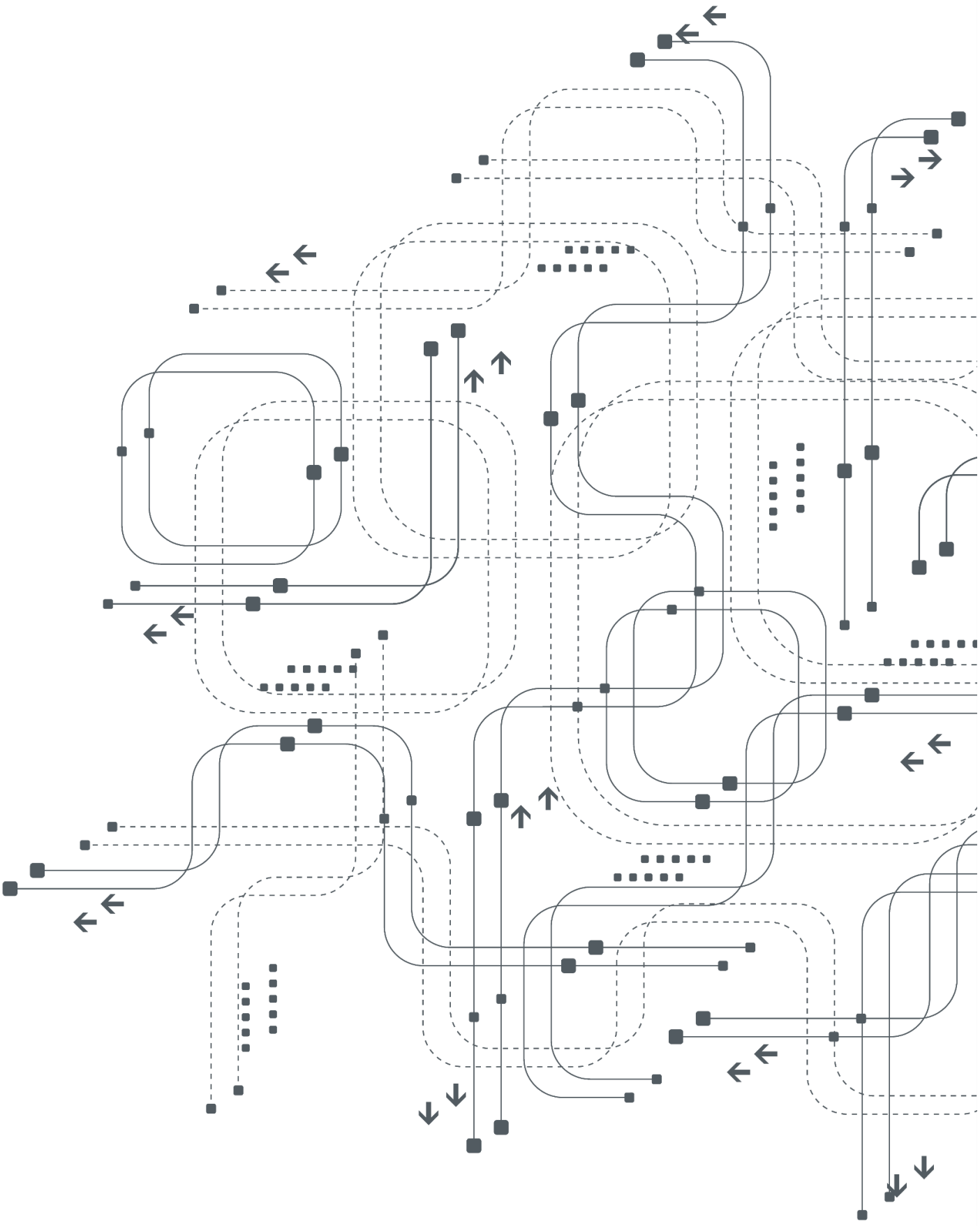


Tachy-Compile
Guidebook

Agenda

1. Overview

1-1) What is the Tachy Compile?

