ESWIN

ENNP USER GUIDE

EIC7x Series AI SoC

Engineering Draft / Rev1. 3

Feb. 26, 2025

Notice

Revision History

Document Version	date	illustrate
v0.3	2024/02/22	Initial release
v0.4	2024/03/25	Modify NPU Run time summary description and call graph, etc.
v0.5	2024/07/17	Modified the chapter order and the description of Es AAC and Es Quant tools.
v0.6	2024/08/09	Improve the instructions for using Docker.
v0.7	2024/08/28	Added example of exporting ONNX model from GitHub.
v0.8	2024/09/10	Improve the yolo v3 example description.
v0.9	2024/10/12	Added instructions for implementing dynamic batch size and composite models in NPU Runtime.
v0.9.2	2024/10/22	Revised the description of NPU Runtime to implement dynamic batch size and composite models.
v1.0	2024/11/18	Added instructions for using custom preprocessing functions in user quantization process.
v1.1	2024/12/26	Revisions: 1. Added description of E sAAC hybrid model generation method; 2. Add user-defined DSP operator development examples; 3. Improved the list of operators supported by DSP. Impact: This version is updated. The compilation instructions of the hybrid model are added in Section 3.2; the content of custom DSP operator development is added in Chapter 8; and the newly supported DSP operators are added in the appendix of Chapter 9. There are no additions or deletions to the other chapters, which will not affect the user's previous usage.
v1.2	2025/01/06	Revisions: 1. Added E sAAC for vit Description of the model's compilation option --enable-uosp. Impact: This version is updated, adding compilation option descriptions in Section 3.1, and no additions or deletions to other sections. When building a model, please add the compilation option --enable-uosp ; in other scenarios, users can continue to use it as before.
v1.3	2025/01/21	Added description of EsAAC compilation option --lut dsp .

Table of contents

Table of Contents

Figure Catalog

1. Overview

1.1 ENNP Introduction

computing heterogeneous acceleration platform for ESWIN media processing chips. Developers can call the hardware acceleration module inside EIC7700 based on the API interface provided by ENNP to implement the reasoning of neural network models, image transformation processing and other custom functions that require hardware acceleration. The ENNP platform can provide reasoning based on hardware acceleration models such as HAE, GPU, DSP, NPU, etc., to realize application scenarios such as target recognition, object detection, and image classification.

ENNP includes an offline development tool kit and a runtime software framework. Development based on ENNP is divided into offline development and online development.

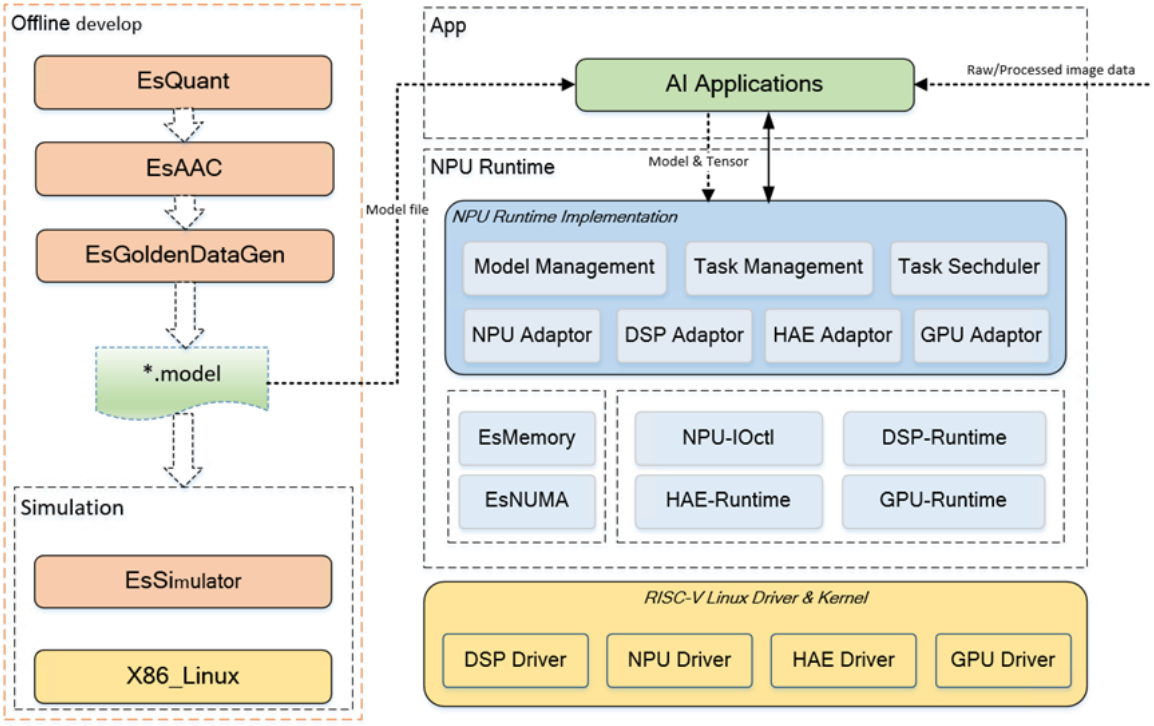

Figure 1-1 ENNP software stack diagram

Figure 1-1includes EsQuant quantization tools, EsAAC model compilation tools, EsGoldenDataGen reference data generation tools, and EsSimulator simulation verification tools. These belong to the ENNP offline development tool suite. Using the above tool suite, the algorithm model trained by the artificial intelligence framework (TensorFlow, PyTorch, Caffe, and ONNX, etc.) can be converted into a unified intermediate expression (IR) and generate an offline model, while providing end-to-end model optimization, offline model generation and verification, etc. For detailed information on each suite tool, please refer to the corresponding chapter.

After the model is generated, you can write runtime software based on ESSDK, call the NPU Runtime interface, and use the NPU subsystem hardware (NPU, DSP, HAE, GPU, etc.) for reasoning to implement image classification, target detection, image segmentation, natural language processing and other functions based on chip hardware acceleration.

1.2 Runtime framework

NPU Runtime, also referred to below, is a runtime system for loading NN models provided by ENNP. It can directly load offline models compiled based on ENNP tools and perform inference to complete functions such as target recognition and image classification. Users develop intelligent analysis

solutions based on NPU Runtime, and the compiler automatically analyzes and optimizes the execution process, and generates optimized offline models, maximizing the reuse of NPU, DSP, HAE, GPU and other hardware, improving hardware utilization and optimizing system power consumption.

NPU Runtime is an API interface that ENNP uses to provide model reasoning capabilities to user applications. Users can directly call the rich APIs of NPU Runtime to implement reasoning scenarios for actual applications. Currently, NPU Runtime supports various network models generated by the ENNP platform model compiler EsAAC (see: EsAAC User Guidefor details). NPU Runtime supports HAE hardware pre-processing (Resize, Normalization, CVT), supports DSP rich and high-performance operators (see: Appendix for a list of operators appendix, and supports ESWIN's NPU hardware acceleration unit.

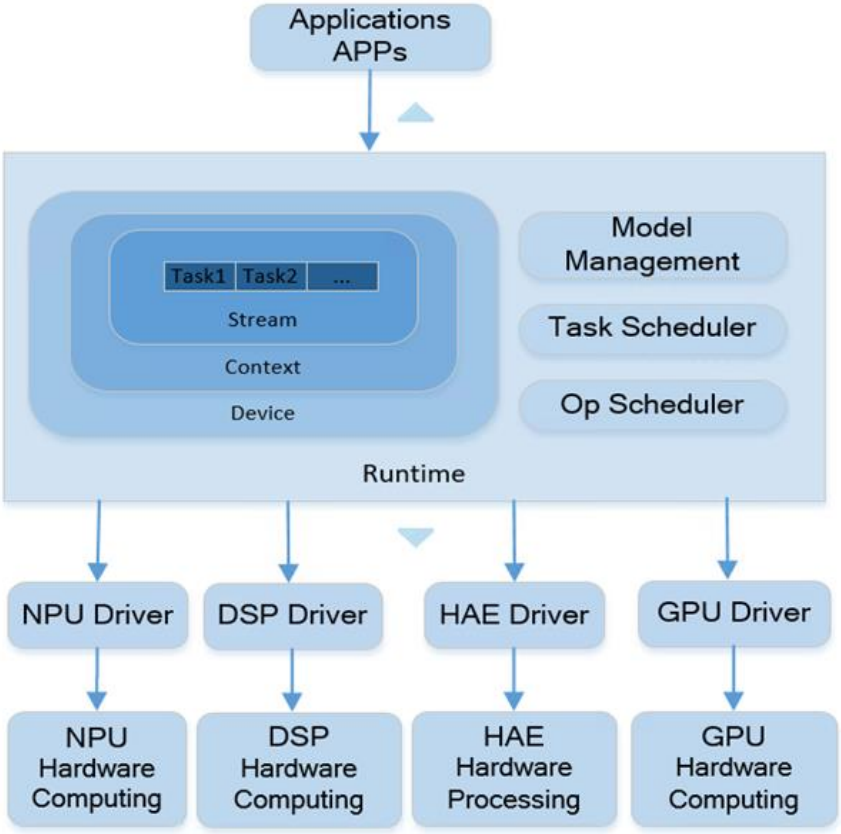

Figure 1-2 NPU Runtime Block Diagram

- Model inference interface: supports the inference of multiple models on different hardware acceleration units, and coordinates the execution order of multiple tasks through the task scheduler.
  - Device: represents an NPU inference hardware execution unit. Tasks on different devices are executed in parallel.
  - Context: describes a configuration context for NPU operation, which is convenient for managing the corresponding memory allocation and hardware resources; at the same time, it acts as a container to manage the life cycle of objects created on the corresponding Context, and is used to manage asynchronously submitted tasks. A Context can be associated with multiple threads of a process, and the streams of different Contexts are completely isolated. Tasks submitted based on the same Context will be scheduled in order. Call the ES_NPU_CreateContext interface in a process or thread to create a Context.
  - Stream: It is used to maintain the queue waiting for the user to retrieve after the asynchronous operation task is completed. Call the ES_NPU_CreateStream interface in the process or thread to create a Stream. Stream belongs to Context.
  - Task: The execution carrier of a task, which contains the necessary information required to execute a task.
- Reasoning acceleration hardware: You can use NPU, DSP, HAE, GPU and other hardware to

implement the corresponding acceleration algorithm. After the NN model is quantized and compiled using ENNP, it can be directly loaded into the runtime, and the framework software and firmware will automatically and efficiently schedule it to each hardware unit for execution. You can also call the corresponding operator separately through AcceleratorKit to perform a single calculation.

1.3 Software Development Process

The development process based on ENNP can be divided into two parts: offline tool use and online development verification.

1.3.1 Using Offline Tools

Relevant tools include: quantification tool EsQuant, model compilation tool EsAAC, reference data generation tool EsGoldenDataGen, and simulation verification tool EsSimulator. The development steps are as follows:

- **ENNP tool environment configuration**: Follow the guide to install the tool suite required for ENNP on the PC.
- **Model acquisition**: Use the PyTorch framework to build and train the neural network, and then export the model to ONNX format (.onnx file) for subsequent processing.
- **Acquisition of quantification parameters**: EsQuant generates quantification parameters based on ONNX files and quantification configuration files, and saves them in json format.
- **Model generation**: The EsAAC tool compiles and generates. model files based on ONNX files and quantified parameter json files.
- **Simulation verification**: Use EsGoldenDataGen to generate reference data based on ONNX files, and conduct simulation verification through EsSimulator to ensure that the model accuracy meets expectations.

1.3.2 Online Development Verification

- **C language interface programming:** Based on the C interface API provided by ESSDK, write adapted C language code, and modify the CMakeLists.txt file to adapt to the compilation process.
- **Execution verification:** Use a cross-compiler to compile the above code and generate an executable program. Copy this program to the EIC7700 development board and execute it to verify the actual running effect and model performance of the program.

Through the above steps, developers can efficiently develop, optimize, compile, and verify the model, and finally deploy and run it on the target hardware device, thereby completing the entire development process from model design to actual application.

1.4 Development Environment

For details about operating system versions and SDK versions, please refer to Chapter 6.

1.5 Related Documents

Related documents that ENNP development relies on, including the ENNP Developer's Manual.

2. EsQuant User Guide

2.1 EsQuant Tool Introduction

EsQuant tool provides a Python interface for users to use. The advantage of using the Python interface to quantify the model is that users can customize the preprocessing function. At the same time, the preprocessing functions of ImageNet and YOLO are provided in advance to facilitate the quantization of classification models and detection models.

EsQuant can easily complete the following tasks:

- Quantization function: supports quantizing floating-point models into fixed-point models, supports symmetric quantization, and supports mixed-precision quantization at different layers;
- Quantization accuracy analysis: This function gives the cosine similarity between the inference results of each layer of the model after quantization and the floating-point inference results, including cumulative error analysis and reset error analysis. The cumulative error analysis can be used to analyze how the quantization error occurs, providing ideas for improving the accuracy of the quantization model.

2.2 Model Quantification Description

2.2.1 Quantization Profile Description

The configuration file is a json file. Taking the yolov3 model as an example, the format is as follows:

```
eswin@debian:~# cat config.json
{
"version": "1.3",
"model": {
"model_path": "/yolov3.onnx",
"save_path": "/home/yolov3/",
"images_list": "/home/yolov3/cal_lists.txt",
"analysis_list": "/home/yolov3/file_list_1.txt"
},
"quant": {
"quantized_method": "per_channel",
        "quantized_dtype": "int8",
        "requant_mode": "mean",
        "quantized_algorithm": "AT_EIC",
        "optimization_option": "auto",
        "bias_option": "absmax",
        "nodes_option1": [],
        "nodes_option2": [],
        "nodes_i8": [],
        "nodes_i16": ["conv1"],
        "mean": [
            0,0,0
        ],
        "std": [
            1,1,1
        ],
        "norm": true,
        "scale_path": "",
        "enable_analyse": true,
        "device": "cpu",
        "output_variables_dtype": {
            "326": "int16",
            "392": "int16",
            "458": "int16"
        }
    },
    "preprocess": {
        "input_format": "RGB",
        "keep_ratio":false,
"resize_shape": [
416,416
],
"crop_shape": [
416,
416
]
}
}
```

Parameter description is shown in Table Table 2-1.

