure of model-zoo is as follows:

```
|── config.yaml  
|── requirements.txt  
|── dataset  
|── harness  
|── output  
|── ...
```

- config.yaml: contains generic configuration: dataset directory, model root directory, etc., as well as some reused parameters and commands
- requirements.txt: contains python dependencies for model-zoo.
- dataset: directory contains the datasets of the models in modelzoo, which will be called by tpu_perf as plugins.
- output: directory will be used to store the compiled output bmodel and some intermediate data.
- The other directories contain information and configuration for each model. The directory corresponding to each model has a config.yaml file, which configures the model's name, path and FLOPs, dataset production parameters, and the model's quantization compilation commands.

14.4 Prepare the runtime environment

Install the dependencies needed to run model-zoo on your system (outside of the Docker container):

```
# for ubuntu operating system  
$ sudo apt install build-essential  
$ sudo apt install python3-dev  
$ sudo apt install -y libgl1  
# for centos operating system  
$ sudo yum install make automake gcc gcc-c++ kernel-devel  
$ sudo yum install python-devel  
$ sudo yum install mesa-libGL  
# accuracy tests require the following operations to be performed, performance tests can be [F]  
→ performed without, it is recommended to use Anaconda to create a virtual environment of [F]  
→ python 3.7 or above
```

(continues on next page)

(continued from previous page)

```
$ cd path/to/model-zoo  
$ pip3 install -r requirements.txt
```

In addition, tpu hardware needs to be invoked for performance and accuracy tests, so please install the runtime environment for the TPU hardware.

14.5 Configure SoC device

Note: If your device is a PCIE board, you can skip this section directly.

The performance test only depends on the runtime environment for the TPU hardware, so after packaging models, compiled in the toolchain compilation environment, and `model-zoo`, the performance test can be carried out in the SoC environment by `tpu_perf`. However, the complete `model-zoo` as well as compiled output contents may not be fully copied to the SoC since the storage on the SoC device is limited. Here is a method to run tests on SoC devices through linux nfs remote file system mounts.

First, install the nfs service on the toolchain environment server “host system” :

```
$ sudo apt install nfs-kernel-server
```

Add the following content to `/etc/exports` (configure the shared directory):

```
/the/absolute/path/of/model-zoo *(rw,sync,no_subtree_check,no_root_squash)
```

Where * means that everyone can access the shared directory. Moreover, it can be configured to be accessible by a specific network segment or IP, such as:

```
/the/absolute/path/of/model-zoo 192.168.43.0/24(rw,sync,no_subtree_check,no_root_squash)
```

Then execute the following command to make the configuration take effect:

```
$ sudo exportfs -a  
$ sudo systemctl restart nfs-kernel-server
```

In addition, you need to add read permissions to the images in the dataset directory:

```
$ chmod -R +r path/to/model-zoo/dataset
```

Install the client on the SoC device and mount the shared directory:

```
$ mkdir model-zoo  
$ sudo apt-get install -y nfs-common  
$ sudo mount -t nfs <IP>:/path/to/model-zoo ./model-zoo
```

In this way, the test directory is accessible in the SoC environment. The rest of the SoC test operation is basically the same as that of PCIE. Please refer to the following content

for operation. The difference in command execution position and operating environment has been explained in the execution place.

14.6 Prepare dataset

Note: Due to the limited CPU resources of SoC devices, precision testing is not recommended. Therefore, the SoC device test can skip the data set preparation and precision test part.

14.6.1 ImageNet

Download ImageNet 2012 Dataset .

After unzipping, move the data under Data/CLS_LOC/val to a directory like model-zoo:

```
$ cd path/to/sophon/model-zoo  
$ mkdir -p dataset/ILSVRC2012/ILSVRC2012_img_val  
$ mv path/to/imagenet-object-localization-challenge/Data/CLS_LOC/val dataset/ILSVRC2012/  
→ILSVRC2012_img_val  
# It is also possible to map the dataset directory to dataset/ILSVRC2012/ILSVRC2012_img_val[F  
→through the soft link ln -s]
```

14.6.2 COCO (optional)

If the precision test uses the coco dataset (networks trained with coco such as yolo), please download and unzip it as follows:

```
$ cd path/to/model-zoo/dataset/COCO2017/  
$ wget http://images.cocodataset.org/annotations/annotations_trainval2017.zip  
$ wget http://images.cocodataset.org/zips/val2017.zip  
$ unzip annotations_trainval2017.zip  
$ unzip val2017.zip
```

14.6.3 Vid4 (optional)

If you need precision test on BasicVSR, please download and unzip the Vid4 dataset as follows:

```
$ pip3 install gdown  
$ cd path/to/model-zoo/dataset/basicvsr/  
$ gdown https://drive.google.com/open?id=1ZuvNNLgR85TV_whJoHM7uVb-XW1y70DW --fuzzy  
$ unzip -o Vid4.zip -d eval
```

14.7 Prepare the toolchain compilation environment

It is recommended to use the toolchain software in a docker environment, see [Base environment configuration](#) to install Docker. and execute the following commands in your working directory (the directory which model-zoo is located) to create a Docker container:

```
$ docker pull sophgo/tpuc_dev:v3.4  
$ docker run --name myname -v $PWD:/workspace -it sophgo/tpuc_dev:v3.4
```

If you want to keep the container after it exits, simply remove the `--rm` parameter:

```
$ docker run --name myname -v $PWD:/workspace -it sophgo/tpuc_dev:v3.4 --rm
```

After running the command, it will be in a Docker container. You can the latest tpu-mlir wheel installation package from the SDK package provided by SOPHGO, such as `tpu_mlir-*‑py3-none-any.whl`. Install `tpu_mlir` in the Docker container:

```
$ pip install tpu_mlir-*‑py3-none-any.whl[all]
```

14.8 Model performance and accuracy testing process

14.8.1 Compile the model

The model compilation process needs to be done within Docker, where `tpu_mlir` need to be installed as described above.

`config.yaml` in `model-zoo` configures the test content of the SDK. For example, the configuration file for resnet18 is `model-zoo/vision/classification/resnet18-v2/mlir.config.yaml`.

Execute the following command to compile the `resnet18-v2` model:

```
$ cd ..model-zoo  
$ python3 -m tpu_perf.build --target BM1684X --mlir vision/classification/resnet18-v2/mlir.config.  
→yaml
```

where the `--target` is used to specify the processor model, which currently supports `BM1684`, `BM1684X`, `BM1688`, `BM1690` and `CV186X`.

Execute the following command to compile all the high priority test samples:

```
$ cd ..model-zoo  
$ python3 -m tpu_perf.build --target BM1684X --mlir -l full_cases.txt --priority_filter high
```

Full compilation may require reserving more than 2T of space, please adjust according to actual conditions. The `--clear_if_success` parameter can be used to delete intermediate files after successful compilation, saving space.

The following high priority models will be compiled (Due to continuous additions of models in the model-zoo, only a partial list of models is provided here):

```
* efficientnet-lite4
* mobilenetv2
* resnet18-v2
* resnet50-v2
* shufflenet_v2
* squeezenet1.0
* vgg16
* yolov5s
* ...
```

After the command is finished, you will see the newly generated output folder. This compilation result can be used for performance and accuracy testing without recompilation. But you need modify the properties of the output folder to make it accessible to systems outside of Docker:

```
$ chmod -R a+rwx output
```

14.8.2 Performance test

Running the test needs to be done in an environment outside Docker, it is assumed that you have installed and configured the runtime environment for the TPU hardware, so you can exit the Docker environment:

```
$ exit
```

PCIE board

Run the following commands under the PCIE board to test the performance of the generated high priority model bmodel :

```
$ cd model-zoo
$ python3 -m tpu_perf.run --target BM1684X --mlir -l full_cases.txt --priority_filter high
```

where the --target is used to specify the processor model, which currently supports BM1684 , BM1684X , BM1688 , BM1690 and CV186X .