1-2) Key Features and Advantages

2. Format Optimization Flow

2-1) Model Transformation Overview

2-2) ONNX conversion process → Tachy format conversion

3. Use Example Guide

〈YOLOv9 Model Learning〉

3-1) Preparedness

3-2) Main Training

3-3) Verifying Learning Results

3-4) Optimization Training (optional)

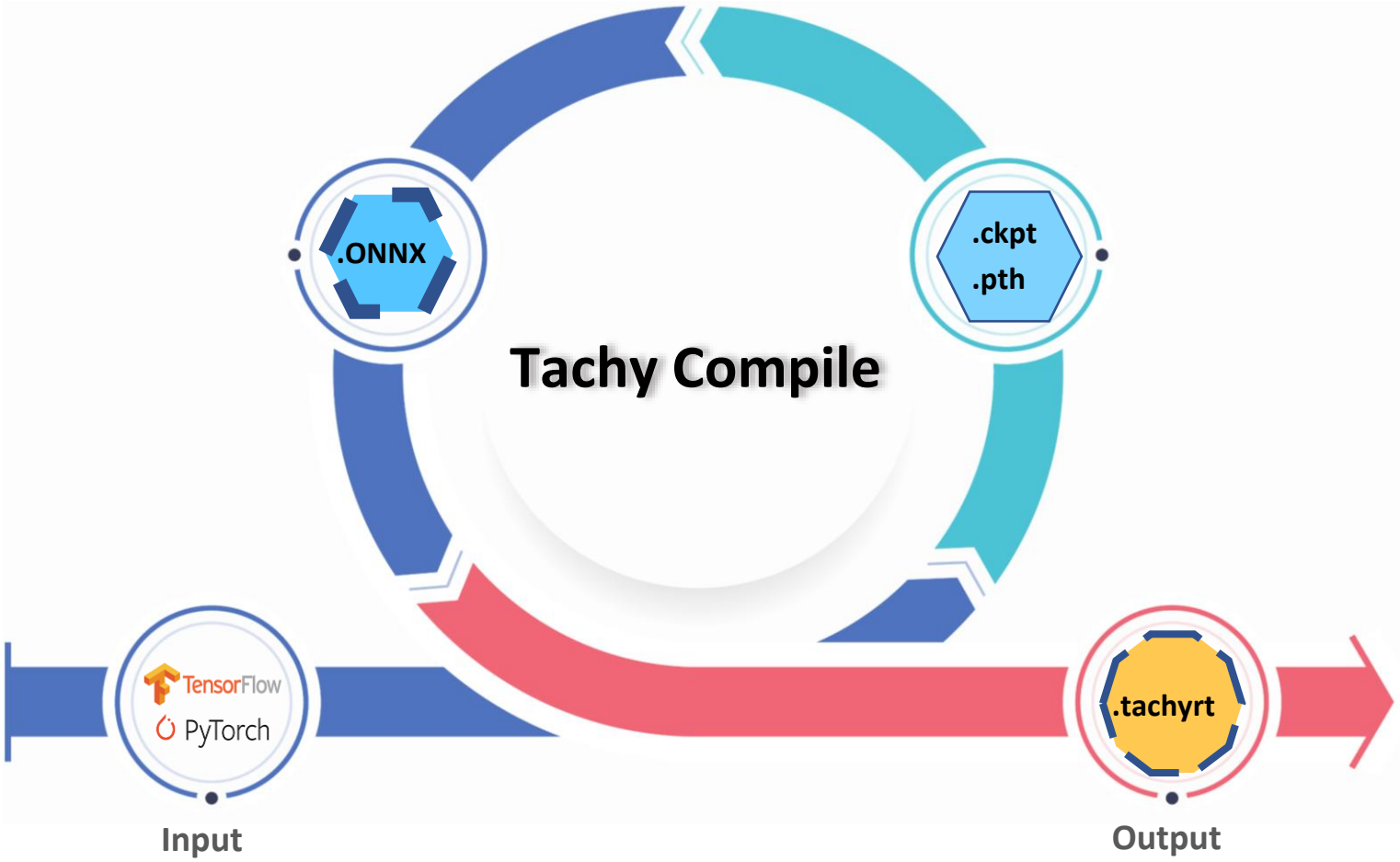
3-5) Model Transformation (PyTorch → ONNX → TACHY)

1. Overview

1-1) What is the Tachy Compile?

