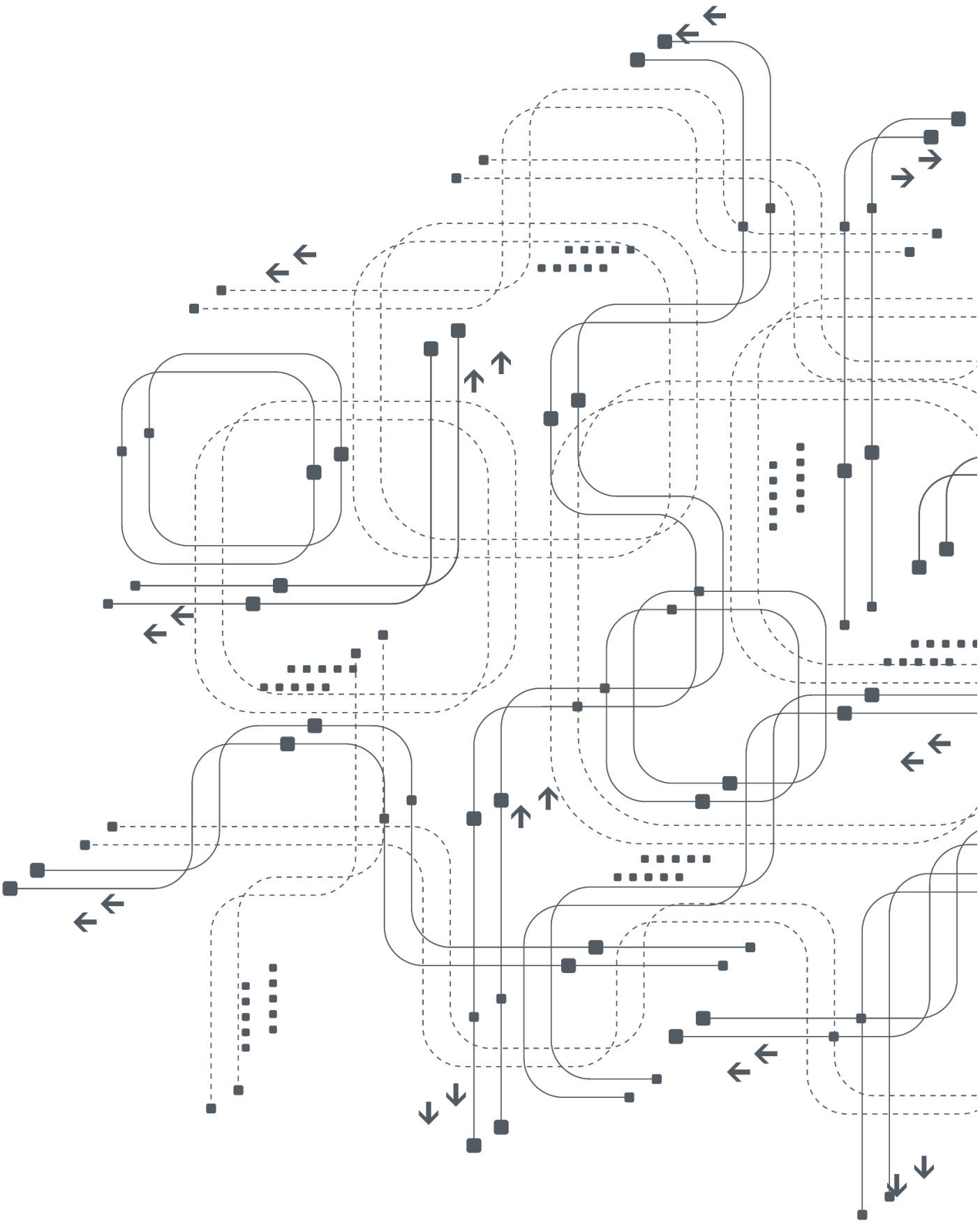


DDesigner API

Guidebook

Agenda

1. Overview

1-1) What is the DDesigner API?