Table 2-1 Configuration file parameter description

Configuration	Example	illustrate
model_path (Required parameter)	/path/resnet.onnx	The model path that needs to be quantized. Currently, the supported model is ONNX model. If not configured, an error will be reported and the program will be terminated.
save_path (Required parameter)	/path/save/	The save path of intermediate data (including table.json) and quantitative accuracy analysis results. The quantitative accuracy analysis results are in txt files. If not configured, an error will be reported and the program will be terminated.
images_list (Required parameter)	/path/image_list.txt	txt file path. When calculating the table, it is necessary to count the distribution of data of multiple graphs in the network reasoning process. The txt file contains the paths of all the images that need to be calculated. For example, if the path example/ contains 1,000 jpg images, all the images in the path will be written to the txt file. It is recommended that the detection and classification models provide 1,000 data.
analysis_list (Required parameter)	/path/image_list_2.txt	txt file path for accuracy analysis.
scale_path	/path/save/scale.json	1. The saving path of each layer tensor statistical information, which can be reused and saved in JSON format; 2. If not configured, the scale.json file will be automatically added under the save_path path. 3. Process: When quantize is run for the first time, the network tensor is statistically analyzed to obtain the original scale and saved in json. During the precision analysis phase, when the quantization method is adjusted through config.json or a layer data type is switched, scale.json under scale_path will be automatically read, and statistical information will not be repeated to reduce quantization time; 4. The scale.json file in the scale_path path must correspond to the model, otherwise an error will be reported and the program will be terminated.
quantized_method	per_channel	The configurable values are: per_channel, per_layer, the default is per_channel, and currently only per_channel is supported. When quantizing data of different channels (such as weight and bias of conv), calculate the scale (per_channel) of each channel separately and then quantize them separately, or calculate the distribution of all channels and quantize them uniformly (per_layer) .
quantized_dtype	int8	Configurable values are: int8, int16, default is

Configuration	Example	illustrate
		int8. The global calculation accuracy selected when quantizing the network, that is, the data type selected when the network layer is calculated when it is not specially marked by nodes_i8 or nodes_i16. Note: When setting it to int16 , you need to manually change the datatype of input and output in table.json to int16
requant_mode	max	The configurable values are: max, mean, the default is max. The scale of the input and output tensors of each network layer is counted so that the tensors can be quantized. However, the scale of each layer output is often different, and the quantized integer multiplication and division of each layer's scale needs to be transformed to achieve identity, which is achieved by inserting requant (insert); but this requires additional computational overhead, and the additional multiplication can be avoided by unifying the scales of several adjacent network layers. The methods of unifying the scales include taking the mean (mean) and taking the maximum value (max) .
quantized_algorithm	at_eic	The quantization method used when calculating the quantization parameters of each layer. The configurable values are: at_eic (adaptive threshold), mse_eic , the default is at_eic . When calculating the table, it is necessary to count the numerical distribution of the input and output tensors of each layer, exclude outliers to determine the valid numerical range, and determine the scale based on the numerical range. It is recommended to use mse_eic for the Yolov 5 model , and at_eic works better for most classification models.
optimization_option	auto	Model optimization level. The configurable values are: option1, option2, auto. The default value is auto. The order of operator arrangement and fusion affects the calculation accuracy and speed. In most cases, option 2 has higher calculation speed and accuracy, but option 1 is still better for some network layers. auto will automatically adjust the optimization strategy according to the network structure .
bias_option	absmax	Statistics bias range mode. Configurable values are: absmax, max, default is absmax. absmax works better in most models.
enable_analyse	true	Whether to enable precision analysis, if enabled, true, otherwise false, enabled by default

Configuration	Example	illustrate
device	cuda	String, fill in "cpu" if using cpu, fill in "cuda" if using gpu , gpu is used by default.
nodes_option1	["conv_1"]	For a global option2 or auto quantized network, you can specify a specific conv node as option1 quantized.
nodes_option2	["conv_1"]	For a network with global option1 or auto quantization, you can specify a specific conv node as option2 quantization .
nodes_i8	["conv_1"]	For a global int16 quantized network, some layers can still maintain high computational accuracy even if they are replaced with int8 calculations. You can specify them as int8 quantization to speed up the calculation.
nodes_i16	["conv_1"]	For the global int8 quantized network, the calculation accuracy of some layers is greatly reduced after quantization. In order to improve the calculation accuracy, it can be specified as int16 quantization, especially for the last conv layer of the multi-output detection model. In order to obtain the maximum accuracy, manually fill in the corresponding node-name. Notice: 1. If the same tensor is shared by different branches, do not change the data type of the first layer of the branch .
output_variables_dtype	{"326": "int16","392": "int16", "458": "int16"}	Dictionary data type, the keyword is the output tensor name, the value is the data type. Generally, only the target detection model needs to be configured. Yolov 5 recommends configuring it to int 16 to ensure detection accuracy.
mean (Required parameter)	[0.485, 0.456, 0.406]	divided by 255) to be subtracted from the values of each channel during preprocessing. The parameter format is a list. Even if the user uses a custom preprocessing function, it needs to be configured in config. When normalizing, configure Min; When customers customize the preprocessing function, please note that the mean parameter set in the configuration file must be consistent with the preprocessing parameter.
std (required parameter)	[0.229, 0.224, 0.225]	divided by 255) by which the values of each channel need to be divided in preprocessing. The parameter format is a list. Even if the user uses a custom preprocessing function, it needs to be configured in config. When normalizing, configure 1/(Max-Min); When customers customize the preprocessing function, please note that the variance parameters set in the configuration file must be consistent with the preprocessing parameters.
input_format	BGR	Configurable values are: RGB, BGR, the order of

Configuration	Example	illustrate
(Required parameter)		color layers of the input image required by the network .
keep_ratio (Required parameter)	true	Boolean type parameter. If it is true, the aspect ratio is maintained during the resizing process, otherwise it is not maintained .
norm (Required parameter)	t rue	Whether to perform normalization before preprocessing, that is, divide the input by 255 first; the default is true. When it is true, the mean variance value must be between 0 and 1 .
channel_first	true	Set it according to the input ONNX layout. If the input layout is NHWC, set it to false in the configuration file. The default preprocessing stage is used for executable programs, and no setting is required for Python client-defined preprocessing.
resize_shape (Required parameter)	[1024,2048]	The height and width dimensions of the image to be scaled. When keep_ratio is true, resize_shape is required to be a 1-dimensional vector, and when it is false, it is a 2-dimensional vector .
crop_shape (Required parameter)	[1024,2048]	The length and width of the image are currently captured from the center. In the Python interface, resize_shape needs to be larger than crop_shape .

2.2.2 Quantization command description

The quantization commands are as follows. For specific environment configuration, see 6.2.1.

```
eswin@debian:~# python Example_ with_config.py --config_path /config.json --preprocess_name
Yolo
```

In the above command, the --preprocess_name parameter is the preprocessing function selected by the user. Es Quant pre-provides preprocessing functions for Yolo, ImagenetOpencv, and ImagenetTorch. For classification models, there are two types: ImagenetOpencv and ImagenetTorch. ImagenetOpencv is implemented using opencv, while ImagenetTorch is implemented using the standard pytorch method. The mean, variance, and shape are adjusted as needed based on the user's model and preprocessing method. The Yolo preprocessing function is applicable to all officially provided Yolo detection networks. For other image tasks, customers can customize preprocessing functions. They only need to complete the preprocessing function to use it. The command example is shown below.

```
eswin@debian:~# python Example_with_config.py --config_path /config.json --preprocess_name
ImagenetOpencv_custom --is_custom_preproc=True --custom_preproc_path=
datasets/preprocess_custom.py
```

The --is_custom_preproc parameter specifies whether to use a custom preprocessing function, and --custom_preproc_path indicates the path of the Python file where the custom preprocessing function is implemented. For a custom preprocessing function, its implementation class needs to be defined in the corresponding Python file, and the corresponding __init___ and __call__ methods need to be completed. In addition, a dictionary named preprocess_registry needs to be defined. The key of the element in the dictionary is the name of the preprocessing function entered in the --preprocess_name parameter, and the value is the class name of the corresponding preprocessing function implemented in the Python file.

2.3 Quantitative Accuracy Analysis Description

This function mainly performs floating-point reasoning and quantitative reasoning and generates data for each layer. It then compares the cosine similarity of each layer of data between the two and performs quantitative accuracy analysis based on the cosine similarity results.

The precision analysis will generate two txt files. The information in precision_accumulate_result.txt is the cosine similarity error accumulated layer by layer for the quantized model and the floating-point model; the information in precision_reset_result.txt is the cosine similarity error obtained by resetting the input for each layer of the quantized model and the floating-point model. The cosine similarity information is recorded in the file.

The details are as follows:

```
eswin@debian:~# cat precision_reset_result.tx t
conv2_1 : 0.999249001344045
relu2_1 : 0.999094545841217
conv2_2 : 0.9989643030696445
relu2_2 : 0.9991797871059842
pool2 : 0.9991780718167623
```

First, check the last op - name in precision_accumulate_result.txt (the last layer operation in the network structure). Is the cos_dist(mean) corresponding to the name greater than 95%? If it is greater than the threshold, the error is considered to be within an acceptable range. If not, you need to check another file, precision_reset_result.txt. At this time, you need to locate which layer has a significant decrease in precision loss, and then replace this layer with int16 in the configuration file config.json (do not change the datatype of the first layer operation and the last output operation). The operation is to find nodes_int16 in the configuration file and fill in the node-name field. Re-quantize and conduct precision analysis, and then check the quantization precision error. After the process is completed, a higher precision can generally be obtained. If the precision is still lower than the threshold, you need to check whether the configuration parameters are correct.

If the customer uses a custom preprocessing function, first check the input_format parameter to see if the color channel of the customer's custom preprocessing (customer-defined preprocessing function) is consistent with the actual input of the model; secondly, check the input layout. EsQuant accepts the NCHW layout by default. It is necessary to check whether the custom preprocessing input is consistent with the model requirements. If the setting is wrong, it will lead to extremely low cumulative errors.

If the accuracy still does not meet the requirements, consider whether the optimization_option and quantized_method are configured according to the recommended settings. It is recommended to use the default optimal quantization method first.

When the final accuracy result meets the requirements, the table .json (the actual quantization table name is composed of the onnx file name and the parent directory) file generated by the quantization tool and the model compilation tool can be used to generate a model that runs on the hardware.

2.4 EsGoldenDataGen Tool Introduction

The EsGoldenDataGen tool uses the open source framework ONNXR untime to perform floating-point inference on the ONNX model, generating binary data files of input and matching output, which are used to verify the quantization model generated by EsACC and the accuracy and correctness of its compilation and generation process and quantization process. Note that this process is only used to verify the quantization process, and the data provided must be calibration data.

How to use EsGoldenDataGen:

```
eswin@debian:~# python -m esquant.es_goldendata_gen.EsGoldenDataManager --config config_cls.json
```

The configuration file is a json file with the following format:

```
eswin@debian:~# cat config_cls.json
{
"model": {
"model_path": #Model file path
"/yolov3_sim_extract_416_notranspose_noreshape.onnx",
"save_path": "/output/", # Save input and output file paths
"batchsize": 4 # batchsize
},
"dataset": {
"data_root": "/image/", # Image file path
"transform_cfgs": [
{
"type": "ToRGB" # Preprocessing parameter configuration; write CustomPreProcess by default; preprocessing provided by default includes Imagenet, etc.
},
{
"type": "Resize",
"shape": 256
},
{
"type": "CenterCrop",
        "crop_size": 224
      },
      {
        "type": "Normalize",
        "mean": [123.675, 116.28, 103.53],
        "std": [58.394161, 57.120009, 57.375638],
        "norm": false
      },
      {
        "type": "ToCHW"
      },
      {
        "type": "TransPrecision",
        "precision": "float32"
      }
    ]
  },
  "quant": {                    # 量化相关参数,
"input": { # Input related parameters
"format": {
"input1": [1,3,224,224, 1 ], # Keep consistent with EsAAC input
        "input2": [1,3,224,224,1] #consistent with EsAAC input
},
" color_format_convert ": true, # Indicates whether to adjust the color channel order of the dump data
"scale": {
"input1": 0.02042430216871847 , # Input 1 corresponds to the step value of int 8 or int 16 in the table
"input2": 0.02042430216871847 , # Input 1 corresponds to the step value of int 8 or int 16 in the table
},
"data_type": 0, # The data type to be converted: 0 represents int8, 1 represents int16
```

```
"channel_first": true
},
"output": {
"format": {
"output1": [1,255,20,20 ,1 ] #Supported format is NCHW
"output2": [1,255,40,40,1 ] # Supported formats: NCHW
},
"scale": {
"output1": 0.1886681467294693 # Output 1 corresponds to the step value of int 8 or int 16 in the table
"output2": 0.1886681467294693 # Output 2 corresponds to the step value of int 8 or int 16 in the table
},
"data_type": 0 # The data type to be converted: 0 represents int8, 1 represents int16
}
}
}
```

- For multiple inputs and multiple outputs, add the corresponding tensor-name and related layout and scale information at the end.
- The shape information needs to be consistent with the input shape of EsAAC. For example, if the input is NHWC, the input data is adjusted to [1,1,224,224,3]. If the input is NCHW, the input data information is adjusted to [1,3,224,224,1].
- For multi -batch models, the N in shape needs to be changed to the number of batches. For example, when batch=6, shape=[6,255,20,20,1], and the provided ONNX model must also be multi-batch.
- The saved data is a binary file in the specified path. The file name consists of the tensor name and the image name, connected by the "_" character in the middle.
- For customized preprocessing, the customer needs to implement the custom_preprocess(img) function in esquant/es_goldendata_gen/CustomPreProcess.py and change the type of transform_cfgs to Custom in the configuration file.
- Scale information is obtained from the table generated by quantization, that is, 2.2The output datatype must be consistent with the quantization model. For detection tasks, the recommended datatype is 1 (int16).
- color_format_convert indicates whether to adjust the color channel order of the model input during the dump data stage. If the model input is bgr, but you want to get rgb data, use true, otherwise fill in false, and the same applies to other situations.