Tachy Compile is a compiler that transforms models learned from various deep learning frameworks (TensorFlow, PyTorch, etc.) to be optimized and run in high-performance NPU environments. Specifically, by standardizing the model around the Open Neural Network Exchange (ONNX) format and converting it into a TACHY-only format (.tachy), it supports lightweight, high speed, and real-time inference.

<Format Optimization Flowchart>



1-2) Key Features and Advantages

→ Consolidate learning, transformation, optimization, and deployment processes into one flow

→ Graph optimization and memory weight reduction

→ Support and compatibility of a variety of platforms

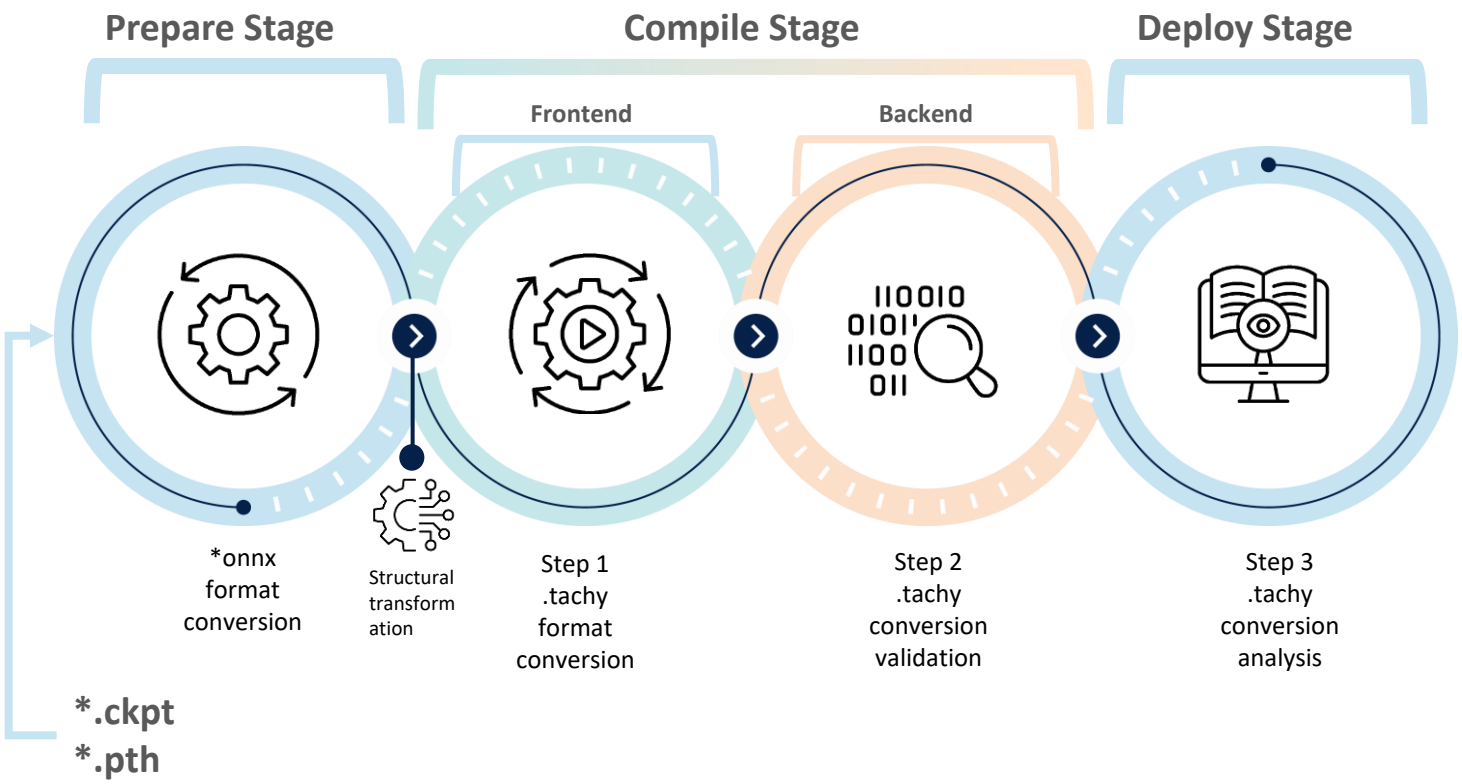
→ Optimization for NPU Application

2. Format Optimization Flow

2-1) Model Transformation Overview

The User prepares a pre-trained model with XWN in TensorFlow or PyTorch, converts it into ONNX format, and finally creates a .tachy file through the Tachy Compiler to optimize the file for application to Deeper-I NPU boards.

2-2) ONNX conversion process → Tachy format conversion



After converting the learned model to ONNX, the ONNX modifier performs operator structure modification and unnecessary node removal. The optimized ONNX model maximizes execution efficiency through XWN lightening. The Tachy Compiler then finally reconstructs the memory structure and operation flow and converts it into .tachy format.

3-1) Preparedness

☐ **Building a Learning Environment**

Check GPU and CUDA compatibility

```
$ nvidia-smi
```