Note: If multiple SOPHGO accelerator cards are installed on the host, you can specify the running device of tpu_perf by adding --devices id when using tpu_perf. Such as:

```
$ python3 -m tpu_perf.run --target BM1684X --devices 2 --mlir -l full_cases.txt --priority_filter high
```

SoC device

The SoC device uses the following steps to test the performance of the generated high priority model bmodel.

```
$ cd model-zoo
$ python3 -m tpu_perf.run --target BM1684X --mlir -l full_cases.txt --priority_filter high
```

Output results

After that, performance data is available in `output/stats.csv`, in which the running time, computing resource utilization, and bandwidth utilization of the relevant models are recorded. The performance test results for `resnet18-v2` as follows:

```
name,prec,shape,gops,time(ms),mac_utilization,ddr_utilization,processor_usage
resnet18-v2,FP32,1x3x224x224,3.636,6.800,26.73%,10.83%,3.00%
resnet18-v2,FP16,1x3x224x224,3.636,1.231,18.46%,29.65%,2.00%
resnet18-v2,INT8,1x3x224x224,3.636,0.552,20.59%,33.20%,3.00%
resnet18-v2,FP32,4x3x224x224,14.542,26.023,27.94%,3.30%,3.00%
resnet18-v2,FP16,4x3x224x224,14.542,3.278,27.73%,13.01%,2.00%
resnet18-v2,INT8,4x3x224x224,14.542,1.353,33.59%,15.46%,2.00%
```

14.8.3 Precision test

Note: Due to the limited CPU resources of SoC devices, precision testing is not recommended. Therefore, the SoC device test can skip the precision test part.

Running the test needs to be done in an environment outside Docker, it is assumed that you have installed and configured the runtime environment for the TPU hardware, so you can exit the Docker environment:

```
$ exit
```

Run the following commands under the PCIE board to test the precision of the generated high priority model `bmodel` :

```
$ cd model-zoo
$ python3 -m tpu_perf.precision_benchmark --target BM1684X --mlir -l full_cases.txt --priority_
→filter high
```

where the `--target` is used to specify the processor model, which currently supports BM1684 , BM1684X , BM1688 , BM1690 and CV186X .

Note:

- If multiple SOPHGO accelerator cards are installed on the host, you can specify the running device of `tpu_perf` by adding `--devices id` when using `tpu_perf`. Such as:

```
$ python3 -m tpu_perf.precision_benchmark --target BM1684X --devices 2 --mlir -l full_cases.txt --
→priority_filter high
```

- The precision test of BM1688, BM1690, and CV186X requires additional configuration of the following environment variables:

```
$ export SET_NUM_SAMPLES_YOLO=200
$ export SET_NUM_SAMPLES_TOPK=100
$ export SET_NUM_SAMPLES_BERT=200
```

Specific parameter descriptions can be obtained with the following commands:

```
$ python3 -m tpu_perf.precision_benchmark --help
```

The output precision data is available in `output/topk.csv` . The precision results for `resnet18-v2`:

```
name,top1,top5
resnet18-v2-FP32,69.68%,89.23%
resnet18-v2-INT8,69.26%,89.08%
```

CHAPTER 15

Appendix.05: TPU Profile Tool Guidance

This chapter mainly introduces how to use Profile data and Tensor Computing Processor Profile tools to visualize the complete running process of the model to facilitate model performance analysis. Currently, Profile tools support BM1684, BM1684X, BM1688, CV186X and BM1690.

15.1 Compile Bmodel

TPU Profile is a tool for converting Profile data into visual web pages. First, generate bmodel. The following uses the yolov5s model in the tpu-mlir project to demonstrate.

Since Profile data will save some layer information during compilation into bmodel, causing the size of bmodel to increase, it is turned off by default. The way to open it is to call `model_deploy` with the `--debug` option. If this option is not turned on at compile time, some data will be missing when the data obtained by turning on Profile at runtime is visualized. The command to generate a bmodel within Docker is as follows:

```
# generate top mlir
$ model_transform \
--model_name yolov5s \
--model_def ..../yolov5s.onnx \
--input_shapes [[1,3,640,640]] \
--mean 0.0,0.0,0.0 \
--scale 0.0039216,0.0039216,0.0039216 \
--keep_aspect_ratio \
--pixel_format rgb \
--output_names 350,498,646 \
--test_input ..../image/dog.jpg \
```

(continues on next page)

(continued from previous page)

```
--test_result yolov5s_top_outputs.npz \
--mlir yolov5s.mlir
```

```
# convert top mlir to fp16 precision bmodel
$ model_deploy \
  --mlir yolov5s.mlir \
  --quantize F16 \
  --processor BM1684X \
  --test_input yolov5s_in_f32.npz \
  --test_reference yolov5s_top_outputs.npz \
  --model yolov5s_1684x_f16.bmodel \
  --debug
```

Through the above commands, `yolov5s.onnx` is compiled into `yolov5s_bm1684x_f16.bmodel`, and the `--debug` parameter will record the profile data.

15.2 Generate Raw Profile Data

Copy the generated `yolov5s_bm1684x_f16.bmodel` to the running environment with `libsophon`. In the same compilation process, the `Profile` function at runtime is turned off by default to prevent additional time consumption when saving and transmitting profiles. When you need to enable the profile function, set the environment variable `BMRUNTIME_ENABLE_PROFILE=1` before running the compiled application. Then use the model testing tool `bmrt_test` provided in `libsophon` to run `bmodel` and generate profile data. Execute the following command outside of Docker:

```
export BMRUNTIME_ENABLE_PROFILE=1
bmrt_test --bmodel yolov5s_1684x_f16.bmodel
```

The following is the log output after opening Profile:

After the run completes, the `bmprofile_data-1` folder is generated in the current directory, which contains all Profile data.

15.3 Visualize Profile Data

Copy the `bmprofile_data-1` directory back to the `tpu-mlir` project environment inside Docker. `Tpu-mlir` provides the `tpu_profile` script to convert the generated profile data into a web page file for visualization. Execute the following command inside Docker:

```
# Convert the original profile data in the bmprofile_data_0 directory into a web
# page and place it in the bmprofile_out directory
# If there is a graphical interface, the browser will be opened directly and the
# results will be seen directly.
tpu_profile bmprofile_data-1 bmprofile_out --arch BM1684X
```