3. EsAAC User Guide

3.1 EsAAC Tool Introduction

EsAAC is ESWIN's deep learning compiler for self-developed chips. It can convert network models of mainstream artificial intelligence frameworks (TensorFlow, Pytorch, Caffe, ONNX, etc.) into a unified intermediate representation (IR), and optimize the model based on EIC7700 hardware to generate an offline model that can be loaded and run by EIC7700, which integrates conversion, optimization, and compilation. EsAAC is applicable to network models including image classification, object detection, image segmentation, etc.

Specific supported features include:

• Supports optimization and compilation of models in the fields of image classification, object detection, image segmentation, etc. Supports input network model formats: ONNX

• Supports compilation of multiple batch models, with a maximum batch size of 16 (the specific number depends on the actual model). Offline models only support static multiple batches.

• Supports the use of different heterogeneous computing units and sets the maximum resource usage of each computing unit;

• Supports cache optimization using on-chip high-speed storage SRAM;

• Supports user-customized pre-processing or post-processing calculations (including color conversion and normalization of input images, and output non-maximum suppression and other calculations).

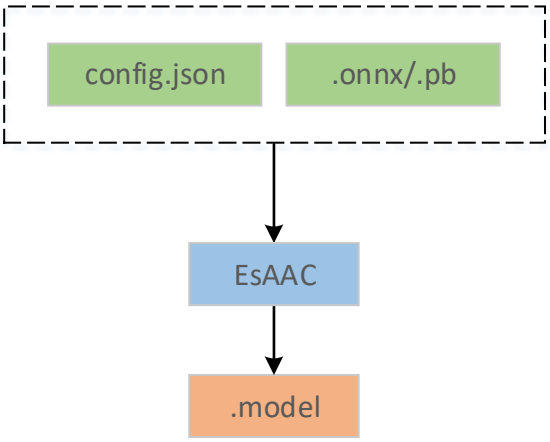

Figure 3-1EsAAC compilation process

The main compilation process of EsAAC is shown in Figure Figure 3-1is mainly divided into the following steps:
EsAAC reads the trained model file input by the user and starts compiling according to the configuration file and command line parameters.
Users can choose to configure the available amount of SRAM to use SRAM to cache intermediate results. EsAAC uses an efficient algorithm to schedule and allocate SRAM space to improve memory access efficiency.
EsAAC parses the trained ONNX model and maps the computing configuration to the hardware unit to generate an offline model that can be run by the Runtime. This model is a serialized binary file named xxx.model.

3.2 Generate Model Function Introduction

The EsAAC tool provides an executable file named EsAAC, which is used to convert the ONNX format model into a .model format file that can be executed on the hardware.

surface 3-1 Command line parameter description

parameter	illustrate	default value	Required
--input-model	Import model file	none	yes
--input-nodes	Model input nodes	none	no
--input-shapes	The shape of each node of the model input. If this parameter is configured, input-nodes must be configured	Each node of the model has its own shape	no
--output-nodes	Output nodes of the model	none	no
--output-shapes	The model outputs the shape of each node. If this parameter is configured, output-nodes must be configured	none	no
--quant-stats	Quantitative statistics file	none	no
--equant-config-path	Quantization configuration file, configure quantization related parameters (refer to other chapters)	none	no
--loadable-name	The name of the output model file	Enter the name of the model file.model	no
--run-weight-csc	Whether to perform csc on weight Compression	no	no
--sram-capacity	SRAM space size , unit KB	4 096	no
--dsp-core	Specify the number of DSPs , 1~4	1	no
--inference-datatype	The data type of the model to be generated. When using a quantization file or configuration file, this parameter has no effect and can be left unconfigured. The optional values are DT_QINT8 and DT_QINT16	DT_QINT8	no
--reverse-input-channel	Whether to switch the channel position of the input model	no	no
- -input-format	Set the input shape format to N CHW or N HWC	NCHW	no
--convert -to-qm	Set whether to convert the input model into q m format for use by EsQuant	no	no
- -run-device	Specifies the type of inference hardware used to generate the model. This option is optional. The optional values are n pu , dsp and a ll	npu	no
- -enable - uosp	This option needs to be added when compiling the vit model for shape transformation, but it does not need to be enabled for the large language model.	F alse	no

- - lut dsp	Set whether to use the DSP's l ut operator.	F alse	no

Note: Currently, the following throughput models are supported by the " --run -device all" command: [Inception_V1, inception-v3, inception-v4, resnet18, resnet50, segnet, squeezenet_v1.0, squeezenet_v1.1, mobilenet_v1, mobilenet_v2, densenet121, densenet201, yolov2, yolov3, senet50, yolov5s] , and the file name passed in by "--input-model" needs to be consistent with the above name, such as: "--input-model resnet50.onnx" .

3.3 EsSimulator Tool Introduction

EsSimulator is an offline model testing tool that is used to compare whether the generated model is consistent with the original model calculation results. EsSimulator receives the offline model generated by EsACC and verifies the correctness of the model based on the input and expected output Tensor data generated by EsGoldenDataGen. EsSimulator is an offline verification tool that can quickly verify the correctness of the model, facilitate debugging and locating model problems, and evaluate the model effect.

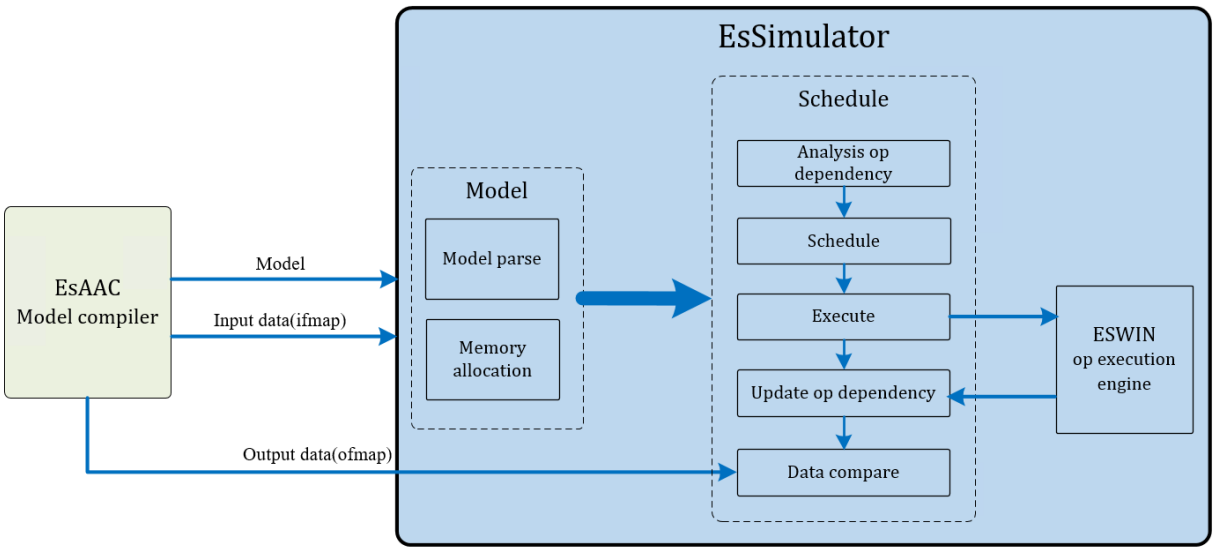

Figure 3-2 EsSimulator Execution Architecture

The usage of EsSimulator can be displayed via --help.

```
eswin@debian:~# ./EsSimulator –help
Usage:
EsSimulator --model=<xx.model> --input=<ifmap.bin> --output=<ofmap.bin> [--tolerance=xx, default:0.05] [--dsp_op_dir=xx, default:./dsp_kernels]
EsSimulator --model=<xx.model> --input=<ifmap1.bin,ifmap2.bin,ifmap3.bin> --output=<ofmap1.bin,ofmap2.bin,ofmap3.bin> [--tolerance=xx, default:0.05] [--dsp_op_dir=xx, default:./dsp_kernels]
```

The EsSimulator parameters are described as follows:

```
--model is a required parameter, which specifies the input model file
--input is a required parameter, which specifies the input ifmap. Multiple ifmaps are supported.
--output is a required parameter, which specifies the model output file ofmap. Multiple ofmaps are supported.
--tolerance is an optional parameter that specifies the error tolerance. The default value is 0.009. It is valid when the model data is float 32 or float 16.
```

4. NPU Runtime Development Guide

NPU Runtime is a runtime system provided by ENNP for loading NN models. It can directly load offline models that are quantized and compiled using ENNP tools to implement functions such as target recognition and image classification. Users can develop intelligent analysis solutions based on NPU Runtime, where the compiler automatically analyzes and optimizes the execution process and generates optimized offline models. This maximizes the reuse of NPU hardware, improves hardware utilization, and optimizes system power consumption. The architecture block diagram of NPU Runtime is referenced in Figure 1-2.

4.1 Input model requirements

The model input to NPU Runtime must be a network model generated by the ESWIN EsAAC model compiler. After the model is generated, it is recommended to use the ESWIN EsGoldenDataGen / EsSimulator tools to verify the correctness and validity of the model. For specific models supported by NPU Runtime, please refer to the EsAAC User Guide.

4.2 Development Process

4.2.1 NPU Runtime API Invocation

Users can call the NPU Runtime APIs based on their specific use cases to achieve the corresponding functionalities. The latest version of NPU Runtime introduces dynamic batch size and composite model features, enhancing the flexibility of application development and the inference performance of NPU Runtime.

- The dynamic batch size feature allows users to input any number of images for inference at any time. In other words, multiple tasks (one task per image) can be submitted for inference simultaneously, with N being any value.
- The composite model maximizes the capabilities of ENNP hardware to improve computation speed.

Note: The composite model supports parallel inference on both NPU and DSP hardware.

The NPU Runtime API invocation is shown in Figure 4-1.

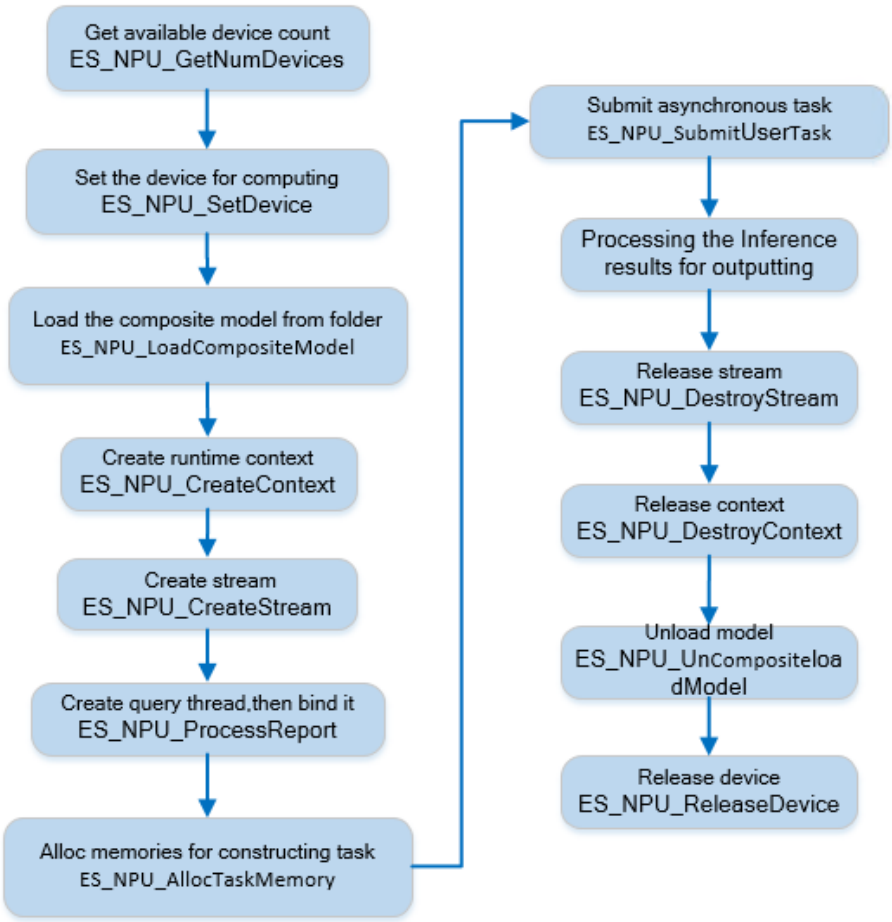

Figure 4-1 NPU API Invocation Process

4.2.2 Using es_run_model for Model Evaluation

To facilitate users in evaluating models using NPU Runtime, NPU Runtime provides the es_run_model tool. This tool offers several options that make it easy to test various aspects of the model, such as speed and accuracy. The parameter information for es_run_model is as follows:

```
usage: es_run_model --model=string [options] ...
options:
-M, --mode the mode of es_run_model(int [=0], 0 is sync mode, 1 is nbatch mode.)
-m, --model path to a model file (string)
-b, --batch             the model batch count (int [=1])
-r, --repeat            repeat times running a model (int [=1])
-w, --warmup              repeat times before running a model to warming up (int [=0])
-i, --input-dir         the directory of each inputs (folders) located (string [=])
-o, --output-dir          the directory of each outputs (folders) will saved in (string [=])
-l, --list            the list of inputs which will test (string [=])
-v  --verify            verify outputs after running model
-s  --save-perf           save performance result(min, max, avg) as a json file
-?, --help            print this message
```

For detailed application examples, refer to Section 6.6.1.