Check the CUDA version that your GPU supports.

☐ **Preparing Dataset**

```
{dataset_path}/
  ├── train/
  │    ├── images/
  │    └── labels/
  ├── val/
  │    ├── images/
  │    └── labels/
  └── test/
       ├── images/
       └── labels/
```

☐ **Label Format (YOLO format)**

```
$ class x_center y_center width height
```

3-2) Main Training

☐ **Create Dataset YAML File**

```
$ git clone http://asdf.com
$ cd yolov9
```

```
$ cd data
$ vi {DATASET_NAME.yaml}
```

※ Create a New File

```
path: /mnt/DPI-SHARED/tom/20240625_yolov9/dataset/FP # dataset root dir
train: train/images
val: val/images
test: test/images

# Classes
names:
  0: person
  1: car
  2: bus
  3: truck
```

☐ **Create a Model YAML File**

```
$ cd ../models/detect
$ cp bsnet-t.yaml {bsnet-t-DATASET_NAME.yaml}
$ vi {bsnet-t-DATASET_NAME.yaml}
```

modification: Correction: Change the nc (number of classes) value to match the dataset

```
# YOLOv9

# parameters
nc: 4  # number of classes
```

Example {bsnet-t-DATASET_NAME.yaml}

☐ **Set up a Learning Script**

```
$ cd ../..
$ vi train      ※ Create a New File
```

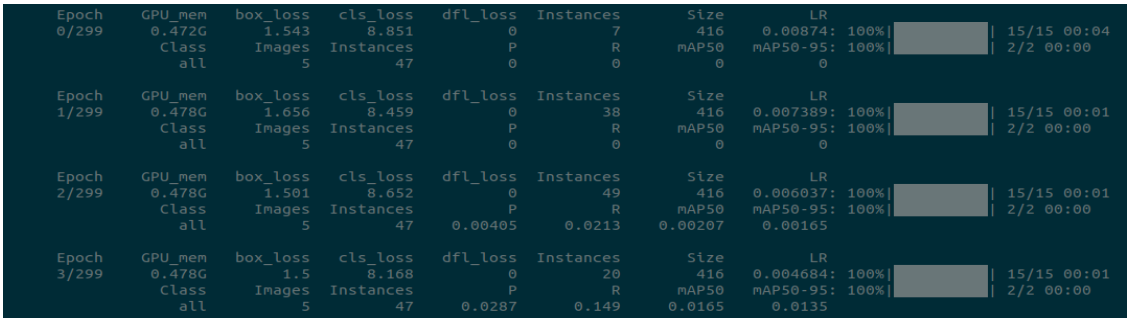
ex) train command - open the train file, paste the code at the bottom

```
python train.py --workers 8 \
        --device 0 \
        --batch 8 \
        --data data/{DATASET_NAME.yaml} \
        --img 416 \
        --cfg models/detect/bsnet-t-{DATASET_NAME.yaml} \
        --weights '' \
        --name {guide} \
        --hyp hyp.pretrained.yaml \
        --min-items 0 \
          --epochs 300 \
        --close-mosaic 15
```