1-2) Key Features and Advantages

2. Requirements

2-1) System Requirements

2-2) Pre-Installation Preparations

3. Use Example Guide

3-1) Step 1 – Learn Basic Model

3-2) Step 2 – Validation of DDesigner API interworking

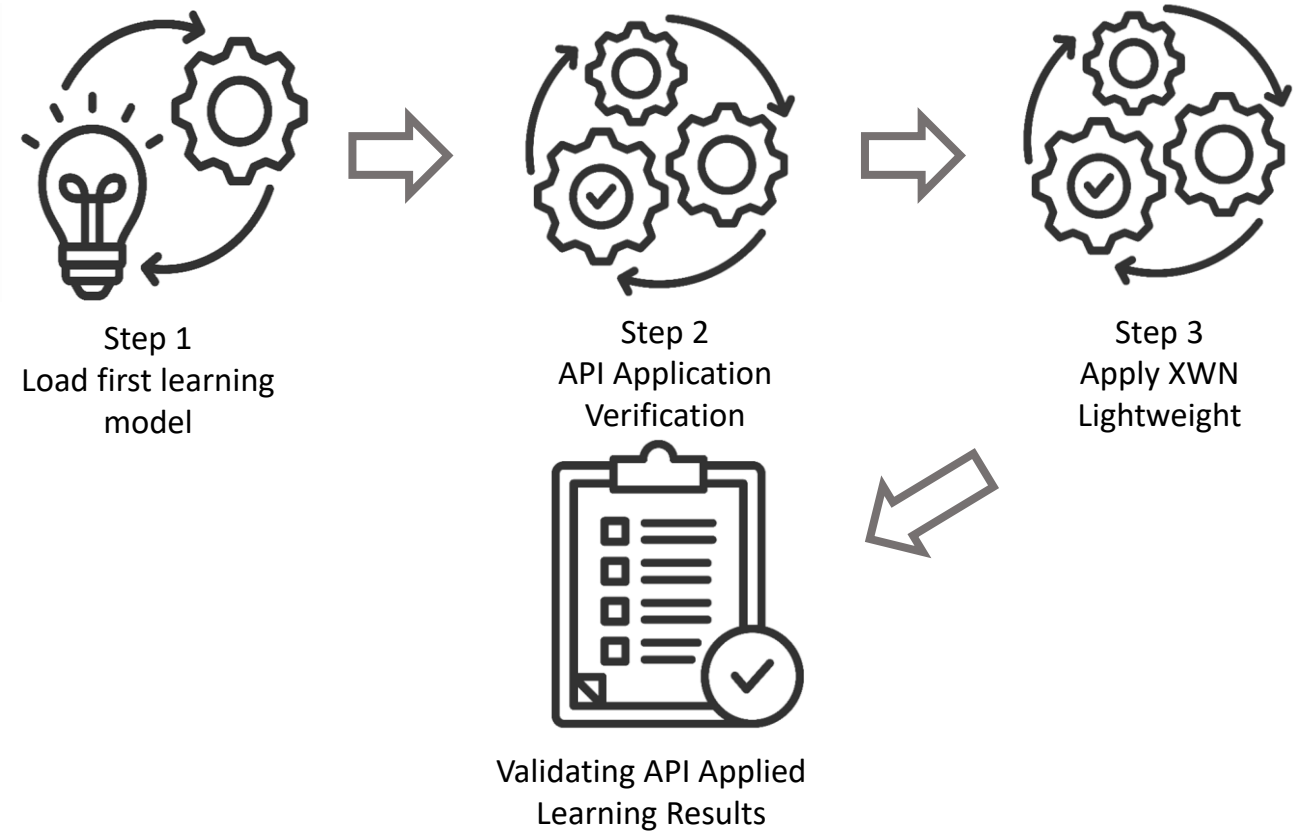
3-3) Step 3 – Learning lightweight (XWN) application

3-4) Accuracy comparison and normal operation verification criteria

1. Overview

Explain the XWN-based light weight application process when learning the learning model using the DDesigner API. The learning is divided into three stages, each of which includes verifying the success of the application through API linkage and accuracy comparison.

It also aims to help efficient learning and optimization by linking APIs in a practical environment.



1-1) What is DDesigner API?