4.3 NPU Runtime Debugging

NPU Runtime currently provides debugging information based on the Release version, with the default debug level set to "Error".

```
[1970-01-01 08:01:01] [ E ] [ES_NPU] [-1466355936] [readModelToMem] [186] couldn't open /IBU_8T/IBU_SOFTWARE/npu_release_models/npu_release/opt/eswin/data/models/mobilenet_v1_int8_1x224x224x3I

[1970-01-01 08:01:01] [ E ] [ES_NPU] [-1466355936] [loadModel] [34] read model to memory error, err=-1609605117
```

Additionally, debugging can also be performed using error codes returned by the function calls. For more details, please refer to the "ENNP Developer's Guide".

5. AcceleratorKit Development Guide

AcceleratorKit is an API library for calling operators that are not compiled into the ONNX model, such as some more complex pre-processing and post-processing operators. AcceleratorKit provides computing acceleration functions that can be executed on different types of devices to maximize the computing acceleration capabilities of different types of devices.

5.1 Software Framework

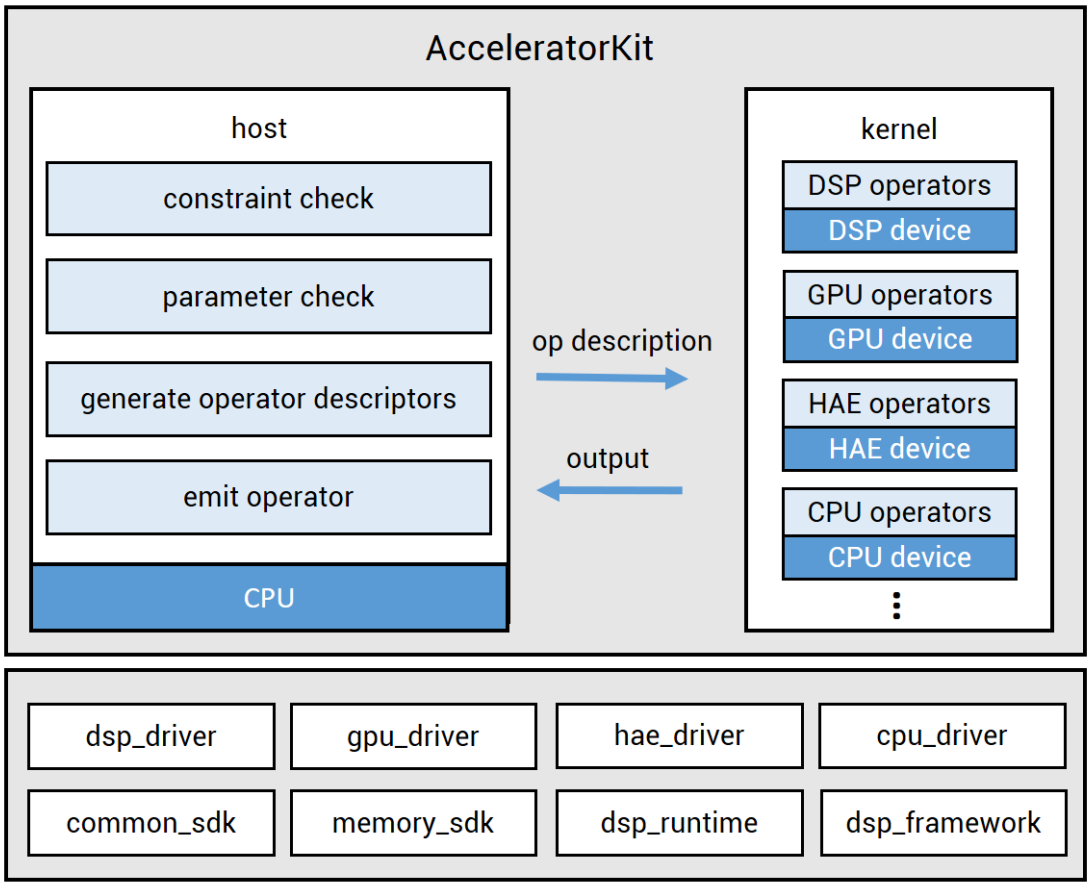

Figure 5-1 AcceleratorKit Software Architecture

The currently supported operators Table 5-1

Table 5-1 List of operators supported by AcceleratorKit

Operator Name	Operator Function

ES_AK_DSP_CosDistance	Calculates the cosine distance between two sets of data.
ES_AK_DSP_Argmax	Perform Argmax calculation to perform maximum sorting on the specified dimension of the input data.
ES_AK_DSP_DetectionOut	Post-process the output feature map of the detection network.
ES_AK_DSP_Softmax	Perform Softmax calculation.
ES_AK_DSP_PerspectiveAffine	Apply perspective transformation to the image.
ES_AK_DSP_WarpAffine	Apply an affine transform to the image.
ES_AK_CPU_SimilarityTransform	Compute the similarity transformation matrix.

For detailed definitions and instructions on how to use operators, please refer to *the ENNP Developer's Manual* or contact technical support.

5.2 Software Usage Process

5.2.1 Dependent Modules

- DSP related modules include dsp_compiler, dsp_runtime, dsp_driver, dsp_fw, and dsp_kernels.
- The common_sdk is used to provide common data types.
- The memory_sdk is used to provide memory management tools.

5.2.2 Development Process

- Call interfaces such as ES_GetVersion to complete operations such as obtaining version information and setting log levels;
- Call ES_AK_Init to initialize hardware device resources;
- Call ES_AK_SetDevice to configure computing devices based on specific requirements;
- Call ES_SAK_GetDevice to obtain the computing device;
- Configure the input and output memory of operators through ES_MemAlloc;
- Call the corresponding operator interface based on the configured specific hardware device, such as ES_AK_DSP_DetectionOut;
- Release the configured input and output memory by calling ES_SYS_MemFree after the operator calculation is completed;
- After the calculation is completed, ES_AK_Deinit needs to be called to release device resources;

For detailed API usage, please refer to the AcceleratorKit chapter in *the ENNP Developer Manual*.

6. Model Development Example

This chapter introduces the complete operation process of ONNX model conversion, compilation, and development board deployment and operation. The complete ONNX process is as follows:
Use the EsQuant tool on the PC based on the x86 platform, the ONNX original model and the quantization configuration (Json file) to generate the quantization table table.json (the actual quantization table name is composed of the onnx file name and the parent directory);
Use the EsAAC model compiled on the PC based on the x86 platform, the ONNX original model and the quantization table table.json (the actual quantization table name is composed of the onnx file name and the parent directory) to generate a model supported by ENNP hardware;
Use the x86-based PC-side tool EsSimulator to simulate the ENNP hardware to verify the correctness of the model;
Use the RSIC-V platform-based tool es_run_model to verify the model's functionality on hardware and calculate its performance.
Use the RSIC-V platform to call the NPU Runtime interface to develop user apps and perform task

reasoning on the hardware.
In addition, NPU Runtime provides sample _npu reference code to accelerate the development of user apps.

6.1 Sample file description

Sample files involve tools based on the x86 platform PC and development samples based on the RSIC-V platform development board. Therefore, ENNP samples are described as PC samples and board samples.

6.1.1 PC sample

pc sample directory is shown below. The EsNNTools folder contains the docker image file of the required tool chain, and the sample/yolov3 folder contains the input files required for quantization, compilation, and other links.

```
└─── EsNNTools
├─── esquant-1.0-py3-none-any.whl
    ├─── esquant_docker.tar
    ├─── esaac_essimulator_docker.tar
    ├─── Example_with_config.py
    ├─── libs
    │    ├─── libconv_model_creator.so
    │    ├─── libmanager.so
    │    ├─── libmapper.so
    │    └─── libtimeloop-model.so
  └─── sample
        ├─── yolov3
                │   ├─── esaac
                │   │   └─── table.json
                │   ├─── esquant
                │   │   ├─── alys_list.txt
                │   │   ├─── config.json
                │   │   └─── img_list.txt
                │   ├─── essimulator
                │   │   ├─── ifmap.bin
                │   │   ├─── ofmap0.bin
                │   │   ├─── ofmap1.bin
                │   │   ├─── ofmap2.bin
                │   │   └─── yolov3.json
                │   └─── model
                │       └─── yolov3_sim_extract_416_notranspose_noreshape.onnx
                └─── yolov5
          ├─── eaac
          │   └─── table.json
          ├─── esquant
          │   ├─── alys_list.txt
          │   ├─── config.json
          │   └─── img_list.txt
          ├─── essimulator
          │   ├─── ifmap.bin
```

```
│   ├── ofmap0.bin
│   ├── ofmap1.bin
│   ├── ofmap2.bin
│   └── yolov5.json
└── model
    └── yolov5s_416-sim_extract.onnx
```

6.1.2 board sample

Install the sample deb package as root account, which requires an Ethernet connection:

- Install the npu-sample package:

  apt install es-sdk-sample-npu

  After installation, sample is located in the / opt/eswin/sample-code/npu_sample/npu_runtime_sample directory

- Install npu resnet50 package:

  apt install es-sdk-sample-npu-resnet50

  After installation, sample is located in /opt/eswin/sample-code/ npu_sample/npu_resnet50_sample directory

- Install the npu-mobilenetv2 package:
  apt install es-sdk-sample-npu-mobilenetv2
  After installation, Sample is located in the directory /opt/eswin/sample-code/npu_sample /npu_mobilenetv2_sample

6.1.2.1 es-sdk-sample-npu

The sample directory (npu_runtime_sample) provided by NPU Runtime mainly includes sample codes for calling NPU Runtime interfaces and model directories: src and models. The detailed directory structure is as follows:

```
├─── models
│   └─── yolov3
│   ├─── es_yolov3_classes.txt
│   ├─── es_yolov3_post_process.json
│       ├─── es_yolov3_pre_process.json
│       ├─── git_yolov3_416_mix_1x3x416x416_dyn_latency.model
│       ├─── git_yolov3_416_mix_1x3x416x416_dyn_latency.ofmap_order.txt
│       ├─── git_yolov3_416_mix_1x3x416x416_dyn.model
│       ├─── input
│       └─────model.json
└─── src
    ├─── build.sh
    ├─── CMakeLists.txt
    ├─── README.md
    ├─── sample_npu_comm.cpp
    ├─── sample_npu_comm.h
    ├─── sample_npu.cpp
    └─── utils
```

- Sample code directory (src)

The entry point of the sample code provided by NPU Runtime is the sample_npu.cpp file. This file code contains a series of sample cases, such as synchronous, asynchronous, multi-stream, context, and dynamic batchsize cases. Each case has a separate entry function in the code, so users can easily find the sample that suits their application.

The README.md document contains examples of how to compile the sample_npu code and run the sample_npu command line.

Utils mainly contains some auxiliary tool libraries, such as image preprocessing and postprocessing of inference data.

- Models directory (models)
  - Configuration file introduction:
    es_yolov3_classes.txt contains the class label information.
    es_yolov3_pre_process.json contains the preprocessing strategy and parameters.
    es_yolov3_post_process.json contains the parameters related to the post-processing operator.
  - Input directory:
    This directory contains the Pictures and P reprocessed directories. The Pictures directory stores the original image files to be inferred. The P reprocessed directory stores the binary data files that have been preprocessed (Decode, Resize, Normalization and CVT).
  - Output directory:
    The output directory is not used in this sample. The es_run_model tool will use the data in the output directory to verify the NPU. The correctness of the results of R untime reasoning.

6.1.2.2 es-sdk-sample-npu-resnet50

npu_resnet50_sample) provided by NPU resnet50 mainly includes sample codes and model directories for calling the NPU Runtime interface, binary programs: src and models, bin. The detailed directory structure is as follows:

```
├─── bin
│   └─── es_ai_inference
├─── models
│   ├─── es_resnet50_classes.txt
│   ├─── es_resnet50_post_process.json
│   ├─── es_resnet50_pre_process.json
│   ├─── git_resnet50_mix_4x3x224x224_dyn_latency.model
│   ├─── git_resnet50_mix_4x3x224x224_dyn_latency.ofmap_order.txt
│    ├─── git_resnet50_mix_4x3x224x224_dyn.model
│    ├─── git_resnet50_mix_4x3x224x224_dyn_throughput.model
│    ├─── git_resnet50_mix_4x3x224x224_dyn_throughput.ofmap_order.txt
│    └─── input
└─── src
    ├─── build.sh
    ├─── CMakeLists.txt
    ├─── common
    │    └─── utils
    ├─── es_ai_inference.cpp
    ├─── es_ai_inference.h
    ├─── list.txt
    ├─── main.cpp
    ├─── README.md
    ├─── runtime
    │    └─── include
└─── utils
├─── json
├─── npu_test_utils.cpp
├─── npu_test_utils.h
├─── postprocess
└─── preprocess
```

- Sample code directory (src)
  The entry point of the sample code provided by NPU resnet50 is the es_ai_inference.cpp file. The README.md document contains the compilation method and running commands for

compiling the resnet50 sample code.
Utils mainly contains some auxiliary tool libraries, such as image preprocessing and postprocessing of inference data.

- Models directory (models)
  - Configuration file introduction:
    es_resnet50_classes.txt contains the class label information.
    es_resnet50_pre_process .json contains the preprocessing strategy and parameters.
    es_resnet50_post_process .json contains the post-processing operator related parameters.
  - Input directory:
    This directory stores the original image files to be inferred.
- Binary program directory (bin)
  This directory stores binary executable programs.