(continues on next page)

```
[BMRT][load_bmodel:1084] INFO:Loading bmodel from [yolov5s_1684x_f16.bmodel]. Thanks for your patience...
[BMRT][load_bmodel:1023] INFO:pre net num: 0, load net num: 1
[BMRT][show_net_info:1520] INFO: #####
[BMRT][show_net_info:1521] INFO: NetName: yolov5s, Index=0
[BMRT][show_net_info:1523] INFO: ---- stage 0 ----
[BMRT][show_net_info:1532] INFO: Input 0) 'Images' shape=[ 1 3 640 640 ] dtype=FLOAT32 scale=1 zero_point=0
[BMRT][show_net_info:1542] INFO: Output 0) '350_Transpose_f32' shape=[ 1 3 80 80 85 ] dtype=FLOAT32 scale=1 zero_point=0
[BMRT][show_net_info:1542] INFO: Output 1) '498_Transpose_f32' shape=[ 1 3 40 40 85 ] dtype=FLOAT32 scale=1 zero_point=0
[BMRT][show_net_info:1542] INFO: Output 2) '646_Transpose_f32' shape=[ 1 3 20 20 85 ] dtype=FLOAT32 scale=1 zero_point=0
[BMRT][show_net_info:1545] INFO: #####
[BMRT][bmrt_test:782] INFO:==> running network #0, name: yolov5s, loop: 0
[BMRT][bmrt_test:868] INFO:reading input #0, bytesize=4915200
[BMRT][print_array:706] INFO: -> input_data: < 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ... > len=1228800
[BMRT][write_block:295] INFO:write_block: type=1, len=36
[BMRT][write_block:295] INFO:write_block: type=8, len=44
[BMRT][end:76] INFO:bdc record_num=1838, max_record_num=1048576 The saved profile data information
[BMRT][write_block:295] INFO:write_block: type=3, len=58816
[BMRT][end:89] INFO:gdm record_num=196, max_record_num=1048576
[BMRT][write_block:295] INFO:write_block: type=4, len=37632
[BMRT][write_block:295] INFO:write_block: type=5, len=256
[BMRT][write_block:295] INFO:write_block: type=6, len=1072
[BMRT][print_note:94] INFO:*****
[BMRT][print_note:95] INFO: * PROFILE MODE due to BMRUNTIME_ENABLE_PROFILE=1
[BMRT][print_note:96] INFO: * Note: BMRuntime will collect time data during running * Profile Mode Tips to avoid misuse during formal deploying
[BMRT][print_note:97] INFO: * that will cost extra time.
[BMRT][print_note:98] INFO: * Close PROFILE Mode by "unset BMRUNTIME_ENABLE_PROFILE"
[BMRT][print_note:99] INFO:*****
[BMRT][bmrt_test:1065] INFO:reading output #0, bytesize=6528000
[BMRT][print_array:706] INFO: -> output ref_data: < 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ... > len=1632000
[BMRT][bmrt_test:1065] INFO:reading output #1, bytesize=1632000
[BMRT][print_array:706] INFO: -> output ref_data: < 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ... > len=4080000
[BMRT][bmrt_test:1065] INFO:reading output #2, bytesize=4080000
[BMRT][bmrt_test:1065] INFO: -> output ref_data: < 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ... > len=1020000
[BMRT][bmrt_test:1039] INFO:net[yolov5s] stage[0], launch total time is 375609 us (npu 5650 us, cpu 369959 us) CPU Time is not accuracy on Profile Mode
[BMRT][bmrt_test:1042] INFO:++ The network[yolov5s] stage[0] output_data ++
[BMRT][print_array:706] INFO:output data #0 shape: [1 3 80 80 85 ] < 0.30957 -0.289551 0.0744629 -0.203003 -11.9375 -1.54297 -5.25391 -3.05859 -5.39453 -5.64844 ·
95 -6.26172 ... > len=1632000
[BMRT][print_array:706] INFO:output data #1 shape: [1 3 40 40 85 ] < -0.0398254 0.253174 -0.383057 -0.520996 -11.0391 -1.3125 -5.11328 -3.32031 -5.46484 -5.97656
812 -6.29681 ... > len=4080000
[BMRT][print_array:706] INFO:output data #2 shape: [1 3 20 20 85 ] < 0.713379 0.654297 -0.534668 -0.241577 -9.82031 -1.23047 -5.77344 -3.35547 -5.92578 -5.12109 ·
44 -6.63672 ... > len=102000
[BMRT][bmrt_test:1083] INFO:load input time(s): 0.003650
[BMRT][bmrt_test:1084] INFO:calculate time(s): 0.375613
[BMRT][bmrt_test:1085] INFO:get output time(s): 0.003838
[BMRT][bmrt_test:1086] INFO:compare time(s): 0.000827
```

Fig. 15.1: The log output after opening Profile

(continued from previous page)

```
ls bmprofile_out
# echarts.min.js profile_data.js result.html
```

For the BM1688 or CV186X models, if you want to integrate layer information in the profile, you need to copy tensor_location.json and final.mlir in the yolov5s_1684x_f16 directory (the same name as the bmodel) to the bmprofile_data-1 directory. BM1690 does not currently support displaying model layer information.

Open `bmprofile_out/result.html` with a browser to see the profile chart. In addition, there are other uses of this tool, which can be viewed through the command as follows:

```
tpu_profile --help
```

There is a more detailed explanation of the Profile in the development manual for your reference.

CHAPTER 16

Appendix.06: TDB Guidance

This chapter mainly introduces the use of the Tensor Debugger(TDB) tool. TDB provides a debugging window similar to pdb and gdb interfaces, which can be used to debug the BModel running process, including adding breakpoints, single-step execution, viewing memory data, data comparison, and other functions.

This tool currently supports BM1684, BM1684X, and BM1688.

16.1 Preparatory work

Environment configuration

First you need to refer to [Environment Setup chapter](#) to complete the environment configuration, enter the Docker container of TPU-MLIR, and install tpu_mlir in it.

If you have completed the environment configuration, you can ignore this step.

Generate bmodel

Before using TDB, you need to generate the bmodel file through TPU-MLIR, refer to [Compile the ONNX model chapter](#) to generate the bmodel file from the model.

You need to use the following two commands:

```
# Convert the ONNX model to top_mlir
$ model_transform
# Convert top_mlir to bmodel
$ model_deploy
```

Here, the `model_deploy` command needs to add `--debug` and `--compare_all` arguments to save the `tpu_output.npz` file and keep the intermediate data.

When the bmodel is built, a directory with the compilation.bmodel and final.mlir files is automatically generated, this directory is called the Context directory.

16.2 Start TDB

```
$ tdb [-h]
      [--inputs [INPUTS]]
      [--ref_data [REF_DATA ...]]
      [--plugins [PLUGINS]]
      [--ddr_size [DDR_SIZE]] [-v]
      [context_dir]
```

The main parameters of the `tdb` command are described as follows:

Table 16.1: Function of tdb parameters

Name	Required?	Explanation
context_dir	Y	The directory where the bmodel file resides, which is the current directory by default
-h, --help	N	Display help information
--inputs	N	Specify input data for the bmodel file
--ref_data	N	Specify reference data for the bmodel file
--plugins	N	Add extra plugins
--ddr_size	N	Specify the ddr_size of the cmodel
-v, --verbose	N	Use the progress bar

Example of starting TDB:

```
$ tdb
# equivalent to
$ tdb ./
```

16.3 TDB command summary

After entering TDB, press tab twice to get the command prompt. The display looks like this:

```
(tdb)
EOF      continue    enable      n          print      r          skip       w
b        delete      help       next      py         reload    start      watch
break    disable     info       p          q          run       static-check
c        display    list      plugin    quit      s          status
```

After entering TDB, the following commands can be used:

Table 16.2: TDB command summary

Command	Explanation
s/start	Load the bmodel and initialize it
r/run	Execute from the beginning to the end, the run instruction contains the initialization function
b/break	Add breakpoints in final.mlir
delete	Delete breakpoint
n/next	Execute the next instruction, you can use n [num] to execute more than one instruction
c/continue	Continue the instruction until the break point or the end of the run
info	Print breakpoint information or instructions in different formats
p/print	Print the current instruction or the data corresponding to the instruction
w/watch	Monitors a certain input/output of the current or previous atomic instruction and returns a prompt when the data at its address changes
q/quit	Quit TDB
py [py_cmd]	Execute python commands in TDB, integrated with pdb's code completion function

Where num represents number; py_cmd denotes the python command.

16.4 TDB usage process

```
# start TDB in context directory
$ cd path/to/context_dir
$ tdb
# initialize
$ s
# execute line by line
$ n
# add breakpoint
$ b
# keep running
$ c
# continue debugging
$ info/p/w
# quit
$ q
```

16.5 TDB function description

16.5.1 next feature

```
# execute line by line use next  
(tdb) n  
# execute multiple instructions  
(tdb) n [num]  
# execute 3 instructions  
(tdb) n 3
```

The instruction displayed after the `n` command is the next unexecuted instruction.