The DDesigner API is an API that transforms and optimizes the learned model into a form suitable for Deeper-I NPU boards. It takes PyTorch and TensorFlow-based learning models as inputs and performs structural weight reduction based on the XWN.

1-2) Key Features and Advantages

→ Lightweight (XWN) support reduces the amount of computation
→ Natural integration with existing learning pipelines
→ Support for a variety of CNN structures
→ Maintain fast learning performance

2. Requirements

2-1) System Requirements

Supported models: PyTorch, TensorFlow

Input Format: *.pth, *.pt, *.ckpt

2-2) Pre-Installation Preparations

→ torch 1.13.1

→ python 3.8.5

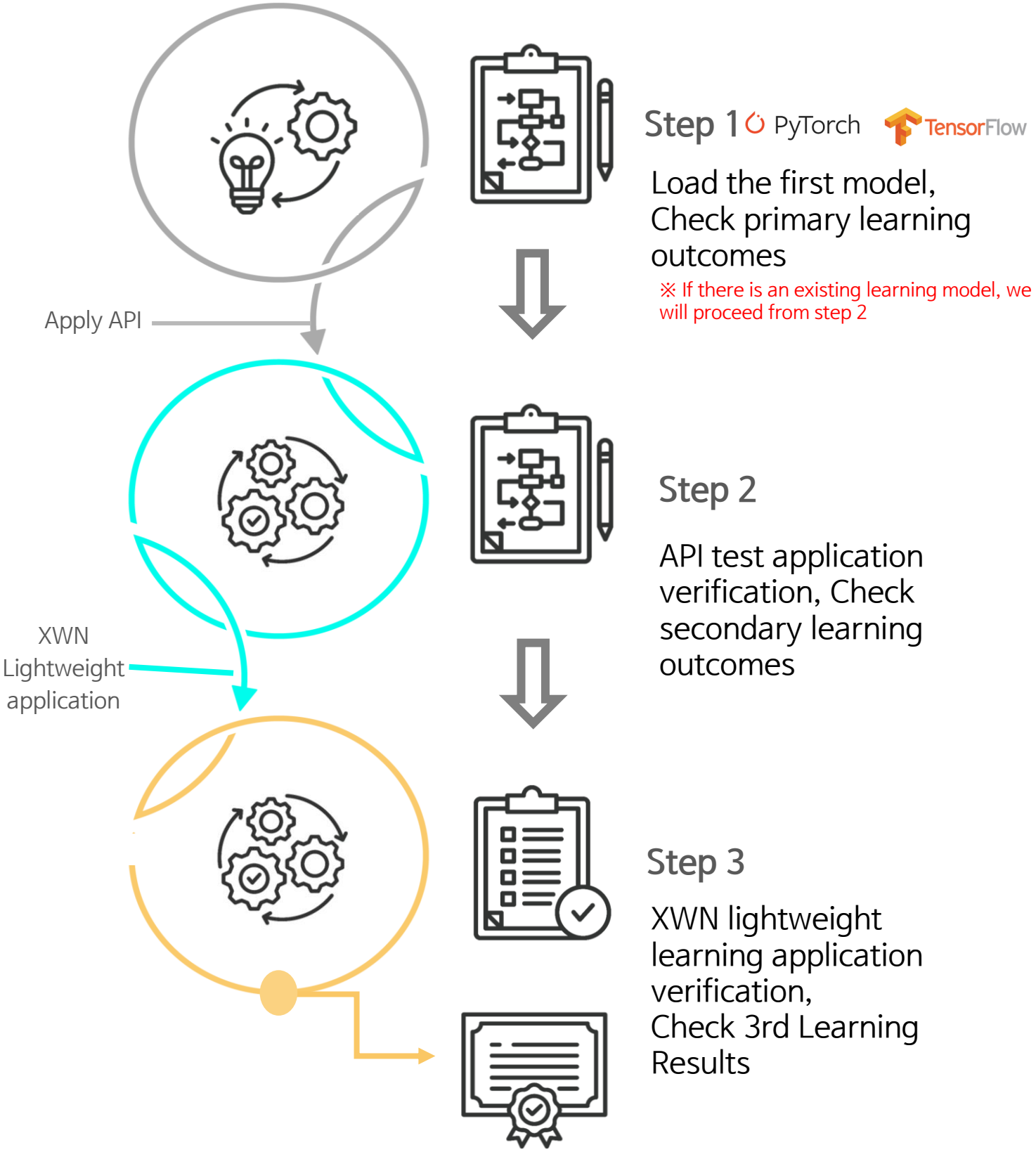
→ numpy 1.23.5

※ This guide is based on the vim environment

3. Use Example Guide

This guide describes the XWN-based lightweight application process in model training using the DDesigner API.