6.1.2.3 es-sdk-sample-npu-mobilenetv2

npu_mobilenetv2_sample) provided by NPU mobilenetv2 mainly includes sample codes and model directories for calling NPU Runtime interfaces, binary programs: src, models, and bin. The detailed directory structure is as follows:

```
├──── bin
│   └──── es_ai_inference
├──── models
│   ├──── es_mobilenet_classes.txt
│   ├──── es_mobilenet_post_process.json
│   ├──── es_mobilenet_pre_process.json
│   ├──── git_mobilenetv2_mix_4x3x224x224_dyn_latency.model
│   ├──── git_mobilenetv2_mix_4x3x224x224_dyn_latency.ofmap_order.txt
│    ├──── git_mobilenetv2_mix_4x3x224x224_dyn.model
│    ├────git_mobilenetv2_mix_4x3x224x224_dyn_throughput.model
│    ├──── git_mobilenetv2_mix_4x3x224x224_dyn_throughput.ofmap_order.txt
│    └──── input
└──── src
   ├──── build.sh
   ├──── CMakeLists.txt
   ├──── common
   │    └──── utils
   ├──── es_ai_inference.cpp
   ├──── es_ai_inference.h
   ├──── list.txt
   ├──── main.cpp
   ├──── README.md
   ├──── runtime
   │    └──── include
└──── utils
├──── json
├──── npu_test_utils.cpp
├──── npu_test_utils.h
├──── postprocess
└──── preprocess
```

- Sample code directory (src)

  The entry point of the sample code provided by NPU mobilenetv2 is the es_ai_inference.cpp file.

  The README.md document contains the compilation method and running commands for compiling the mobilenetv2 sample code.

  Utils mainly contains some auxiliary tool libraries, such as image preprocessing and postprocessing of inference data.
- Models directory (models)

- Configuration file introduction:
  es_mobilenet_classes.txt contains the category label information.
  es_mobilenet_pre_process .json contains the preprocessing strategy and parameters.
  es_mobilenet_post_process .json contains the parameters related to the post-processing operator.
- Input directory:
  This directory stores the original image files to be inferred.

- Binary program directory (bin)
  This directory stores binary executable programs.

6.2 Installing the Development Environment

This section introduces the preparation of the development environment before using the E NNP tool chain. To facilitate the deployment and installation of the tool chain, E NNP provides docker containers for tool chain integration, where Es Quant is packaged as one image, and Es AAC and Es Simulator are packaged as one image, a total of two docker images, please pay attention to the distinction. The following describes the installation and startup of these two container images, and the image files involved are all in the EsNNTools folder of the pc sample.

6.2.1 EsQuant deployment

EsQuant is released in docker form. The overall installation and running process of the docker container is as follows:
Load the docker image.

```
eswin@debian:~# docker load -i ${EsNNTools}/esquant_docker.tar
```

View the docker image.

```
eswin@debian:~# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE
esquant v 0.7 c711e4a379ca 4 days ago 18.2GB
```

Create a container.

```
# Start the gpu image
# --privieged=true Privileged mode: Give the container almost the same permissions as the host
eswin@debian:~# docker run --name [container_name] -it --runtime=nvidia --privieged=true --gpus all -v /hostPath:containerPath esquant:v0.7 /bin/bash
eswin@debian:~# docker run --name [container_name] -it --runtime=nvidia --gpus all -v /hostPath:containerPath esquant:v0.7 /bin/bash

# Start cpu image
eswin@debian:~# docker run --name [container_name] -it -v /hostPath:containerPath esquant:v0.7 /bin/bash
```

Install the whl package.

```
root @d76a884a0d56:~$cd EsNNTools/
root @d76a884a0d56:~$pip3 install esquant*.whl
```

Check the version information and git - hash information to confirm whether the deployment is successful.

```
root @d76a884a0d56:~$python3
import esquant
esquant.version()

version: 1.0.0
sha id: af6da07d5c205583f1a71cfee12339ed10623d08
```

6.2.2 EsAAC and EsSimulator deployment

EsAAC and EsSimulator are released in the same docker form, and the operation process is as follows:

Docker image:

```
eswin@debian:~# docker load -i ${EsNNTools}/esaac_essimulator_docker.tar
```

View the docker image:

```
eswin@debian:~# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE
esaac_essimulator latest 26c09ab3286b 7 minutes ago 4.02GB
```

Create a container:

```
eswin@debian:~# docker run [-v hostPath:containerPath] --name [container_name] -it --rm esaac_essimulator:latest /bin/bash
```

View Es AAC version information:

```
eswin@d76a884a0d56:~$ ls
EsAAC EsSimulator EsSimulator_copyright.txt dsp_kernels

eswin@ d76a884a0d56:~$ ./EsAAC --version
eaac version: 0.0.3
```

View Es Simulator version information:

```
eswin@d76a884a0d56:~$ ls
EsAAC EsSimulator EsSimulator_copyright.txt dsp_kernels

eswin@d76a884a0d56:~$ ./EsSimulator --version
EsSimulator version: 0.0.3(Fri Jul 19 14:06:01 2024 +0800).
```

6.3 ONNX model export

This section describes how to export the yolov 3 model from the official website and crop the model.

6.3.1 Yolov 3 model export

Download the official website code from the GitHub link and export the yolov3 model according to the following process. The quantization tool supports opset versions 1 1-14. You need to select version 1 1-14 when exporting the model:

Note that the official commented out the onnx library and onnx - simplifier library in requirements.txt. Remember to modify and install them when installing the environment. In addition, the torch and torchvision versions need to be installed torch ==1.12.0 and torchvision ==0.13.0 versions respectively.

```
git clone https://github.com/ultralytics/yolov3 # clone
cd yolov3
pip install -r requirements.txt # install

m kdir model_file
wget https://github.com/ultralytics/yolov3/releases/download/v9.6.0/yolov3.pt
python3 export.py --weights model_file/yolov3.pt --img-size 416 --simplify --opset 13 --include onnx
```

6.3.2 Yolov 3 model cutting

Since the exported model contains some post-processing operations, and the quantization tool does not support post-processing operations for the time being, the post-processing part needs to be trimmed. Note that the trimmed part is the operations after conv. Fill in the input_names and output_names of the trimming script.

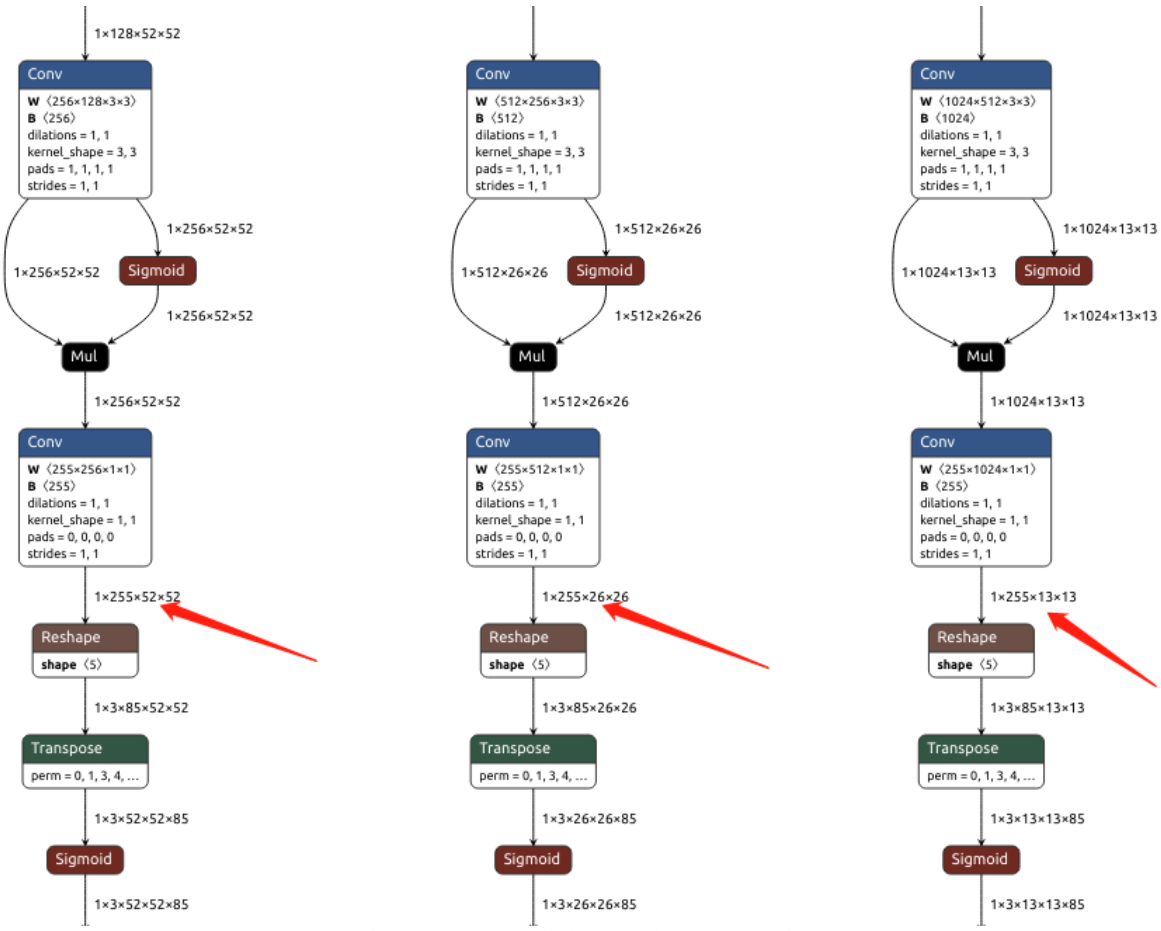

Figure 6 -1 Model cropping example

```
import onnx

input_onnx = "yolov3.onnx"
input_names = ["images"]
output_names = ["onnx::Shape_406", "onnx::Shape_461", "onnx::Reshape_516"]

output_onnx = input_onnx
cut_suffix = "_sim_extract_416_notranspose_noreshape." + input_onnx.split('.')[-1]
new_output_onnx = output_onnx.replace(".onnx", cut_suffix)
print(new_output_onnx)

onnx.utils.extract_model(input_onnx, new_output_onnx, input_names, output_names)
```

6.4 ONNX model quantization and accuracy analysis

This section describes the process of using the EsQuant tool to perform quantification and quantification accuracy analysis.

6.4.1 Generate a quantitative model using EsQuant

6.4.1.1 Configure the config.json file

For the yolov3 model, the recommended configuration parameters are as follows. The path-related parameters should be modified according to the actual development environment. Please refer to 2.2.1. The complete config.json, img_list.txt , and alys_list.txt files can be obtained under EsNNTools / sample / yolov3 / esquant . The images used in img_list.txt are from the coco2017 dataset, which can be downloaded by the user.

Note that the calibration data needs to be within 1,000 images and must be configured in the

configuration file based on whether a GPU is present.

```
root @d76a884a0d56:~$ cat config.json
{
    "version": "1.3",
    "model": {
        "model_path": "/yolov3_sim_extract_416_notranspose_noreshape.onnx",
        "save_path": "/home/yolov3/",
        "images_list": "/home/yolov3/cal_lists.txt",
        "analysis_list": "/home/yolov3/file_list.txt"
    },
    "quant": {
        "quantized_method": "per_channel",
        "quantized_dtype": "int8",
        "requant_mode": "mean",
        "quantized_algorithm": "at_eic",
        "optimization_option": "auto",
        "bias_option": "absmax",
        "nodes_option1": [],
        "nodes_option2": [],
        "nodes_i8": [],
        "nodes_i16": [],
        "mean": [
            0,0,0
        ],
        "std": [
            1,1,1
        ],
        "norm": true,
        "scale_path": "",
        "enable_analyse": true,
        "device": "cpu",
        "output_variables_dtype": {
            "onnx::Shape_406": "int16",
            "onnx::Shape_461": "int16",
            "onnx::Reshape_516": "int16"
        }
    },
    "preprocess": {
        "input_format": "RGB",
        "keep_ratio":false,
        "resize_shape": [
          416,416
        ],
        "crop_shape": [
          416,
416
]
}
}
```

6.4.1.2 Perform quantization

Run the following script to perform quantization. The built-in Yolo preprocessing method is used here. Users can also define their own preprocessing functions based on the provided samples. The sample code is obtained in the EsNNTools directory.

```
root @d76a884a0d56:~$python Example_ with_config.py --config _ path /config.json --preprocess_name Yolo
```

After running the code, the following display is obtained in the terminal, indicating that the quantization is completed.

```
When preparing to quantize your network, check the following settings:
TARGET PLATFORM : EIC
NETWORK INPUTSHAPE    : [1, 3, 224, 224] [2024-01-30 19:56:02][warn][config.cpp:66]:Configuration
[10:05:12] ESQUANT Quantize Simplify Pass Running ...          Finished.
[10:05:12] ESQUANT Quantization Fusion Pass Running ...        Finished.
Calibration Progress(Phase 1): 100%|████████████████| 1000/1000 [00:13<00:00, 75.10it/s]
Calibration Progress(Phase 2): 100%|████████████████| 1000/1000 [00:16<00:00, 62.34it/s]
Finished.
[10:05:54] ESQUANT Quantization Fusion Pass Running ...        [Warning] More than 1 pattern root was found, Complext Pattern might cause memory overflow ...
[Warning] More than 1 pattern root was found, Complext Pattern might cause memory overflow ...
[Warning] More than 1 pattern root was found, Complext Pattern might cause memory overflow ...
Finished.
[10:05:54] ESQUANT Parameter Quantization Pass Running ... Finished.
[10:05:54] EIC Parameter Quantization Pass Running...
[10:05:54] EIC Input Tensor Quantization Pass Running ... Finished.
[10:05:54] ESQUANT Passive Parameter Quantization Running ... Finished.
[10:05:54] ESQUANT Quantization Alignment Pass Running ... Finished.
[10:05:54] ESQUANT Parameter Baking Pass Running ... Finished.
Network Quantization Finished.
The network quantization error (COSINE) is being calculated. The error of the last layer should be less than 0.1 to ensure quantization accuracy:
Analyzing Graphwise Quantization Error(Phrase 1):: 100%|██████████| 8/8 [00:01<00:00, 6.30it/s]
Analyzing Graphwise Quantization Error(Phrase 2):: 100%|██████████| 8/8 [00:01<00:00, 6.04it/s]
The network quantization is finished and the target file is being generated:
[Warning] File /quantized- yolov 3-eic-at-0715.onnx is already existed, Exporter will overwrite it.
[Warning] File /quantized- yolov 3-eic-at-0715.json is already existed, Exporter will overwrite it.
```

After the quantization is completed, enter the "save_path" directory defined in the config.json file , and you can see the generated table.json file (the actual quantization table name is composed of the onnx file name and the parent directory), the onnx file after graph fusion, and the precision analysis result files precision_accumulate_result.txt and precision_reset_result.txt . In the following document, table.json is the quantization information table, and the file name is no longer specifically stated. The "save_path" parameter in the config.json file is a mandatory parameter. If it is not configured, an error will be reported. The table.json file is one of the inputs for model compilation, which will be described in detail in the model compilation section.