16.5.2 breakpoint feature

Breakpoint feature include viewing breakpoints, adding/removing breakpoints, and turning breakpoints on/off. Here's how to use it:

Table 16.3: breakpoint feature

Command	Explanation	Example
info b/break	View breakpoint information	info b; info break
b/break	Add breakpoint	b 1
enable	Enable breakpoint	enable 1; enable 1,2
disable	Disable breakpoint	disable 1; disable 1,2
delete	Delete breakpoint	delete 1

Currently supported breakpoint types are as follows:

value-id

The Operation prefix in `final.mlir` corresponding to `bmodel`, for example:

```
%140 = "tpu.Load"(%6) {do_bcast = false ...}
```

where `%140` and `%6` are `value-id` , adding this type of breakpoint example is as follows:

```
(tdb) b %140  
(tdb) b %6
```

op-name

The Operation name in `final.mlir` , in the above example, `tpu.Load` is the Op name, add this type of breakpoint example is as follows:

```
(tdb) b tpu.Load
```

cmd-id

The **cmd-id** of asm which is resolved. In the above example, D1 and B0 are **cmd-id**. The example of adding this type of breakpoint is as follows:

```
(tdb) b D2  
(tdb) b B4
```

16.5.3 info feature

The info feature can print breakpoint information or instructions in different formats as follows:

info b

View breakpoint information.

```
(tdb) info b  
index  type enable  text hit  
  1  dialect      y tpu.load  0  
  2  addr         y      R0  3  
  3  cmd-id       y      D1  0  
  4  value-id     y      %7  0
```

info asm

Show the current asm instruction.

```
(tdb) info asm  
%R0, %B15 = "arith.add"(%R13, %C1.0, %D3) {round_mode = 0} : (memref<1x32x54x160xf32, F  
→strides: [8640, 8640, 160, 1], f32, none) -> (memref<1x32x54x160xf32, strides: [8640, 8640, 160,  
→ 1], f32, none)
```

info mlir

Show the Operation in final.mlir that corresponds to the current instruction.

```
(tdb) info mlir  
%137 = "tpu.Active"(%134) {ginfo = #tpu.lg<out_addr = 212992, out_size = 35456, buffer_  
→addr = 0, buffer_size = 71040, eu_align = true, n_idx = [0], n_slice = [1], c_idx = [0], c_  
→slice = [32], d_idx = [0], d_slice = [1], h_idx = [0, 53, 107, 161, 215, 267], h_slice = [54, 55, 55,  
→ 55, 53, 53], w_idx = [0, 159], w_slice = [160, 161], id = 6, stage = 1, group_type = 0>, mode F  
→= #tpu<active_mode SILU>} : (tensor<1x32x320x320xf32>) -> tensor<1x32x320x320xf32> F  
→loc(#loc19)
```

info reg

Show the value of each field after the current command has been parsed.

```
(tdb) info reg  
{'cmd_short': 1, 'cmd_id': 15, 'cmd_id_dep': 3, 'tsk_typ': 3, 'tsk_eu_typ': 2, 'opd0_const': 0,  
→'opd1_const': 1, 'opd2_const': 0, 'tsk_opd_num': 2, 'cmd_id_en': 1, 'pwr_step': 0, 'intr_en': 0,  
→(continues on next page)
```

(continued from previous page)

```

→ 'res0_prec': 2, 'opd0_prec': 2, 'opd1_prec': 2, 'opd2_prec': 0, 'opd0_sign': 1, 'opd1_sign': 1,
→ 'res0_str': 0, 'opd0_str': 0, 'opd1_str': 0, 'opd2_n_str': 0, 'rsvd0': 0, 'res0_n': 1, 'res0_c': 32,
→ 'res0_h': 54, 'res0_w': 160, 'res0_addr': 0, 'opd0_addr': 212992, 'opd1_addr': 1065353216,
→ 'opd2_addr': 0, 'res0_n_str': 0, 'res0_c_str': 0, 'opd0_n_str': 0, 'opd0_c_str': 0, 'opd1_n_str':
→ ': 0, 'opd1_c_str': 0, 'res0_h_str': 0, 'res0_w_str': 0, 'opd0_h_str': 0, 'opd2_sign': 0, 'rsvd1': 0,
→ 'opd0_w_str': 0, 'opd1_h_str': 0, 'opd1_w_str': 0, 'rsvd2': 0}

```

info loc

Show the corresponding Operation information of `tensor_location.json` in the Context directory.

```
(tdb) info loc
{'core_id': 0,
'file_line': 27,
'loc_index': 4,
'opcode': 'tpu.Active',
'operands': [@163840({name=122_Conv, layout=eu_align, slice=[0:1, 0:32, 0:1, 0:54, 0:160], mlir_
→type=tensor<1x32x320x320xf32>, memory_type=<1x32x54x160xf32>}]),
'results': [@212992({name=124_Mul, layout=eu_align, slice=[0:1, 0:32, 0:1, 0:54, 0:160], mlir_
→type=tensor<1x32x320x320xf32>, memory_type=<1x32x54x160xf32>}]),
'slice_all': False,
'subnet_id': 0,
'tiu_dma_id_after': [17, 3],
'tiu_dma_id_before': [1, 3]}}
```

16.5.4 print feature

The print feature not only prints the current asm instruction, but also the input and output data of the instruction, the method of use is as follows:

Table 16.4: print feature

Command	Explanation	Example
p op	Show upcoming commands	p op
p pre/next	Show the previous or next instruction	p pre; p next
p in	Show the input data for the next unexecuted instruction	p in; p in 0
p out	Show the output data of the previous executed instruction	p out; p out 0

16.5.5 watchpoint feature

The watchpoint feature can monitor the input/output data of an instruction and return an alert when the data of a monitored variable changes, the method of use is as follows:

w

Show the currently added watchpoints, see the following example:

```
(tdb) w
index cmd_type cmd_id core_id enabled value
 1 CMDType.dma   2   0     y %G0: memref<1x32x3x36xf32, strides: [3456, 108, 36, 1]>
```

w in

Adds one of the inputs for the next pending instruction as a watchpoint, see the following example:

```
(tdb) n
%R15.2688, %D2 = "dma.tensor"(%G0, %B0) {decompress = False} : (memref<1x32x3x36xf32, F
→ strides: [3456, 108, 36, 1]>, none) -> (memref<1x32x3x36xf32, strides: [108, 108, 36, 1]>, none)
(tdb) w in 0
(tdb) w
index cmd_type cmd_id core_id enabled value
 1 CMDType.dma   2   0     y %G0: memref<1x32x3x36xf32, strides: [3456, 108, 36, 1]>
```

as you can see, **w in 0** adds the first input **%G0** of the next pending instruction as watchpoint.

w out

Adds one of the outputs of the last executed instruction as a watchpoint, see the following example:

```
(tdb) w out 0
(tdb) w
index cmd_type cmd_id core_id enabled value
 1 CMDType.dma   2   0     y    %G0: memref<1x32x3x36xf32, strides: [3456, 108, 36, F
→ 1]>
 2 CMDType.dma   1   0     y %R0: memref<1x3x110x322xf32, strides: [35424, 35424, 322,
→ 1]>
```

p w idx old/now

Prints the value of the added watchpoint, as shown in the following example:

Where **idx** is the index of the watchpoint returned using the **w** command, **old** means to view the data when the watchpoint was originally added, and **now** means to view the current data of the watchpoint.

The **old/now** can be omitted and the default is **now**, which means view the current data of the watchpoint.

```
(tdb) w
index cmd_type cmd_id core_id enabled
  1 CMDType.dma  2  0    y    %G0: memref<1x32x3x36xf32, strides: [3456, 108, 36, F
→ 1]>
  2 CMDType.dma  1  0    y %R0: memref<1x3x110x322xf32, strides: [35424, 35424, 322,
→ 1]>
(tdb) p w 1
(tdb) p w 1 old
```

w delete [idx]

Deletes the added watchpoint, as shown in the following example:

When idx is entered, the corresponding watchpoint will be deleted; when idx is not entered, all watchpoints will be deleted.

```
(tdb) w
index cmd_type cmd_id core_id enabled
  1 CMDType.dma  2  0    y    %G0: memref<1x32x3x36xf32, strides: [3456, 108, 36, F
→ 1]>
  2 CMDType.dma  1  0    y %R0: memref<1x3x110x322xf32, strides: [35424, 35424, 322,
→ 1]>
  3 CMDType.tiu  11  0   y %R13: memref<1x32x54x160xsi16, strides: [8640, 8640, 160,
→ 1]>
(tdb) w delete 1
(tdb) w
index cmd_type cmd_id core_id enabled
  2 CMDType.dma  1  0    y %R0: memref<1x3x110x322xf32, strides: [35424, 35424, 322,
→ 1]>
  3 CMDType.tiu  11  0   y %R13: memref<1x32x54x160xsi16, strides: [8640, 8640, 160,
→ 1]>
(tdb) w delete
(tdb) w
index cmd_type cmd_id core_id enabled value
```

16.5.6 py feature

The py feature can execute python commands directly in the TDB environment, the method of use is as follows:

```
(tdb) py a = 2
(tdb) py b = a + 2
(tdb) py print(b)
4
```

16.6 BModel Disassembler

BModel Disassembler can disassemble the bmodel file to get the assembly code of atomic instruction in MLIR format, which is asm instruction. They are used to analyze the final runtime instruction of the model.