Training for DDesigner API application consists of a total of 3 steps



Each step verifies the success of the application through API interworking and accuracy comparison.

DDesigner API configuration settings

```
pip install DDesignerAPI
```

1) Step 1 – Learn Basic Model

※ If there is an existing learning model, we will proceed from step 2.

→ First time model loading and learning outcomes

☐ **example clone of git**

```
git clone https://github.com/kuangliu/pytorch-cifar.git
```

※ Check the installation file 'pytorch-cifer'

※ API Example Utilization Link for Pytoch – Example File 'resnet.py

'https://github.com/kuangliu/pytorch-cifar/blob/master/models/resnet.py

☐ **Directory Path Settings**

```
cd pytorch-cifar
```

※ After installation, check the presence or absence of main.py

☐ **Run vim main.py**

```
vim main.py
```

vim main.py execution screen

main.py

```python
'''Train CIFAR10 with PyTorch.'''
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torch.backends.cudnn as cudnn

import torchvision
import torchvision.transforms as transforms

import os
import argparse

from models import *
from utils import progress_bar


parser = argparse.ArgumentParser(description='PyTorch CIFAR10 Training')
parser.add_argument('--lr', default=0.1, type=float, help='learning rate')
parser.add_argument('--resume', '-r', action='store_true',
                    help='resume from checkpoint')
args = parser.parse_args()

device = 'cuda' if torch.cuda.is_available() else 'cpu'
best_acc = 0  # best test accuracy
start_epoch = 0  # start from epoch 0 or last checkpoint epoch

```

☐ vim main.py fix list

#net = ResNet18() -> net =ResNet18() - '#' to activate the model

```
classes = ('plane', 'car', 'bird', 'cat', 'deer',
           'dog', 'frog', 'horse', 'ship', 'truck')

# Model
print('==> Building model..')
# net = VGG('VGG19')
# net = ResNet18()
# net = PreActResNet18()
# net = GoogLeNet()
# net = DenseNet121()
# net = ResNeXt29_2x64d()
# net = MobileNet()
# net = MobileNetV2()
# net = DPN92()
# net = ShuffleNetG2()
# net = SENet18()
# net = ShuffleNetV2(1)
# net = EfficientNetB0()
# net = RegNetX_200MF()
net = SimpleDLA()
net = net.to(device)
if device == 'cuda':
    net = torch.nn.DataParallel(net)
    cudnn.benchmark = True
```

```
classes = ('plane', 'car', 'bird', 'cat', 'deer',
           'dog', 'frog', 'horse', 'ship', 'truck')

# Model
print('==> Building model..')
# net = VGG('VGG19')
net = ResNet18()
# net = PreActResNet18()
# net = GoogLeNet()
# net = DenseNet121()
# net = ResNeXt29_2x64d()
# net = MobileNet()
# net = MobileNetV2()
# net = DPN92()
# net = ShuffleNetG2()
# net = SENet18()
# net = ShuffleNetV2(1)
# net = EfficientNetB0()
# net = RegNetX_200MF()
net = SimpleDLA()
net = net.to(device)
if device == 'cuda':
    net = torch.nn.DataParallel(net)
    cudnn.benchmark = True
```