The quantized onnx performs some graph fusion operations. When compiling the model, you can use the original onnx model or the quantized onnx model. When using the EsGoldenDataGen tool, you need to use the original onnx file.

```
root @d76a884a0d56:~$ cd ${save_path}
root @d76a884a0d56:~$ ls

precision_accumulate_result.txt
precision_reset_result.txt
table.json
yolov3. onnx
```

6.4.2 Quantitative model accuracy analysis

Precision analysis relies on the precision_accumulate_result.txt and precision_reset_result.txt files generated during the quantization process. The information counted by precision_accumulate_result.txt is the cosine similarity error accumulated layer by layer for the quantization model and the floating-point model; the information counted by precision_reset_result.txt is the cosine similarity error obtained by resetting the input for each layer of the quantization model and the floating-point model. The cosine similarity information is recorded inside the file.

The details are as follows:

```
root @d76a884a0d56:~$ cat precision_accumulate_result.txt
Conv_0 : 0.9973047614097595
Sigmoid_1 : 0.999542486667633
Mul_2 : 0.9864129543304443
Conv_3 : 0.9846434354782104
Sigmoid_4 : 0.9976542234420777
Mul_5 : 0.9705418705940246
```

First, check the last op - name in precision_accumulate_result.txt (the last layer operation in the network structure). Whether the cos_dist(mean) corresponding to the node name is greater than 95%. If it is greater than the threshold, the error is considered to be within an acceptable range. If not, you need to check another file, namely precision_reset_result.txt. At this time, you need to locate which layer has a significant decrease in precision loss, and then replace this layer with int16 in the configuration file config.json (do not change the datatype of the first layer operation and the last output operation). The operation is to find nodes_int16 in the configuration file and fill in the node-name field. Re-quantize and conduct precision analysis, and then check the quantization precision error. After the process is completed, a higher precision can generally be obtained. If the precision is still lower than the threshold, you need to check whether the configuration parameters are correct. For more precision analysis methods and precision improvement methods, please refer to Section 2.3.

6.4.3 GoldenData Generation

EsGoldenDataGen is used to generate reference data, including input feature map and output feature map, and then provide the data generated by the tool to EsSimulator for verification.

EsGoldenDataGen runs the floating point model based on the ONNXRuntime framework and then quantizes it, which can be used as real data. EsSimulator runs the quantized model and compares the quantized result with the result of ONNXRuntime by cosine similarity. The current threshold for cosine similarity is 95%. If the threshold requirement is reached, it is considered that the quantization accuracy loss basically meets the requirements.

Note that this process is only used to verify the quantization process. The data provided must be calibration data. Just provide one picture.

6.4.3.1 Configure yolov3 run file

The complete configuration file can be obtained in the EsNNTools/sample/yolov3 / essimulator directory. It is recommended that customers configure the esgolden_data_gen parameters according to actual conditions. For details, please refer to the EsGoldenDataGen tool introduction.

```
root @d76a884a0d56:~$ cat yolov3.json
{
"model": {
"model_path": "/yolov3/yolov3_sim_extract_416_notranspose_noreshape.onnx",
"save_path": "/Model/yolov3/",
"batchsize": 1
},
"dataset": {
"data_root": "/img_list/",
        "transform_cfgs": [
      {
        "type": "ToRGB"
      },
      {
        "type": "LetterBox",
        "new_shape": [
          416,
          416
        ],
        "auto": false
      },
      {
        "type": "Normalize",
        "norm": true
      },
      {
        "type": "ToCHW"
      },
      {
        "type": "TransPrecision",
        "precision": "float32"
      }
        ]
    },
    "quant": {
        "input": {
            "format": {
                "images": [
                    1,
                    3,
                    416,
                    416,
                    1
                ]
            },
            " color_format_convert ": true,
            "scale": {
                "images": 0.007874015718698502
            },
            "data_type": 0,
```

```
        },
        "output": {
            "format": {
                "onnx::Shape_406": [1,255,52,52,1],
                "onnx::Shape_461": [1,255,26,26,1],
                "onnx::Reshape_516": [1,255,13,13,1]
            },
            "scale": {
                "onnx::Shape_406": 0.13676564395427704,
                "onnx::Shape_461": 0.12968941032886505,
"onnx::Reshape_516": 0.12968522310256958
},
"data_type": 0
}
}
}
```

in the EsQuant container to generate ifmap and ofmap.

```
root @d76a884a0d56:~$ python -m esquant.es_goldendata_gen.EsGoldenDataManager --config yolov3.json
```

The corresponding input ifmap and ofmap will be generated under the "save_path" path defined in yolov3.json. The file name consists of the operation name and the image name, connected by the "_" character in the middle, as shown below.

```
root @d76a884a0d56:~$ cd ${save_path}
root @d76a884a0d56:~$ ls

Conv_248_000000092416.bin Conv_292_000000092416.bin
Conv_336_000000092416.bin images _000000092416.bin
```

6.5 Model compilation

6.5.1 Compile the model using EsAAC

Prepare the quantitative file table.json

Compilation requires the quantization file table.json and the O NNX model file as input. After successful quantization in 6.3.1, the quantization file table.json of the model will be generated. Users can also obtain the preset table.json in the EsNNTools/sample/yolov3/EsAAC path .

table.json and ONNX model files to the mount directory ${workdir} of the Es AAC and Es Simulator containers , and execute the Es AAC compilation command in the container .
Run the EsAAC command line

```
eswin@d76a884a0d56:~$ ./EsAAC --input-model $ {workdir} /yolov3_sim_extract_416_notranspose_noreshape.onnx --quant-stats ${workdir}/table.json
```

Print prompt information when running:

```
[EsAAC] Model is starting to compile ...
[EsAAC] Model parsing time: 1s:17ms:454us
[EsAAC] Model conversion time: 7ms:360us
[EsAAC] Model generation time: 11s:346ms:23us
[EsAAC] Model simulation running time: 12us
[EsAAC] Model compilation total time: 12s:465ms:287us
[EsAAC] Model compilation completed.
```

Printing "Mode compilation completed" indicates successful operation. After the operation, the

quantized offline model file $ {workdir}/yolov3_sim_extract_416_notranspose_noreshape.model is generated in the current mount directory .

6.5.2 Use EsSimulator to simulate and verify

To do basic testing based on EsSimulator, you need to specify the model, input, and output parameters. Copy 6.4.3to the mount directory ${workdir} of the EsAAC and EsSimulator containers, and execute the following commands:

```
eswin@d76a884a0d56:~$ ./EsSimulator --model=${workdir}/yolov3_sim_extract_416_notranspos
e_noreshape.model --input=${workdir}/ifmap.bin --output=${workdir}/ofmap0.bin ,
${workdir}/ofmap1.bin,${workdir}/ofmap2.bin
```

It should be noted here that the order ofmap output by the multi-output model may be inconsistent with the output on the model structure, and an error will be reported when it is sent to the test program. In this case, the order ofmap needs to be specified. EsACC will generate a model_name.ofmap_order.txt file in the current directory to describe the order ofmap output. The file format is as follows (models with only one output do not need to pay attention to this file) :

```
5 Conv_292- conv_ biasadd
6 Conv_336- conv_ biasadd
7 Conv_248- conv_ biasadd
```

Therefore, for a multi-output model, when performing EsSimulator simulation verification, multiple ofmap data need to be sent to the EsSimulator test program in the order described in model_name.ofmap_order.txt.

After EsSimulator is finished running, there will be relevant print prompts to indicate whether the model runs successfully or not, which is convenient for locating model problems. The following is a test example print of the y olov3 model.

```
EsSimulator version: 0.0.3(Mon Jul 22 14:36:18 2024 +0800).
yolov3_sim_extract_416_notranspose_noreshape.model test successful.
EsSimulator(version: Mon Jul 22 14:36:18 2024 +0800) finished.
```

When the model data is based on float32 or float16, you can set the error tolerance. When the model compares data, if the absolute error and relative error are less than the tolerance, the model comparison is correct. The configuration tolerance example is as follows:

```
eswin@d76a884a0d56:~$ ./EsSimulator --model=${workdir} yolov3_sim_extract_416_notransp
ose_noreshape .model --input=${workdir}/ifmap.bin --output=${workdir}/ofmap0.bin ,
${workdir}/ofmap1.bin,${workdir}/ofmap2.bin --tolerance=0.008
```

6.6 Model Inference

Model deployment includes calling NPU-related API interfaces, configuring JSON files for post-processing-related operators, and finally running post-processing-related codes.

6.6.1 es_run_model model evaluation tool usage

The es_run_model tool is mainly used to verify the functionality and accuracy of the model on the hardware and to statistically analyze the performance of the model.

This section demonstrates the use of the es_run_model debugging tool. The following command may have format problems if copied directly. It is recommended to format it after copying (manually remove the line breaks). The model used in the example is a composite model. The model can be unpacked from the tar package and copied to the test directory, such as /opt/eswin/data/npu/yolov3/. The user can switch to the root account and execute directly, or replace the model with the offline model compiled in 6.5.1. For the detailed meaning of the parameters in the command line, please refer to Section **错误!未找到引用源。**.

Example 1: Simply testing the model running speed

```
root@rockos-eswin:/# /opt/eswin/bin/es_run_model -m /opt/eswin/data/npu/yolov3/ -r 1000
--------------------------------------------------------------------------------
avg = 11.6540 ms, fps = 85.8074 frames/s ------------------------------------------------------
-------------
```

In this case, the model can be used to determine the shape and precision of the input data, and then the input data of the model can be generated to test the model without the user specifying an input folder.

Example 2: Specifying the input and output folder parameters

In this mode, the list parameter does not need to be specified. es_run_model will search for input data from the input directory and save the inference results in the output directory. If the output directory is not specified, the results will not be saved.

The es_run_model test command is as follows:

```
root@rockos-eswin:/# /opt/eswin/bin/es_run_model -m /opt/eswin/data/npu/yolov3/ -r 10 -i
/opt/eswin/data/npu/yolov3/input/preprocessed/0/ -o /opt/eswin/data/npu/yolov3/output
--------------------------------------------------------------------------------
avg = 11.6852 ms, fps = 85.5783 frames/s
--------------------------------------------------------------------------------
```

Note: The output directory specified by the -o parameter above may not exist yet. It will be automatically created after specifying it here.

When the verify parameter is specified, the specified output directory will be searched for reference data for comparison. If it is not found, an error will be reported. At this time, it should be noted that for multi-output models, multiple output files in the output folder need to be renamed to meet the order ofmap. Here, based on the ofmap order described in the model_name.ofmap_order.txt file generated by EsAAC, the multiple output files are renamed in order, for example, renamed to ofmap 0. bin, ofmap 1. bin, ofmap 2. bin, etc. Only when the order is met can the verification of the multi-output model be correct.

Example 3: Specifying the input and output folders and the list parameter

In addition to specifying the input and output directories, the list parameter is added. When there are subdirectories under the input directory, we need to use list to specify which subdirectories of the input directory to search for input files. In this mode, the list parameter needs to be specified, which indicates which subdirectories in the directory are used as input data.

The directory structure is as follows:

```
root@rockos-eswin:/# cat /opt/eswin/data/npu/ yolov3/ input/preprocessed/list.txt
0
1
```

The es_run_model test command is as follows:

```
root@rockos-eswin:/# /opt/eswin/bin/es_run_model -m /opt/eswin/data/npu/yolov3/ -i
/opt/eswin/data/npu/yolov3/input/preprocessed/ -o /opt/eswin/data/npu/yolov3/output -l
/opt/eswin/data/npu/yolov3/input/preprocessed/list.txt
--------------------------------------------------------------------------------
avg = 11.8720 ms, fps = 84.2318 frames/s ------------------------------------------------------
-----------
```

When the --save-perf parameter is added, the performance test data will be saved in a json file with the following content:

```
root@rockos-eswin:/# cat model_perf_data.json
{
"info": {
"average_inference_time_ms":11.6080,
"max_time_cost":11.6480,
"min_time_cost":11.5680,
"model_path": "yolov3_sim_extract_416_notranspose_noreshape.model",
"repeat_times":1
}
}
```

Example 4 : Using dynamic batch to test model running speed

The dynamic batch interface allows users to batch process images, and the number of each batch can be adjusted dynamically. Use the -M parameter to set the test mode, and the -b parameter to set the batch size. (Note: 1. Dynamic batch supports setting input and output. 2. If an out of memory error occurs, you need to adjust the environment variable ES_NPU_MEMPOOL_CAPABILITY to a value greater than 1 28)

```
root@rockos-eswin:/# /opt/eswin/bin/es_run_model -m /opt/eswin/data/npu/yolov3/ -b 4 -r 1000
-------------------------------------------------------------------------------
avg = 11.6896 ms, fps = 85.5463 frames/s
-------------------------------------------------------------------------------
```

Dynamic batch uses the average value of multiple tasks tested asynchronously, so there is no maximum or minimum value.

6.6.2 Model Inference

This section mainly introduces how to refer to the sample_npu source code provided by NPU Runtime to develop your own smart App and how to execute the sample_npu program and output the inference results.

Next, we will introduce the simple process of the sample_npu source code calling the NPU R untime interface. The sample_npu source code is simply divided into three parts: image pre-processing, a simple process of calling the NPU R untime interface, and output post-processing.