When you use it, you need to enter the Context directory first, and the method of use is as follows:

```
$ bmodel_dis [-h] [--format {mlir,reg,bits,bin,reg-set}] bmodels [bmodels ...]
```

where --format can specify the output format, which default use mlir format, bmodels means the bmodel file to be parsed. Example usage is as follows:

```
$ bmodel_dis compilation.bmodel
$ bmodel_dis --format reg compilation.bmodel
```

The output can be saved to a file as follows:

```
$ bmodel_dis compilation.bmodel > dis_bmodel.mlir
$ bmodel_dis --format reg compilation.bmodel > dis_reg.json
```

16.7 BModel Checker

BModel Checker is used to find errors (codegen errors) in a bmodel, if during model_deploy you find that the generated bmodel cannot be aligned with the tpu's reference data, you can use this tool to locate the error.BModel for BM1684, BM1684X, BM1688 processors is currently supported.

When generating a bmodel file, the model_deploy command needs to add the --debug and -compare_all parameters, which are used to save the tpu_output.npz file and retain intermediate data.

The usage is as follows:

```
$ bmodel_checker [-h]
  [--tolerance TOLERANCE]
  [--report REPORT] [--fail_fast]
  [--quiet] [--no_interactive]
  [--dump_mode {failed,all,never}]
  context_dir reference_data
```

The main parameters of bmodel_checker are described as follows:

Table 16.5: Function of bmodel_checker parameters

Name	Required?	Explanation
context_dir	Y	bmodel file directory
reference_data	Y	tpu_output.npz file location
quiet	N	The execution progress bar is not displayed
fail_fast	N	Stop at the first error
dump_mode	N	Specifies the data to be downloaded by the dump command, the default value is failed, it can also be all or never
tolerance	N	Specify comparison tolerances, default is "0.99,0.90"
report	N	Save the wrong data to file, default is failed_bmodel_outputs.npz
no_interactive	N	After running bmodel_checker, it exits TDB mode directly
cache_mode	No	Cache mode, with three options: online, offline, generate. Default is online.

To use bmodel_checker you need to enter the Context directory, as shown in the following example:

```
$ bmodel_checker ./../yolov5s_bm1684x_f32_tpu_outputs.npz
$ bmodel_checker ./../yolov5s_bm1684x_f32_tpu_outputs.npz --fail_fast
$ bmodel_checker ./../yolov5s_bm1684x_f32_tpu_outputs.npz --tolerance 0.99,0.90
```

After executing the bmodel_checker command, the checker report is output and the error outputs are saved to the failed_bmodel_outputs.npz file, which is described below:

Index	Check-Line-Operand-Result [✓] Summary									
	Check input status			Check output status			Pass mark			
0	(15✓ ?)	(16? ✓)	(17✓ ?)	(18? ✓)	(19✓✓ ✓)	(20✓ ?)	(21? ✓)	(22✓✓ ✓)	(23✓ ✓)	(24✓ ✓)

where the “check” means pass, which the data is checked and its similarity conforms to $\cos > 0.99$, $eul > 0.9$ (This is the default threshold, which can be modified by the tolerance parameter); The “cross” means an error, which the data does not reach the required similarity; The “question mark” means an unknown, which the reference data is not found and the correctness of the data cannot be determined. A complete checker report of a yolov5s model is shown below:

After outputting the check report, it automatically enters the interactive mode. The interactive mode provides a detailed view of the errors and also allows you to quickly jump between lines, as shown in the following example of a cswin_tiny model.

check summary

The check report can be reprinted by using the check summary command:

CHAPTER 16. APPENDIX.06: TDB GUIDANCE

```
(tdb) check summary
(21\|?) (22\|?) (23\|?) (24\|?) (25\|?) (26\|?) (28\|?) (21\|?) (29\|?) (30\|?) (33\|?) (32\|?) (36\|?) (34\|?) (35\|?) (37\|?) (38\|?)
(39\|?) (38\|?) (24\|?) (40\|?) (27\|?) (28\|?) (21\|?) (31\|?) (33\|?) (32\|?) (36\|?) (37\|?) (38\|?) (39\|?) (24\|?) (40\|?) (27\|?) (28\|?)
(28\|?) (21\|?) (31\|?) (33\|?) (32\|?) (36\|?) (37\|?) (39\|?) (38\|?) (24\|?) (40\|?) (27\|?) (28\|?) (21\|?) (31\|?) (33\|?) (32\|?)
(36\|?) (37\|?) (39\|?) (38\|?) (24\|?) (40\|?) (27\|?) (28\|?) (21\|?) (31\|?) (33\|?) (32\|?) (36\|?) (37\|?) (39\|?) (38\|?) (24\|?)
(40\|?) (27\|?) (28\|?) (21\|?) (31\|?) (33\|?) (32\|?) (36\|?) (37\|?) (39\|?) (38\|?) (24\|?) (40\|?) (27\|?) (28\|?) (21\|?) (31\|?)
(33\|?) (32\|?) (36\|?) (37\|?) (39\|?) (38\|?) (24\|?) (40\|?) (27\|?) (28\|?) (21\|?) (31\|?) (33\|?) (32\|?) (36\|?) (37\|?) (39\|?)
(38\|?) (24\|?) (40\|?) (27\|?) (28\|?) (21\|?) (31\|?) (33\|?) (32\|?) (36\|?) (37\|?) (39\|?) (38\|?) (24\|?) (40\|?) (27\|?) (28\|?)
(21\|?) (31\|?) (33\|?) (32\|?) (36\|?) (37\|?) (39\|?) (38\|?) (24\|?) (40\|?) (27\|?) (28\|?) (21\|?) (31\|?) (33\|?) (32\|?) (36\|?)
(37\|?) (39\|?) (38\|?) (24\|?) (40\|?) (27\|?) (28\|?) (54\|?) (31\|?) (33\|?) (32\|?) (36\|?) (37\|?) (56\|?) (39\|?) (38\|?) (55\|?)
(58\|?) (57\|?) (40\|?) (60\|?) (59\|?) (61\|?) (62\|?) (64\|?) (66\|?) (65\|?) (68\|?) (67\|?) (69\|?) (72\|?) (70\|?) (71\|?) (73\|?)
(54\|?) (76\|?) (74\|?) (75\|?) (78\|?) (79\|?) (80\|?) (57\|?) (60\|?) (59\|?) (63\|?) (64\|?) (66\|?) (68\|?) (69\|?) (72\|?)
(73\|?) (54\|?) (76\|?) (78\|?) (79\|?) (58\|?) (88\|?) (57\|?) (60\|?) (59\|?) (63\|?) (64\|?) (68\|?) (69\|?) (72\|?)
(73\|?) (76\|?) (78\|?) (79\|?) (160\|?) (80\|?) (158\|?) (159\|?) (161\|?) (164\|?) (162\|?) (163\|?) (165\|?) (166\|?) (169\|?)
(167\|?) (168\|?) (170\|?) (173\|?) (171\|?) (172\|?) (175\|?) (174\|?) (177\|?) (176\|?) (179\|?) (178\|?) (180\|?) (183\|?) (181\|?) (182\|?)
(184\|?) (185\|?) (188\|?) (186\|?) (187\|?) (189\|?) (192\|?) (190\|?) (191\|?) (193\|?) (196\|?) (194\|?) (195\|?) (198\|?) (197\|?) (200\|?)
(199\|?) (201\|?) (204\|?) (202\|?) (203\|?) (205\|?) (207\|?) (206\|?) (209\|?) (210\|?) (213\|?) (211\|?) (212\|?) (214\|?) (216\|?)
(215\|?) (218\|?) (217\|?) (219\|?) (222\|?) (220\|?) (223\|?) (224\|?) (225\|?) (226\|?) (228\|?) (230\|?) (229\|?) (232\|?)
(231\|?) (234\|?) (233\|?) (236\|?) (235\|?) (239\|?) (237\|?) (238\|?) (241\|?) (240\|?) (242\|?) (244\|?) (246\|?) (245\|?) (248\|?)
(247\|?) (249\|?) (252\|?) (250\|?) (251\|?) (253\|?) (254\|?) (256\|?) (255\|?) (258\|?) (257\|?) (261\|?) (259\|?) (260\|?) (262\|?) (263\|?)
(264\|?) (265\|?) (266\|?) (268\|?) (267\|?) (270\|?) (269\|?) (272\|?) (271\|?) (274\|?) (273\|?) (275\|?) (276\|?) (278\|?) (281\|?)
(279\|?) (280\|?) (283\|?) (282\|?) (285\|?) (284\|?) (286\|?) (289\|?) (287\|?) (288\|?) (290\|?) (293\|?) (291\|?) (292\|?) (294\|?) (295\|?)
(298\|?) (296\|?) (297\|?) (299\|?) (301\|?) (300\|?) (302\|?) (304\|?) (303\|?) (306\|?) (305\|?) (309\|?) (307\|?) (308\|?) (310\|?) (313\|?)
(311\|?) (312\|?) (314\|?) (317\|?) (315\|?) (316\|?) (318\|?) (321\|?) (319\|?) (320\|?) (322\|?) (323\|?) (326\|?) (324\|?) (325\|?) (328\|?)
(327\|?) (329\|?) (361\|?) (363\|?) (362\|?) (366\|?) (364\|?) (365\|?) (367\|?) (370\|?) (368\|?) (369\|?) (372\|?) (371\|?) (374\|?) (373\|?)
(375\|?) (378\|?) (376\|?) (377\|?) (379\|?) (382\|?) (380\|?) (381\|?) (383\|?) (385\|?) (384\|?) (387\|?) (386\|?) (390\|?) (388\|?) (389\|?)
(392\|?) (391\|?) (396\|?) (397\|?) (398\|?) (399\|?) (393\|?) (394\|?) (395\|?) (399\|?) (401\|?) (400\|?) (403\|?) (402\|?) (405\|?) (404\|?) (408\|?)
(406\|?) (407\|?) (409\|?) (410\|?) (412\|?) (411\|?) (415\|?) (413\|?) (414\|?) (420\|?) (416\|?) (417\|?) (418\|?) (419\|?) (422\|?) (421\|?)
(424\|?) (423\|?) (425\|?) (429\|?) (431\|?) (433\|?)
```

```
(tdb) check summary
(13\|?) (14\|?) (23\|?) (26\|?) (24\|?) (25\|?) (27\|?) (28\|?) (29\|?) (32\|?) (31\|?) (35\|?) (33\|?) (34\|?) (36\|?) (37\|?) (38\|?)
(42\|?) (43\|?) (45\|?) (47\|?) (49\|?) (52\|?) (53\|?) (56\|?) (58\|?) (60\|?) (63\|?) (65\|?) (67\|?) (68\|?) (70\|?) (71\|?) (74\|?)
(75\|?) (76\|?) (77\|?) (80\|?) (81\|?) (84\|?) (86\|?) (87\|?) (88\|?) (89\|?) (92\|?) (95\|?) (98\|?) (100\|?) (102\|?) (103\|?) (105\|?)
(106\|?) (108\|?) (109\|?) (110\|?) (111\|?) (112\|?) (115\|?) (116\|?) (131\|?) (132\|?) (135\|?) (133\|?) (134\|?) (137\|?) (136\|?) (138\|?)
(141\|?) (139\|?) (140\|?) (142\|?) (145\|?) (143\|?) (144\|?) (148\|?) (146\|?) (147\|?) (145\|?) (150\|?) (149\|?) (151\|?) (153\|?) (152\|?) (156\|?)
(154\|?) (155\|?) (157\|?) (158\|?) (159\|?) (160\|?) (163\|?) (166\|?) (168\|?) (167\|?) (169\|?) (170\|?) (171\|?) (174\|?) (182\|?) (185\|?)
(183\|?) (184\|?) (186\|?) (189\|?) (187\|?) (188\|?) (191\|?) (190\|?) (194\|?) (192\|?) (193\|?) (195\|?) (196\|?) (197\|?) (201\|?) (202\|?)
(204\|?) (206\|?) (208\|?) (211\|?) (213\|?) (216\|?) (218\|?) (221\|?) (224\|?) (226\|?) (228\|?) (230\|?) (232\|?) (231\|?) (233\|?) (234\|?)
(238\|?) (239\|?) (240\|?) (241\|?) (242\|?) (246\|?) (247\|?) (248\|?) (250\|?) (253\|?) (255\|?) (256\|?) (257\|?) (261\|?) (264\|?) (267\|?)
(269\|?) (271\|?) (273\|?) (275\|?) (274\|?) (276\|?) (277\|?) (281\|?) (282\|?) (283\|?) (284\|?) (285\|?) (289\|?) (290\|?) (291\|?) (293\|?)
(312\|?) (314\|?) (313\|?) (317\|?) (316\|?) (318\|?) (319\|?) (320\|?) (323\|?) (321\|?) (322\|?) (324\|?) (327\|?) (325\|?) (326\|?) (327\|?)
(330\|?) (328\|?) (329\|?) (332\|?) (331\|?) (333\|?) (335\|?) (334\|?) (338\|?) (336\|?) (337\|?) (339\|?) (340\|?) (341\|?) (344\|?)
(342\|?) (343\|?) (346\|?) (345\|?) (349\|?) (347\|?) (348\|?) (350\|?) (351\|?) (352\|?) (356\|?) (357\|?) (359\|?) (361\|?) (363\|?) (366\|?)
(368\|?) (371\|?) (373\|?) (376\|?) (379\|?) (381\|?) (383\|?) (385\|?) (387\|?) (386\|?) (388\|?) (389\|?) (392\|?) (395\|?) (396\|?)
(397\|?) (401\|?) (402\|?) (403\|?) (405\|?) (408\|?) (409\|?) (410\|?) (411\|?) (412\|?) (413\|?) (417\|?) (420\|?) (423\|?) (425\|?) (427\|?) (429\|?)
(431\|?) (430\|?) (432\|?) (433\|?) (437\|?) (438\|?) (439\|?) (440\|?) (441\|?) (445\|?) (446\|?) (447\|?) (449\|?) (464\|?) (466\|?) (465\|?)
(469\|?) (467\|?) (468\|?) (471\|?) (470\|?) (472\|?) (475\|?) (473\|?) (474\|?) (476\|?) (477\|?) (478\|?) (482\|?) (480\|?) (481\|?)
(484\|?) (483\|?) (485\|?) (487\|?) (486\|?) (490\|?) (488\|?) (489\|?) (491\|?) (492\|?) (493\|?) (494\|?) (497\|?) (500\|?) (502\|?) (501\|?)
(503\|?) (504\|?) (505\|?) (508\|?) (516\|?) (519\|?) (517\|?) (518\|?) (520\|?) (523\|?) (521\|?) (522\|?) (525\|?) (527\|?) (526\|?)
```

It is worth noting that you can aggregate inputs and outputs with the same line numbers using the check summary reduce command.

check data

```
(tdb) check data [file-line]
```

where file-line is the line number in the checker report, which corresponds to the line number of final.mlir . This command gives a description of all the input and output data of the command corresponding to file-line, an example is shown below:

```
(tdb) check data 291
```