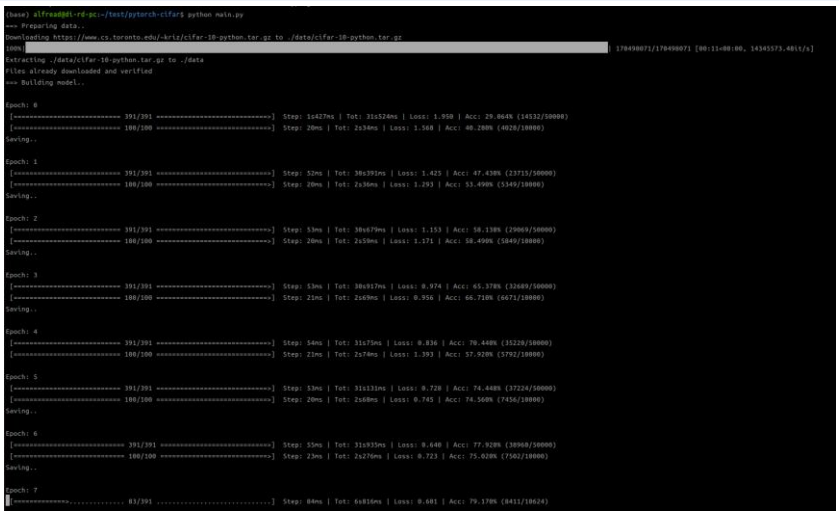
net = net.to(device) -> # net = net.to(device) - '#' Add to disable the model

```
classes = ('plane', 'car', 'bird', 'cat', 'deer',
           'dog', 'frog', 'horse', 'ship', 'truck')

# Model
print('==> Building model..')
# net = VGG('VGG19')
net = ResNet18()
# net = PreActResNet18()
# net = GoogLeNet()
# net = DenseNet121()
# net = ResNeXt29_2x64d()
# net = MobileNet()
# net = MobileNetV2()
# net = DPN92()
# net = ShuffleNetG2()
# net = SENet18()
# net = ShuffleNetV2(1)
# net = EfficientNetB0()
# net = RegNetX_200MF()
net = SimpleDLA()
net = net.to(device)
if device == 'cuda':
    net = torch.nn.DataParallel(net)
    cudnn.benchmark = True
```

```
classes = ('plane', 'car', 'bird', 'cat', 'deer',
           'dog', 'frog', 'horse', 'ship', 'truck')

# Model
print('==> Building model..')
# net = VGG('VGG19')
net = ResNet18()
# net = PreActResNet18()
# net = GoogLeNet()
# net = DenseNet121()
# net = ResNeXt29_2x64d()
# net = MobileNet()
# net = MobileNetV2()
# net = DPN92()
# net = ShuffleNetG2()
# net = SENet18()
# net = ShuffleNetV2(1)
# net = EfficientNetB0()
# net = RegNetX_200MF()
# net = SimpleDLA()
net = net.to(device)
if device == 'cuda':
    net = torch.nn.DataParallel(net)
    cudnn.benchmark = True
```