Image pre-processing

For the json file that configures the pre-processing parameters, refer to 6.1.2

NPU Runtime calling process

```
#Set the device number
ES_NPU_GetNumDevices(&deviceNum);
# Set the device ID
ES_NPU_SetDevice(deviceId);
# Load the model file
ES_NPU_LoadModelFromFile(&modelId, modelPaths[modelIdx].c_str());
# Create context
ES_NPU_CreateContext(&context, 0);
# Create a stream
ES_NPU_CreateStream(&stream);
# Get the number of input tensors
ES_NPU_GetNumInputTensors(modelId, &numInputs);
# Get input tensor description
ES_NPU_GetInputTensorDesc(modelId, inputTensorId, &tensor);
# Allocate tensor memory based on input tensor description
ES_SYS_MemAlloc(&task.inputFd[inputTensorId].memFd, SYS_CACHE_MODE_NOCACHE,
"npu_sample", "mmz_nid_0_part_0", tensor.bufferSize);
#Copy the pre-processed image content to the tensor memory
memcpy(inputBuf, int8Img.data, outDataLen);
# Synchronous submission of tasks
ES_NPU_SubmitSync(task, 1);
# Submitting tasks asynchronously
ES_NPU_SubmitAsync(task, 1, stream);
# Query Tasks
ES_NPU_ProcessReport(info->stream, info->queryMs);
```

In addition to the traditional single-model synchronous and asynchronous task reasoning mentioned above, the NPU runtime API also provides a dynamic batch interface and composite model functions.

asks as required, and the number of T asks submitted can be flexibly handled. The composite model improves the performance of inference through real NPU and DSP hardware parallel computing.

The dynamic batch interface allows users to process tasks in batches, and the batch size does not need to be bound to the model. Dynamic batches are allocated input and output tensor memory by the runtime, and can be used with composite models as follows.

```
# Call the runtime interface to allocate memory. The third parameter is the batch size.
ES_NPU_LoadCompositeModel (&modelId, modelPaths);
ES_NPU_AllocTaskMemory(modelId, stream, 1, &taskMem)
#Submit tasks in batches
ES_NPU_SubmitFlexibleTask (task, 1, stream);
# Call runtime to release memory after the task is completed
ES_NPU_ReleaseTaskMemory(task.modelId, stream, 1, &taskMem);
```

Output post-processing

The sample code uses the Determination Out operator provided 6.1.2. At the same time, configure the es_yolov3_post_process.json file of the post-processing operator information. The json example and precautions of this operator are as follows:

```
{
"config_version": "1.0.0",
"kernel_name": "detection_out",
"detection_net": "yolov3_u",
"in_tensor_num": 3,
"out_tensor_num": 2,
    "input_shape": "[1,255,13,13],[1,255,26,26],[1,255,52,52]",
    "output_shape": "[1,7,1,1024],[1,1,1,1]",
    "input_data_type": "S8,S8,S8",
    "output_data_type": "F32,I32",
    "anchors_num": 3,
    "anchor_scale_table": "[116,90, 156,198, 373,326], [30,61, 62,45, 59,119], [10,13, 16,30, 33,23]",
    "cls_num": 80,
    "img_h": 416,
    "img_w": 416,
    "input_scale": "[0.18310,0.14447,0.15155]",
    "nms_method": "hard_nms",
    "iou_method": "iou",
"out_box_type": "xminyminxmaxymax",
"coord_norm_flag": false,
"max_boxes_per_class": 200,
"max_boxes_per_batch": 500,
"score_threshold": 0.3,
"iou_threshold": 0.3,
"soft_nms_sigma": 0.6,
"effec_img_offset_y": 0,
"effec_img_offset_x": 0
}
```

Note:

1. For detailed information about detection_net in the configuration jso n, please refer to Section 3.4.4 ES_DET_NETWORK_E enumeration in the ENNP Developer Manual;

2. The input_shape in the configuration jso n needs to be arranged in ascending order according to the tensor size, and the configuration order of anchor_scale_table and input_scale should be consistent with input_shape;

3. The output_shape in the configuration json is the shape of the two outputs, outBoxInfo and outBoxNum. The shape represents the NCHW dimension information in turn. The "7" in the shape of the first tensor means that each output detection box has batch id , class id , etc. information; "1024" means that the maximum output of this inference is 1024 detection box (this value can be customized by the user); the second tensor The stored value indicates the actual number of boxes in each batch of this inference. Assume that 8 For batch reasoning, the shape can be configured as "[8,1,1,1] " . The value of the stored ith number is the actual number of boxes in the ith image. For more information, please refer to the description of the ES_AK_DSP_DetectionOut interface in Section 3.3.12 of the ENNP Developer's Manual.

4. The value of anchor_scale_table can generally be the above parameters. If the network is fine-tuned or retrained, it needs to be replaced with its own anchor_scale. The specific value can be obtained from the uncut onnx for example, the anchor scale of the following model is:

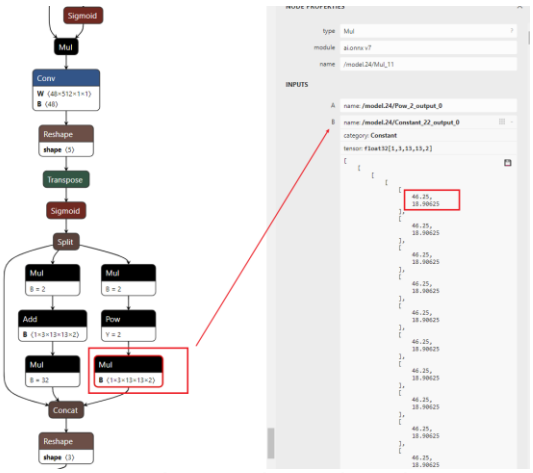

Figure 6 -2 anchor_scale Get location example

5. The value of input_scale needs to be obtained from onnx through the size of input_shape to obtain the corresponding node name, and then the quantization coefficient of the corresponding node is obtained from table.json obtained by quantizing the model.

6. For detailed information about nms_method in the configuration jso n, please refer to Section 3.4.5 ES_NMS_METHOD_E in the ENNP Developer Manual. enumerate;

7. For detailed information about the iou_method in the configuration jso n, please refer to Section 3.4.6 ES_IOU_METHOD_E enumeration in the ENNP Developer Manual.

```
# Fill post-processing operator inputs based on model inference results
for (ES_S32 tensorId = 0; tensorId < task.outputFdNum; tensorId++) {
NPU_TENSOR_S tensor;
   ES_TENSOR_S esTensor;

   ret = ES_NPU_GetOutputTensorDesc(task.modelId, tensorId, &tensor);
   if (ret != ES_SUCCESS) {
      SAMPLE_NPU_DEBUG(ret, "ES_NPU_GetOutputTensorDesc failed\n");
      return ret;
   }

   esTensor.pData = task.outputFd[tensorId];
   esTensor.dataType = postProcess->convertDataType(tensor.dataType);
   if (processType == EsPostProcess::CLASSIFY) {
      esTensor.shapeDim = 6;
   } else if (processType == EsPostProcess::DETECTION_OUT) {
      esTensor.shapeDim = 5;
   } else {
      esTensor.shapeDim = 5;
   }
   esTensor.shape[0] = tensor.dims.n;
   esTensor.shape[1] = tensor.dims.c;
   esTensor.shape[2] = tensor.dims.h;
   esTensor.shape[3] = tensor.dims.w;
   esTensor.shape[4] = 1;
   if (processType == EsPostProcess::CLASSIFY) {
      esTensor.shape[5] = esTensor.shape[4] * esTensor.shape[1];
   }
   postTensors.push_back(esTensor);
}
std::sort(postTensors.begin(), postTensors.end(), [](ES_TENSOR_S &t1, ES_TENSOR_S &t2) {
   return t1.pData.size < t2.pData.size;
});
memset(&output, 0x00, sizeof(output));
memset(&outputCount, 0x00, sizeof(outputCount));
output.shapeDim = 5;
output.shape[0] = 1;
…
outputCount.shapeDim = 5;
outputCount.shape[0] = mDetectOutConfigs.inputShape[0][0];
outputCount.shape[1] = 1;
…
ES_AK_DSP_DetectionOut(
            inTensors.data(), inTensors.size(), output,outputCount,
            (ES_DET_NETWORK_E)mDetectOutConfigs.detectNet, mDetectOutConfigs.anchorsNum,
            mDetectOutConfigs.anchorScale.data(), mDetectOutConfigs.imgH,
      mDetectOutConfigs.imgW, mDetectOutConfigs.clsNum,
            mDetectOutConfigs.inputScale.data(),
      (ES_NMS_METHOD_E)mDetectOutConfigs.nmsMethod,
            (ES_IOU_METHOD_E)mDetectOutConfigs.iouMethod,
      (ES_BOX_TYPE_E)mDetectOutConfigs.outBoxType,   (ES_BOOL)mDetectOutConfigs.coordNorm,
            mDetectOutConfigs.maxBoxesPerClass,
            mDetectOutConfigs.maxBoxesPerBatch, mDetectOutConfigs.scoreThreshold,
      mDetectOutConfigs.iouThreshold, mDetectOutConfigs.softNmsSigma,
            mDetectOutConfigs.effecImgOffsetX, mDetectOutConfigs.effecImgOffsetY);
```

For complete code, please refer to the customer development kit board sample directory. For

compilation and running methods, please refer to the board sample /src/ README .md file.

NPU After the sample _npu source code provided by Runtime is compiled into an executable file, the user needs to copy it to the EIC 7700 development board and run it. In addition, the EIC 7700 development board integrates the sample _npu executable file by default. The following is the execution command and execution result of running the sample _npu that comes with the development board:

```
eswin@debian:~# /opt/eswin/bin/sample_npu -s 2 -m /opt/eswin/data/npu/yolov3/ -i /opt/eswin/data/npu/yolov3/input/pictures
/opt/eswin/data/npu/yolov3/input/pictures/input0/bus.jpg: box count(5)
class_id: 0.000000( ('person')), score: 0.843750, x_min: 26.156250, y_min: 153.250000, x_max: 123.125000, y_max: 348.000000
class_id: 0.000000( ('person')), score: 0.838379, x_min: 343.500000, y_min: 146.500000, x_max: 413.750000, y_max: 344.250000
class_id: 0.000000( ('person')), score: 0.827637, x_min: 118.812500, y_min: 160.000000, x_max: 174.625000, y_max: 325.500000
class_id: 5.000000( ('bus')), score: 0.791504, x_min: 8.250000, y_min: 89.187500, x_max: 407.750000, y_max: 283.750000
class_id: 0.000000( ('person')), score: 0.367432, x_min: 0.000000, y_min: 163.500000, x_max: 39.468750, y_max: 391.500000
/opt/eswin/data/npu/yolov3/input/pictures/input0/dog.jpg: box count(3)
class_id: 16.000000( ('dog')), score: 0.874023, x_min: 71.500000, y_min: 171.750000, x_max: 168.500000, y_max: 383.500000
class_id: 1.000000( ('bicycle')), score: 0.787598, x_min: 64.625000, y_min: 98.625000, x_max: 308.500000, y_max: 296.500000
class_id: 7.000000( ('truck')), score: 0.608887, x_min: 254.750000, y_min: 53.312500, x_max: 374.000000, y_max: 122.687500
```

The categories of each column (separated by commas) printed above are: classification information, confidence, upper left corner coordinate X, upper left corner coordinate Y, lower right corner coordinate X and lower right corner coordinate Y.

Note: If the -o parameter is added when executing sample, sample _npu will also save the processed framed image in the path set by the -o parameter.

7. ENNP Dual-Die Model Inference Architecture

7.1 Dual Die Chip

- A dual-die chip has two identical hardware units on the SoC, which effectively doubles the hardware performance.
- The dual-die SoC operates under the same operating system, and the hardware addresses are linearized within the system, allowing the CPU to access them directly.

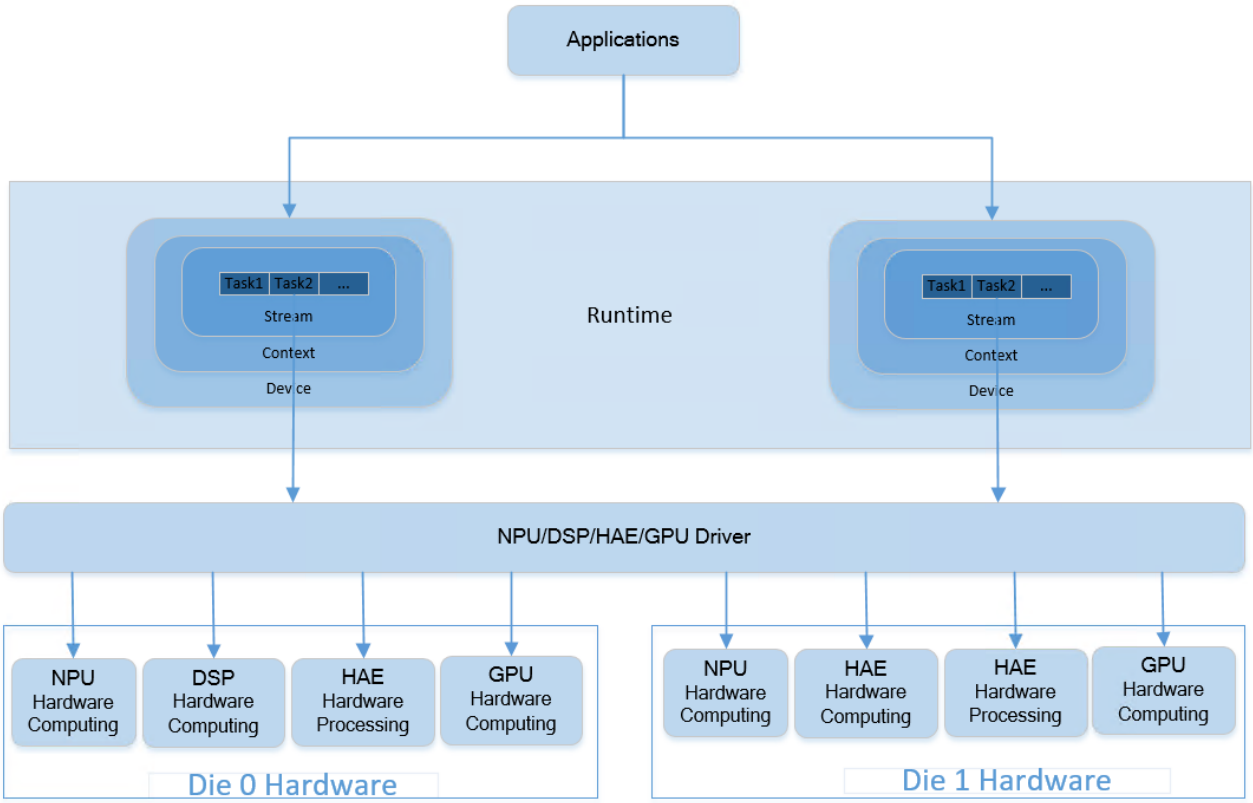

Figure 7-1-Die NPU architecture

7.2 Application Scenarios

- With the addition of an extra hardware unit in a dual-die environment, overall performance is significantly improved, allowing the NPU to handle more tasks for inference.
- In a dual-die environment, cross-die data access performance may be affected. Therefore, it is recommended to minimize cross-die data transfer during system operation to enhance overall efficiency.
- For performance benchmarking scenarios, batch inference tasks can be evenly distributed across both dies, significantly improving performance scores.