291	loc_index	owner	value name	value type	compare	index
0	205	tpu.Add	/stage2.0/attns.1/MatMul_1_output_0_MatMul	operand	✓	0
1	205	tpu.Add	/stage2.0/attns.1/Transpose_7_output_0_Transpose	operand	✓	1
2	205	tpu.Add	/stage2.0/attns.1/Add_output_0_Add	result	✗	2

```
(tdb) check data [file-line] [index]
```

Where index is the index of the data output by the check data [file-line] command. This command gives detailed information about the corresponding index data, and an example of comparing the correct data is shown below:

```
(tdb) check data 291 0
```

value-Info	
opcode	tpu.Add
core_id	0
value	{ "address": 687195987968, "layout": "continuous", "memory_type": "<14x2x56x32xsi8>", "name": "/stage2.0/attns.1/MatMul_1_output_0_MatMul", "reshape": "", "slice": "[...]", "type": "tensor<14x2x56x32x!quant.uniform<i8:f32, 0.033072317322834645>, 687195987968 : i64>" }
tiu_dma_id (before codegen)	[7959, 553]
tiu_dma_id (after codegen)	[7971, 565]
compare	✓
value type	operand

An example of comparison error data is shown below:

SoC Devices

CHAPTER 16. APPENDIX.06: TDB GUIDANCE

```
(tdb) check data 291 2
value-Info
opcode      tpu.Add
core_id      0
value        {
    "address": 687195934720,
    "layout": "continuous",
    "memory_type": "<14x2x56x32x1b>",
    "name": "/stage@.0/attns.1/Add_output_0_Add",
    "reshape": "",
    "slice": "[...,]",
    "type": "tensor<14x2x56x32x1b!quant.uniform<8:f32, 0.018248182677165353>, 687195934720 : 164>"
}
tiu_dma_id (before codegen) [7959, 555]
tiu_dma_id (after codegen) [7971, 565]
compare      x
value type   result
data-error
Not equal to tolerance cos=0.99, euc=0.9
cosine similarity: 0.089906
euclidean similarity: 0.897553
top10 diff:
x: [ 0.310219, 0.474453, 0.474453, 0.310219, 0.474453, 0.474453, 0.474453, -0.529197, 0.474453]
y: [ 0.693431, 0.20073 , 0.20073 , 0.583942, 0.20073 , 0.20073 , 0.20073 , -0.273723, 0.237226]
0:10 data:
x: [-0.127737, -0.054745, 0.86292 , -0.127737, 0.328467, -0.273723, -0.036496, 1.478103, 0.       , -0.237226]
y: [ -0.20073 , -0.018248, 0.766424, -0.01241 , 0.291971, 0.291971, -0.072993, 1.423358, 0.       , -0.237226]

Not equal to tolerance rtol=0.001, atol=0.1
Mismatched elements: 641 / 50176 (1.28%)
Max absolute difference: 0.3832182
Max relative difference: 11.000001
asm
KG2.1196032, %D564C0 = "dma.tensor"(NR8, %B7968C0) : (memref<4x2x56x32x1b, strides: [1792, 1792, 32, 1]>, none) -> (memref<4x2x56x32x1b, strides: [3584, 1792, 32, 1]>, none)
KG2.1210368, %D565C0 = "dma.tensor"(NR14, %B7971C0) : (memref<2x2x56x32x1b, strides: [1792, 1792, 32, 1]>, none) -> (memref<2x2x56x32x1b, strides: [3584, 1792, 32, 1]>, none)
=> KG2.0, %D566C0 = "dma.tensor"(KG2.1196032, %B7971C0) : (memref<14x2x56x32x1b, strides: [3584, 1792, 32, 1]>, none) -> (memref<14x2x56x32x1b, strides: [3584, 32, 64, 1]>, none)
NR4, %D567C0 = "dma.tensor"(KG2.0, %B7971C0) : (memref<1x784x4x1x1b, strides: [15016, 64, 1, 1]>, none) -> (memref<1x784x64x1x1b, strides: [1600, 64, 1, 1]>, none)
NR2, %B7972C0 = "arith.nul_satu"(NR4, %C17, %C252, %D567C0) : (round_mode = 5) : (memref<1x784x64x1x1b, strides: [1600, 64, 1, 1]>, none) -> (memref<1x784x64x1x1b, strides: [1600, 64, 1, 1]>, none)
```

When executing on SoC devices, in order to perform comparisons without introducing an mlir dependency, it is necessary to first generate a cache within a Docker environment. Subsequently, the cached model can be used for comparison in the SoC device environment.

```
$ bmodel_checker ./..../yolov5s_bm1684x_f32_tpu_outputs.npz --cache_mode generate # on TF
→ docker
$ bmodel_checker ./..../yolov5s_bm1684x_f32_tpu_outputs.npz --cache_mode offline # on soc
```

CHAPTER 17

Appendix.07: Supported Operations

17.1 List of operators currently supported by TPU-MLIR

Table 17.1: A

Onnx	Pytorch	Caffe	TOP
Abs	aten::abs	AbsVal	top.A16MatMul
Acos	aten::acos	ArgMax	top.Abs
Add	aten::adaptive_avg_pool1		top.AdaptiveAvgPool
And	aten::adaptive_avg_pool2		top.Add
ArgMax	aten::add		top.AddConst
ArgMin	aten::addmm		top.Arange
Atan	aten::arange		top.Arccos
Atanh	aten::argmax		top.Arctanh
AveragePool	aten::argmin		top.Arg
	aten::atan		top.Attention
	aten::atanh		top.AvgPool
	aten::avg_pool1d		
	aten::avg_pool2d		
	aten::avg_pool3d		

Table 17.2: B

Onnx	Pytorch	Caffe	TOP
BatchNormaliza- tion	aten::baddbmm	BatchNorm	top.BatchNorm
	aten::batch_norm	BN	top.BatchNormBwd
	aten::bmm		top.BatchNormTrain top.BinaryConstShift top.BinaryShift

Table 17.3: C

Onnx	Pytorch	Caffe	TOP
Cast	aten::cat	Concat	top.Cast
Ceil	aten::ceil	ContinuationIndi- cator	top.Ceil
Clip	aten::channel_shuffle	Convolution	top.Clip
Concat	aten::chunk	ConvolutionDepth- wise	top.Compare
Constant	aten::clamp	Crop	top.CompareConst
ConstantOfShape	aten::clone		top.Concat
Conv	aten::constant_pad_nd		top.ConstantFill
ConvTranspose	aten::contiguous		top.Conv
Correlation	aten::_convolution		top.ConvBwd_Weight
Cos	aten::_convolution_mode		top.Convbwd
CumSum	aten::copy		top.Copy
	aten::cos		top.Correlation
	aten::cosh		top.Cos
	aten::cumsum		top.Cosh
			top.Csc
			top.CumSum
			top.Custom

Table 17.4: D

Onnx	Pytorch	Caffe	TOP
DepthToSpace	aten::detach	Deconvolution	top.Deconv
DequantizeLinear	aten::div	DetectionOutput	top.DeformConv2D
Div	aten::dot	Dropout	top.DepackRaw
Dropout	aten::dropout	DummyData	top.Depth2Space top.DequantInt top.DequantizeLinear top.DetectionOutput top.Div top.DivConst top.DtypeCast

Table 17.5: E

Onnx	Pytorch	Caffe	TOP
Einsum	aten::elu	Eltwise	top.Einsum
Elu	aten::embedding	Embed	top.Elu
Equal	aten::empty		top.EmbDenseBwd
Erf	aten::eq		top.Erf
Exp	aten::erf		top.Exp
Expand	aten::exp aten::expand aten::expand_as		top.Expand

Table 17.6: F

Onnx	Pytorch	Caffe	TOP
Flatten	aten::flatten	Flatten	top.FAttention
Floor	aten::flip aten::floor	FrcnDetection	top.Flatten top.Floor
	aten::floor_divide		top.FrcnDetection
	aten::frobenius_norm		