☐ **Run Learning**

```
$ ./train      Execute a file
```

Result: Create running/train/{guide}/weights/best.pt



3-3) Verifying Learning Results

☐ **Setting up and running verification scripts**

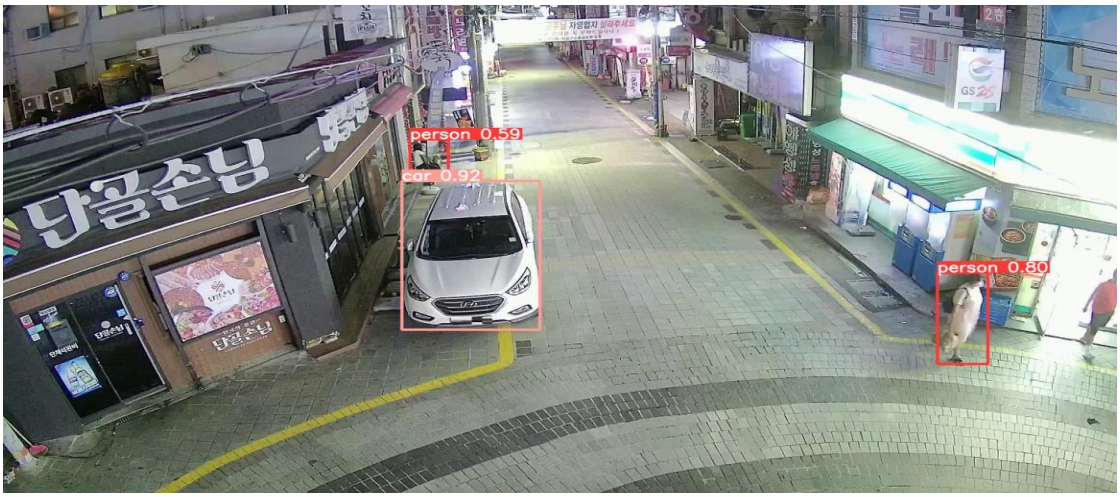
```
$ vi test      ※ Create a New File
```

ex) test command - open test file, paste code at the bottom

```
python detect.py --source {dataset_path}/test/images \
        --img 416 \
        --device 0 \
        --weights runs/train/{guide}/weights/best.pt \
        --name {guide} \
        --save-txt
```

```
$ ./test      Execute a file
```

Results: Save detection results in the runs/detect/{guide} directory



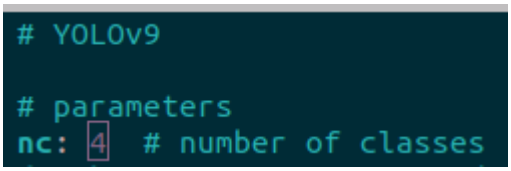
Example detect image

3-4) Optimization Training (optional)

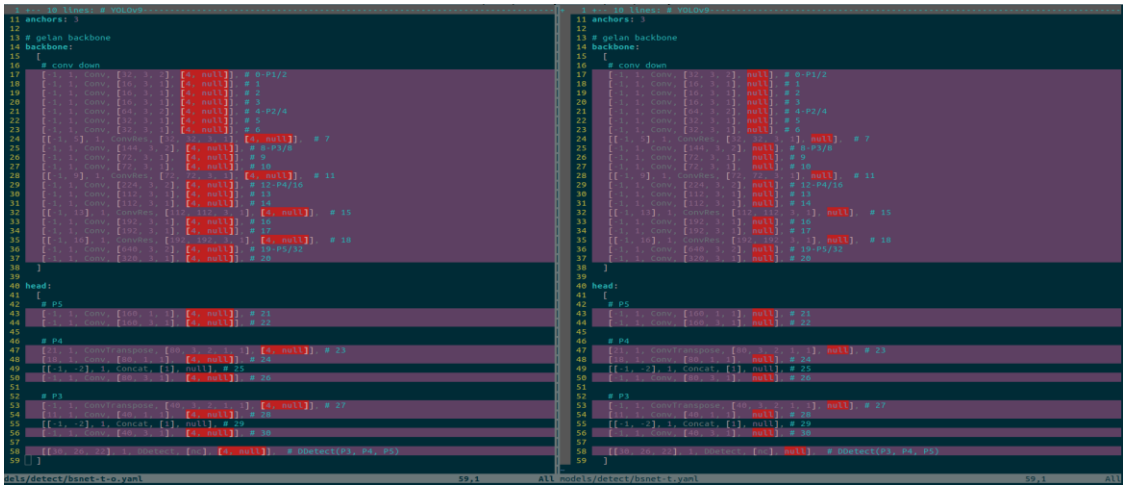
(Optional: When conducting quantization/optimization training)