7.3 Dual-Die Inference Framework

- To avoid the performance impact of cross-die data access, the NPU performs inference completely in parallel on both dies, with no die-to-die data interaction in between.
- NPU Runtime assigns an NPU Context for each pipeline, corresponding to different NPU dies.
- The NPU driver mounts two device nodes based on the current dual-die environment and creates two device data structures to store data for each die separately.
- The NPU driver communicates with the hardware (E31) on each respective die for task distribution and inference processing.

8. Custom DSP operator development example

This chapter introduces the complete operation process of how users can customize the special operators required by the model based on DSP hardware. The complete custom DSP operator development process is as follows:

- Build E SWIN cross-compilation environment;
- Build the Cadence Vision Q7 DSP compilation environment;
- Develop custom dsp operators;
- Compile custom developed DSP operators;

- The DSP operator runs on the board.

Note: For more detailed development guide, please refer to README.md and sample examples in dsp_sample/kernel_sample.

8.1 Sample file description

ESWIN provides dsp_sample/kernel_sample samples to help users quickly get started with custom operator development. The specific directory structure is as follows:

```
kernel_sample
├─── kernels
│   ├─── bev_pool
│   │   ├─── bev_pool.c
│   │   ├─── bev_pool.h
│   │   └─── CMakeLists.txt
│   ├─── build.sh
│   ├─── CMakeLists.txt
│   ├─── include
│   │   ├─── common
│   │   └─── xtensa
│   └─── user_add
│   ├─── CMakeLists.txt
│   ├─── user_add.c
│   └─── user_add.h
├─── README.md
└─── test
├─── build.sh
├─── CMakeLists.txt
├─── data
│   ├─── input_0.bin
│   └─── input_1.bin
└─── src
├─── common.c
├─── common.h
└─── dsp_kernel_test.c
```

It mainly includes the following contents:

README.md: Detailed development and compilation guide;

kernels: dsp-side code, taking the user_add operator as an example, provides a sample example of dsp-side operator development;

test/src: host-side code, taking the call of the user_add operator as an example, provides an example of calling a custom operator;

test /data: In binary form, it stores the input data required by useradded, which users can use to complete custom operator sample self-test.

8.2 Build ESWIN cross-compilation environment

There are two user compilation environments: 20.04 and 22.04. The difference lies in the generation of the cross-compilation directory rootfs. 20.04 can only use the docker compilation provided by eswin to generate rootfs, while 22.04 can directly compile and generate rootfs on x86.

The following operations are based on Ubuntu 20.04.

8.2.1 Get docker and tar.gz

Get the es_debian docker image and eswin-sdk-20241024.tar.gz (or latest version) from ESWIN.

8.2.2 Compile eswin cross-compilation environment

- Refer to the "3.3 System Compilation" section in the "Software Getting Started Guide_CN_v1.3.docx" and execute until the "make_all" command is completed.
- Execute on the Host

```
cd eswin-sdk-20241024/
sudo chmod 777 -R EIDS200B/
cd EIDS200B/output
mkdir rootfs
sudo apt-get install e2fsprogs
e2fsck -f root-EIDS200B-20241025-020530.ext4
resize2fs root-EIDS200B-20241025-020530.ext4 12G
sudo mount root-EIDS200B-20241025-020530.ext4 rootfs/

sudo chroot rootfs/
```

- Execute in rootfs

```
root@ibudev12:/# cd home/eswin/
root@ibudev12:/# apt install -y gcc build-essential cmake
root@ibudev12:/# apt install es-sdk-dsp
root@ibudev12:/# apt install es-sdk-sample-dsp
root@ibudev12:/# apt install es-sdk-common
root@ibudev12:/# cd /usr/include/
root@ibudev12:/usr/include# ln -s riscv64-linux-gnu/bits bits
root@ibudev12:/usr/include# ln -s riscv64-linux-gnu/sys sys
root@ibudev12:/usr/include# ln -s riscv64-linux-gnu/gnu gnu
root@ibudev12:/usr/include# ln -s riscv64-linux-gnu/asm asm
root@ibudev12:/usr/include# cd /usr/lib
root@ibudev12:/usr/lib# cp riscv64-linux-gnu/* ./ -rf
root@ibudev12:/usr/lib# exit
```

8.2.3 Obtain the RISC-V Cross-Compilation Tools

```
wget https://github.com/riscv-collab/riscv-gnu-toolchain/releases/download/2024.04.12/riscv64-glibc-ubuntu-20.04-gcc-nightly-2024.04.12-nightly.tar.gz
sudo tar -xvf riscv64-glibc-ubuntu-20.04-gcc-nightly-2024.04.12-nightly.tar.gz -C /opt
sudo cp /opt/riscv/sysroot/usr/lib/crt1.o eswin-sdk-20241024/EIDS200B/output/rootfs/usr/lib
```

Note: The above opt The directory needs to be consistent with the script in the subsequent host compilation process.

8.3 Cadence Vision Q7 DSP compilation environment construction

EIC7700 integrates 4 DSP cores and uses Cadence's Vision Q7 DSP IP. For the installation of

Cadence DSP development environment, please refer to the official document *dev_tools_install_guide.pdf* provided by Cadence.

Note: DSP development relies on the integrated development environment provided by Cadence. You need to set up the license correctly before you can use Xtensa Develop Tools. EIC7700 does not provide Cadence license, which needs to be purchased separately.

8.4 Custom operator development

8.4.1 Host side preparation

The user needs to complete the generation of op desc, memory allocation and data loading of inputs and outputs on the host. The remaining steps generally do not need to be modified.

- The op desc filling method refers to the prepare_add_op_desc() function implementation in test/src/dsp_kernel_test.c;
- The memory allocation and data loading of inputs refer to the loadInputsData() function in test/src/dsp_kernel_test.c;
- For the memory allocation of outputs, refer to the prepareOutputBuffer() function in test/src/dsp_kernel_test.c.

8.4.2 DSP side operator development

This manual briefly describes the common interfaces for DSP-side operator development. For more detailed development guidelines and instructions, please refer to README.md in dsp_sample/kernel_sample and the provided sample examples.

- DSP side operator function interface

On the dsp side, there are four functions: prepare, eval, get_prepare, and get_eval. Their function names and parameters are fixed interfaces and cannot be modified. At the same time, the contents of the three functions: prepare, get_prepare, and get_eval cannot be modified. Users only need to complete the implementation code of their own operators in the eval function.

- Get the starting address and size of the on-chip memory

```
char *dram0_base_ptr = (char *)((ES_U32)get_available_dram0_base());
ES_U32 dram0_size = get_available_dram0_size();
```

Notice:

1.When developing DSP, it is not recommended to directly access DDR memory, which may cause data inconsistency due to cache. In addition, large amounts of DDR memory access will greatly slow down the calculation performance of the operator. It is recommended to use DMA to copy data to DRAM for operation;

2. Each DSP has two on-chip memories: dram 0 and dram 1. The original size of both drams is 128k. Since the firmware occupies part of the space, the current starting address of dram 0 is 0x2810 5000, and the available space is 108k; the starting address of dram 1 is 0x281 20000, and the available space is 120k.

- Initiate dma copy

```
idma_copy_3d_desc64(CHANNEL, &dst_ptr, &src_ptr, IDMA_DISABLE_INT, row_sz, nrows, ntiles,
src_row_pitch, dst_row_pitch, src_tile_pitch, dst_tile_pitch);
```

- Waiting for dma copy to end

```
while (idma_hw_num_outstanding(CHANNEL) > 0) {
}
```

- Log Printing

On the dsp side, please use the es_printf function to print the log.

```
es_printf("this is eval of add op\n");
```

Notice:

1. Please log in to the serial port and check the print log on the DSP side, for example (the following commands are for reference only, please adjust the specific commands according to the connection method of your own development board):

```
ssh 192.168.1.100 -l hostname
Enter password: password
sudo hostname -D /dev/ttyUSB3
```

After logging in to the serial port, execute the executable program to call the operator, and you can get the content printed by es_printf on the serial port.

2.Using es_printf to print logs on the DSP side will have a significant impact on operator performance. Depending on the length of the printed log, it will roughly increase the time consumption by tens to hundreds of microseconds.

- DSP performance optimization

1.Use DMA to copy data from DDR to DRAM for calculation, avoiding large-scale access to DDR;

2.It is recommended to use the int8 data type first, followed by the int 16 and f 16 data types, and avoid using float32 or double data types as much as possible;

3.Use the S IMD instruction provided by DSP to optimize the calculation speed;

4.Use timestamps to count the time consumed by each part and optimize the performance bottlenecks.

8.5 Operator compilation

Refer to README.md in dsp_sample/kernel_sample. Before compiling the operator, modify the "xtensa configuration" in the "xtensa_env_setup" function in "dsp_sample/kernel_sample/kernels/build.sh" according to the actual situation.

Note: Since XTENSA_CORE and XTENSA_TOOLS_VERSION may differ, please communicate and resolve any compilation issues in a timely manner.

- Compile the test program on the host side

```
cd /path/to/your/dsp_sample/kernel_sample/test
./build.sh /path/to/your/eswin-sdk-20241024/EIDS200B/output/rootfs /path/to/your/kernel_sample/kernels/user_add
```

- Compile custom operators on the dsp side

```
cd /path/to/your/dsp_sample/kernel_sample/kernels/
./build.sh /path/to/your/eswin-sdk-20241024/EIDS200B/output/rootfs
```

8.6 Custom operator testing

- Copy the compiled operator es_dsp_kernel_add.pkg.lib to the development board/lib/firmware/dsp_kernels/;
- Copy the test program dsp_kernel_test and the test files input_0.bin, input_1.bin to any location on the development board, such as: /opt/eswin/;
- Execute the test command:

```
cd /opt/eswin/
./dsp_kernel_test -c 0 -i input_0.bin input_1.bin
```

- Correctness verification:

After execution, the output. bin will be generated in the directory where input_0.bin is located. Copy it to Linux and use:

```
md5sum output.bin
```

Check its md5 If the value of the code is: 4aae41d770845ae89f0cf33be15d0a77, the result is correct.

9. appendix

Table 9-1operators supported by DSP

Operator Name	Operator Function

Add	Supports addition of two inputs
ArgMax	Find the maximum value and its index of the input
AveragePool	the average of the data within the kernel range
BatchNormalization	Perform BatchNormalization on the input
CropAndResize	Extract a specific area from the input image and scale it
Concat	Concatenate multiple inputs on the specified axis
Conv	Perform convolution on the input
Cos	Calculate the Cos value of the input
ConvTranspose	Deconvolution of the input
DepthwiseConv	Perform a depthwise convolution on the input
Focus	Support Focus structure in yolo detection network
Gather	data corresponding to indices from the input data
Gelu	Calculate the input Gelu Activation function output
Gemm	Perform matrix multiplication on the input data
GlobalAveragePool	Calculate the mean of each channel of the input data
HardSigmoid	Calculate the HardSigmoid activation function output of the input
HardSwish	Calculate the HardSwish activation function output of the input
InstanceNormalization	Perform InstanceNormalization on the input
LayerNormalization	Perform LayerNormalization on the input
LeakyRelu	Calculate the output of the LeakyRelu activation function for the input
Log	the input Log Numeric
MatMul	Matrix multiplication of two data matrices
MaxPool	Find the maximum value of the data within the kernel range
Mish	Calculate the Mish activation function output of the input
NonMaxSuppression	Perform NMS operations on the boxes of the detection network
Pad	For input at different dims Perform Pad operations on
PRelu	Calculate the PRelu activation function output of the input
ReduceMax	Find the maximum value of the input along the specified dimension

ReduceSum	Calculates the sum of the input along the specified dimension
Relu	Calculate the Relu activation function output of the input
Relu6	Calculate the Relu6 activation function output of the input
Resize	Scale the input tensor data
RoiAlign	Perform bilinear interpolation on the input region of interest
MaxRoiPool	Execute MaxRoiPool pooling operation
ROPE	Insert position information before the attention module of the LLM model
Sigmoid	Calculate the Sigmoid activation function output of the input
Silu	Calculate the Silu activation function output of the input
Sin	Calculate the input Sin Numeric
Slice	Convert a single input to an attribute Information is split into multiple outputs
Softmax	Calculate the Softmax of the input as output
TopK	K largest values and their indices for the input in the specified dimension
Transpose	According to the perm parameter, transpose the input