Table 17.7: G

Onnx	Pytorch	Caffe	TOP
Gather	aten::gather		top.GELU
GatherElements	aten::ge		top.GRU
GatherND	aten::gelu		top.Gather
GELU	aten::grid_sampler		top.GatherElements
Gemm	aten::group_norm		top.GatherND
GlobalAveragePool	aten::gru		top.GridSampler
GlobalMaxPool	aten::gt		top.GroupNorm
Greater			top.GroupNormTrain
GreaterOrEqual			
GridSample			
GroupNormaliza-			
tion			
GRU			

Table 17.8: H

Onnx	Pytorch	Caffe	TOP
HardSigmoid	aten::hardsigmoid		top.HardSigmoid
HardSwish	aten::hardswish		top.HardSwish
	aten::hardtanh		

Table 17.9: I

Onnx	Pytorch	Caffe	TOP
Identity	aten::index	ImageData	top.If
If	aten::index_put_	InnerProduct	top.IndexPut
InstanceNormal-	aten::index_put	Input	top.Input
ization	aten::index_select	Interp	top.InstanceNorm
	aten::instance_norm		top.Interp

Table 17.10: L

Onnx	Pytorch	Caffe	TOP
LayerNormalization	aten::layer_norm	LRN	top.LRN
LeakyRelu	aten::leaky_relu	LSTM	top.LSTM
Less	aten::less	Lstm	top.LayerNorm
LessOrEqual	aten::linalg_norm		top.LayerNormBwd
Log	aten::linear		top.LayerNormTrain
LogSoftmax	aten::log		top.LeakyRelu
Loop	aten::log2		top.List
LRN	aten::log_sigmoid		top.Log
LSTM	aten::log_softmax		top.LogB
	aten::lstm		top.LogicalAnd
	aten::lt		top.Loop
			top.Lut

Table 17.11: M

Onnx	Pytorch	Caffe	TOP
MatMul	aten::masked_fill	MatMul	top.MaskRCNN_BboxPooler
Max	aten::matmul	Mish	top.MaskRCNN_GetBboxB
MaxPool	aten::max		top.MaskRCNN_MaskPooler
Min	aten::max_pool1d		top.MaskRCNN_RPNGetBboxes
Mod	aten::max_pool2d		top.MaskedFill
Mul	aten::max_pool2d_with_		top.MatMul
	aten::max_pool3d		top.MatchTemplate
	aten::mean		top.Max
	aten::meshgrid		top.MaxConst
	aten::min		top.MaxPool
	aten::mish		top.MaxPoolWithMask
	aten::mm		top.MaxPoolingIndicesBwd
	aten::mul		top.MaxUnpool
	aten::mv		top.MeanRstd
			top.MeanStdScale
			top.MeshGrid
			top.Min
			top.MinConst
			top.Mish
			top.Mmap2Rgbmap
			top.Mod
			top.Mul
			top.MulConst

Table 17.12: N

Onnx	Pytorch	Caffe	TOP
Neg	aten::native_batch_norm	Normalize	top.Nms
NonMaxSuppression	aten::native_group_norm		top.NonZero
NonZero	aten::native_layer_norm		top.None
Not	aten::ne		top.Normalize
	aten::neg		
	aten::new_full		
	aten::new_ones		
	aten::new_zeros		
	aten::nonzero		

Table 17.13: O

Onnx	Pytorch	Caffe	TOP
OneHot	aten::ones		
Or	aten::ones_like		

Table 17.14: P

Onnx	Pytorch	Caffe	TOP
Pad	aten::pad	Padding	top.PReLU
PixelNormalization	aten::permute	Permute	top.Pack
Pow	aten::pixel_shuffle	Pooling	top.Pad
PReLU	aten::pixel_unshuffle	Power	top.Permute
	aten::pow	PReLU	top.PixelNorm
	aten::prelu	PriorBox	top.PoolMask
		Proposal	top.Pow
			top.Pow2
			top.Pow3
			top.Preprocess
			top.PriorBox
			top.Proposal

Table 17.15: Q

Onnx	Pytorch	Caffe	TOP
QuantizeLinear			top.QuantizeLinear

Table 17.16: R

Onnx	Pytorch	Caffe	TOP
RandomNormalLike	aten::reflection_pad1d	Reduction	top.RMSNorm
Range	aten::reflection_pad2d	ReLU	top.ROIPooling
Reciprocal	aten::relu	ReLU6	top.RandnLike
ReduceL1	aten::remainder	Reorg	top.Range
ReduceL2	aten::repeat	Reshape	top.Reciprocal
ReduceLogSumExp	aten::replication_pad1d	RetinaFaceDetection	top.Reduce
ReduceMax	aten::replication_pad2d	Reverse	top.Relu
ReduceMean	aten::reshape	ROIPooling	top.Remainder
ReduceMin	aten::roll		top.Repeat
ReduceProd	aten::rsqrt		top.RequantFp
ReduceSum	aten::rsub		top.RequantInt
Relu			top.Reshape
Reshape			top.RetinaFaceDetection
Resize			top.Reverse
ReverseSequence			top.RoiAlign
RoiAlign			top.RoiExtractor
Round			top.Rope
			top.Round
			top.Rsqrt

Table 17.17: S

Onnx	Pytorch	Caffe	TOP
ScatterElements	aten::scaled_dot_product	Scale	top.Scale
ScatterND	aten::scatter	ShuffleChannel	top.ScaleDotProductAttention
Shape	aten::select	Sigmoid	top.ScaleLut
Sigmoid	aten::sigmoid	Silence	top.ScatterElements
Sign	aten::sign	Slice	top.ScatterND
Sin	aten::silu	Softmax	top.Shape
Slice	aten::sin	Split	top.ShuffleChannel
Softmax	aten::sinh		top.SiLU
Softplus	aten::size		top.Sigmoid
SpaceToDepth	aten::slice		top.Sign
Split	aten::softmax		top.Sin
Sqrt	aten::_softmax		top.Sinh
Squeeze	aten::softplus		top.Size
Sub	aten::sort		top.Slice
Sum	aten::split		top.SliceAxis
	aten::split_with_sizes		top.Softmax
	aten::sqrt		top.SoftmaxBwd
	aten::squeeze		top.Softplus
	aten::stack		top.Softsign
	aten::sub		top.Sort
	aten::sum		top.Split
			top.Sqrt
			top.Squeeze
			top.StridedSlice
			top.Sub
			top.SubConst
			top.SwapChannel
			top.SwapDimInner
			top.Swish

Table 17.18: T

Onnx	Pytorch	Caffe	TOP
Tanh	aten::t	TanH	top.Tan
Tile	aten::tan	Tile	top.Tanh
TopK	aten::tanh		top.Tile
Transpose	aten::tile		top.TopK
Trilu	aten::to		top.Transpose
	aten::_to_copy		top.Trilu
	aten::topk		top.Tuple
	aten::transpose		
	aten::type_as		

Table 17.19: U

Onnx	Pytorch	Caffe	TOP
Unsqueeze	aten::unbind	Upsample	top.UnTuple
Upsample	aten::_unsafe_view aten::unsqueeze aten::upsample_bilinear2d aten::upsample_linear1d aten::upsample_nearest1d aten::upsample_nearest2d aten::upsample_nearest3d		top.Unpack top.Unsqueeze top.Upsample

Table 17.20: V

Onnx	Pytorch	Caffe	TOP
	aten::view		top.Variance top.View

Table 17.21: W

Onnx	Pytorch	Caffe	TOP
Where	aten::where		top.Weight top.WeightReorder top.Where

Table 17.22: X

Onnx	Pytorch	Caffe	TOP
Xor			

Table 17.23: Y

Onnx	Pytorch	Caffe	TOP
		YoloDetection	top.Yield top.YoloDetection top.Yuv2rgbFormula

Table 17.24: Z

Onnx	Pytorch	Caffe	TOP
	aten::zeros aten::zeros_like		