☐ **Create Model YAML for Optimization**

```
$ cd ../models/detect
$ cp bsnet-t-o.yaml {bsnet-t-o-DATASET_NAME.yaml}
$ vi {bsnet-t-o-DATASET_NAME.yaml}
```



Example {bsnet-t-o-DATASET_NAME.yaml}



diff main training model.yaml, opt training model.yaml

opt training differ bit : int / 4 / Quantization range (bit-1 ** 2)

☐ **Setting Up Optimization Learning Scripts**

```
$ cd ../..
$ vi train      ※ Create a New File
```

ex) train command - open the train file, paste the code at the bottom

```
python train.py --workers 8 \
        --device 0 \
        --batch 8 \
        --data data/ {DATASET_NAME.yaml} \
        --img 416 \
        --cfg models/detect/ {bsnet-t-o-DATASET_NAME.yaml} \
        --weights 'runs/train/{guide}/weights/best.pt' \
        --name {guide-opt} \
        --hyp hyp.optimization.yaml \
        --min-items 0 \
          --epochs 100 \
        --close-mosaic 15
```

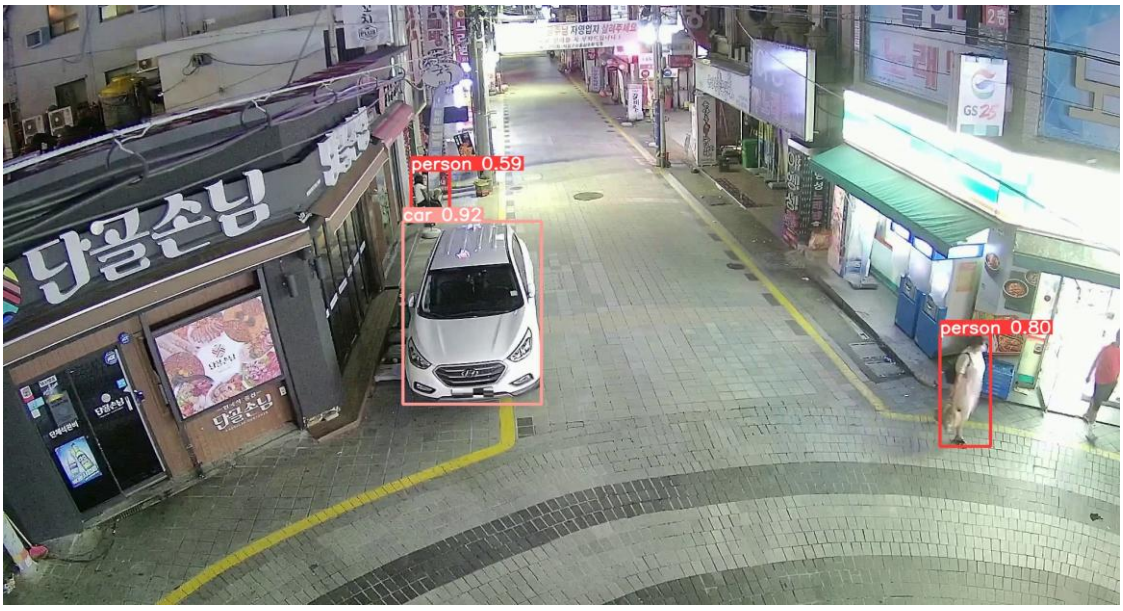
* Pay attention to yaml file name in --data and --cfg options

☐ **Optimization Learning and Validation**

```
$ ./train      Execute a file
```

☐ **Reverify optimized results**

```
$ vi test

python detect.py --source {/dataset_path/test/images} \

        --img 416 –device 0 \

        --weights 'runs/train/{guide-opt}/weights/best.pt' \

        --name {guide-opt} –save-txt

$ ./test      Execute a file
```



3-5) Model Transformation(PyTorch → ONNX → TACHY)

What is Compile?

A batch script that automatically compiles the yolov9 model learned by pt, etc. into a tachy runtime package (.tachyrt, param.bin, etc.) that tachy-bs npu can understand.

☐ **Modify the compile script**

Modify the compile script inside the yolov9 project.

vi compile ※ Open a file

<Explanation of key variables in the compile script> ※ Required Settings

config	default	explanation
Input path		
PRE_PARAM_DIR	runs/golden/drone/bsnet-t-drone3c/weights	Directory path where the pre-optimization model weight (.pt) is located
OPT_PARAM_DIR	runs/golden/drone/bsnet-t-o-drone3c/weights	Directory path where model weight (.pt) is located after optimization
Size		
ONNX_INPUT_SHAPE	"1 3 320 416"	Input tensor form during ONNX conversion (B, C, H, W)
TACHY_INPUT_SHAPE	"1 320 416 3"	Input tensor form for TACHY-Compiler (B, H, W, C)

< Option Settings >

config	default	explanation
TRAIN_FILE_TYPE	"pt_yolov9"	Learning File Format for Deployment Scripts (PyTorch-YOLOv9)
ONNX_FILE_TYPE	"pt2onnx"	PyTorch → ONNX Transformation Script Type
SRC_PRE_PATH	os.path.join(PRE_PARAM_DIR, "best.pt")	Full path to weight files before optimization
SRC_OPT_PATH	os.path.join(OPT_PARAM_DIR, "best.pt")	Full path to weighted files after optimization
WORK_DIR	datetime.datetime.now().strftime("20%y%m%d_%H%M%S")	The name of the working directory that is generated at the time of execution (for example, 20250429_153240)
TEMP_DIR	os.path.join(WORK_DIR, "temp")	Temporary file storage folder inside the working directory
DST_EVAL_PATH	os.path.join(WORK_DIR, "eval.pt")	PyTorch Weight Storage Path for Evaluation (Uncompressed)
DST_COMP_PATH	os.path.join(WORK_DIR, "comp.pt")	Compressed (compiled) PyTorch weight storage path
DST_TEMP_PATH	"" (Empty String)	Paths filled when temporary deliverables are needed (currently unused)
ONNX_EVAL_PATH	os.path.join(WORK_DIR, "eval.onnx")	EVAL ONNX model storage path
ONNX_COMP_PATH	os.path.join(WORK_DIR, "comp.onnx")	COMP ONNX model storage path
LAYER_EVAL_PATH	os.path.join(WORK_DIR, "eval.tachy")	Layer-level EVALTACHY file path
LAYER_COMP_PATH	os.path.join(WORK_DIR, "comp.tachy")	Layer-level COMP TACHY file path
L_DEFAULT_PAD_MODE	--default_pad_mode=dynamic	Padding Mode Setting Flag at Layer Compilation
BLOCK_COMP_PATH	os.path.join(WORK_DIR, "block.tachy")	Block-level compression TACHY file path
B_DEFAULT_PAD_MODE	--default_pad_mode=dynamic	Padding mode setting flag when compiling blocks
DEFAULT_PAD_ORDER	--default_pad_order=up_left	Specify the default direction to apply padding
LOGIT_ORDER	--logit_order 5 3 0 4 2 1	Output logit (class) order rearrangement options

☐ **Execute a file**

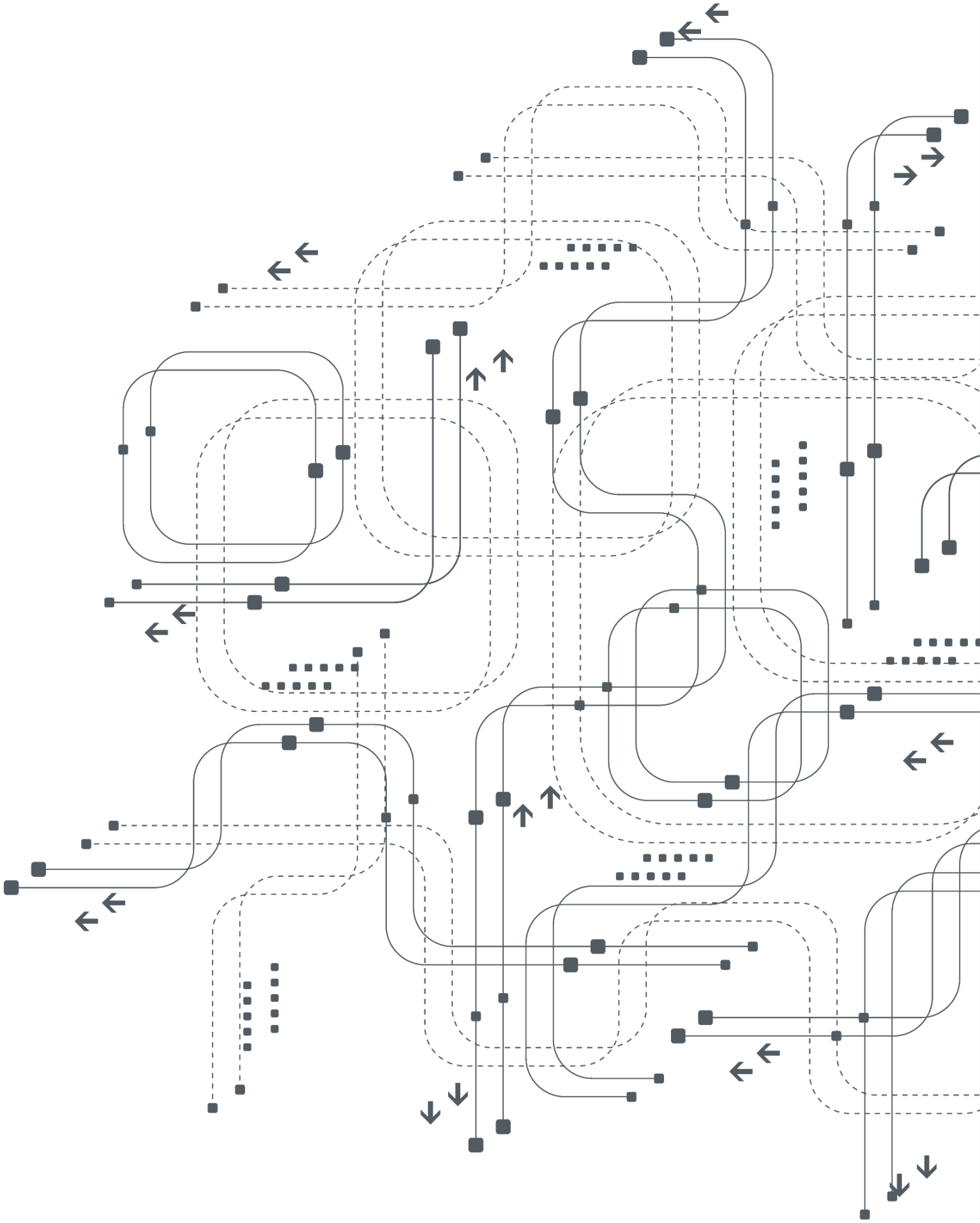
```
./compile
```

☐ **Check the results**

```
ls 20250429_152340/*
```

☐ **Example Output**

```
model_224x224x3_inv-t.tachyrt inst.bin param.bin inst.json
```



Customer Support

- Business : partner@deeper-i.ai
- website : https://www.deeper-i.ai