☐ Write the 1st training start code

```
python main.py
```



Confirmation screen of completion of learning

2) Step 2 – Validation of DDsigner API interworking

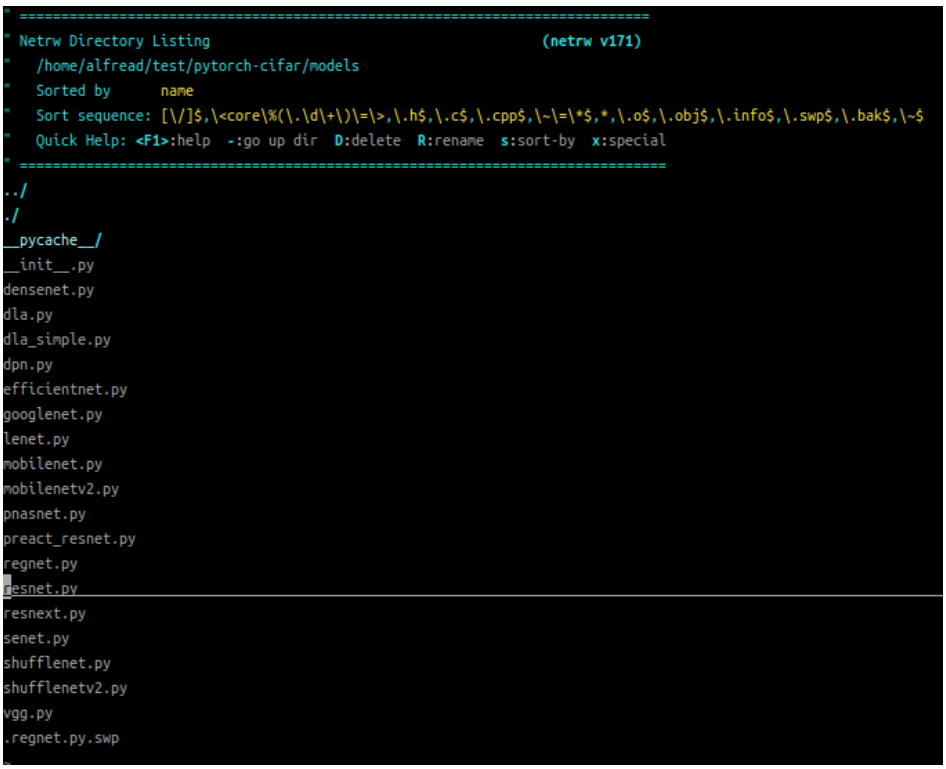
→ Verify API application is successful

→ XWN function is off state

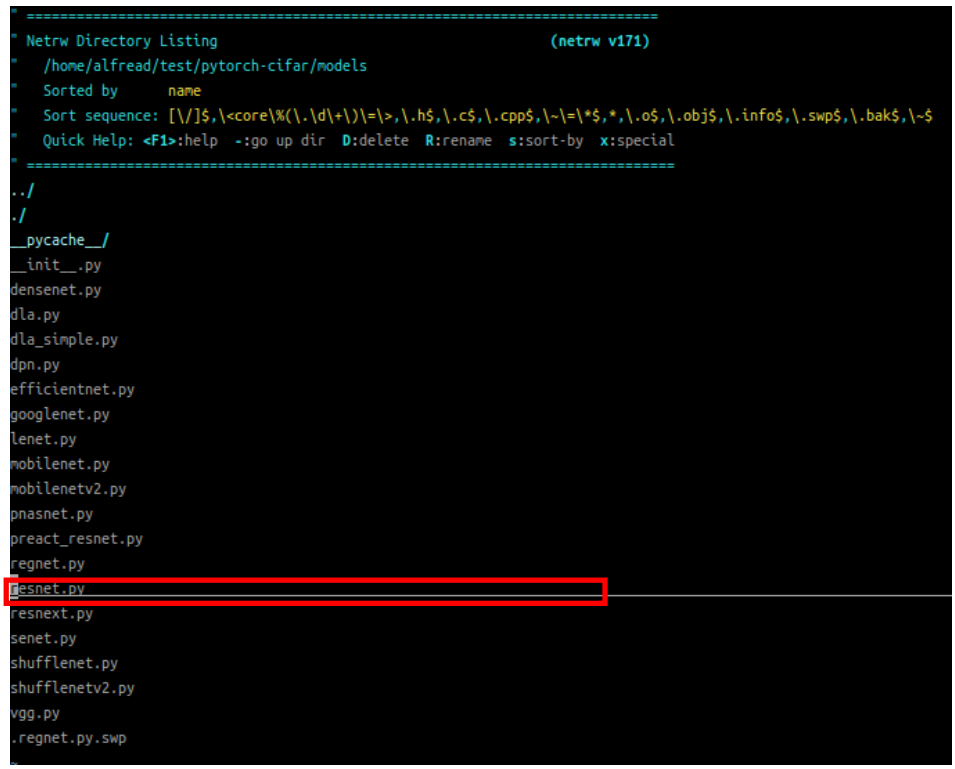
☐ Write vim models code

vim models

vim models execution screen



Select a resnet.py file



resnet.py file execution screen

```
'''
import torch
import torch.nn as nn
import torch.nn.functional as F

from ddesigner_api.pytoch.xwn import torch_nn as cnn

class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, in_planes, planes, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(
            in_planes, planes, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3,
                               stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion*planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, self.expansion*planes,
                          kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(self.expansion*planes)
            )
```

☐ **Write/modify DDesigner API code**

from ddesignr_api.pytorch.xwn import torch_nn as cnn

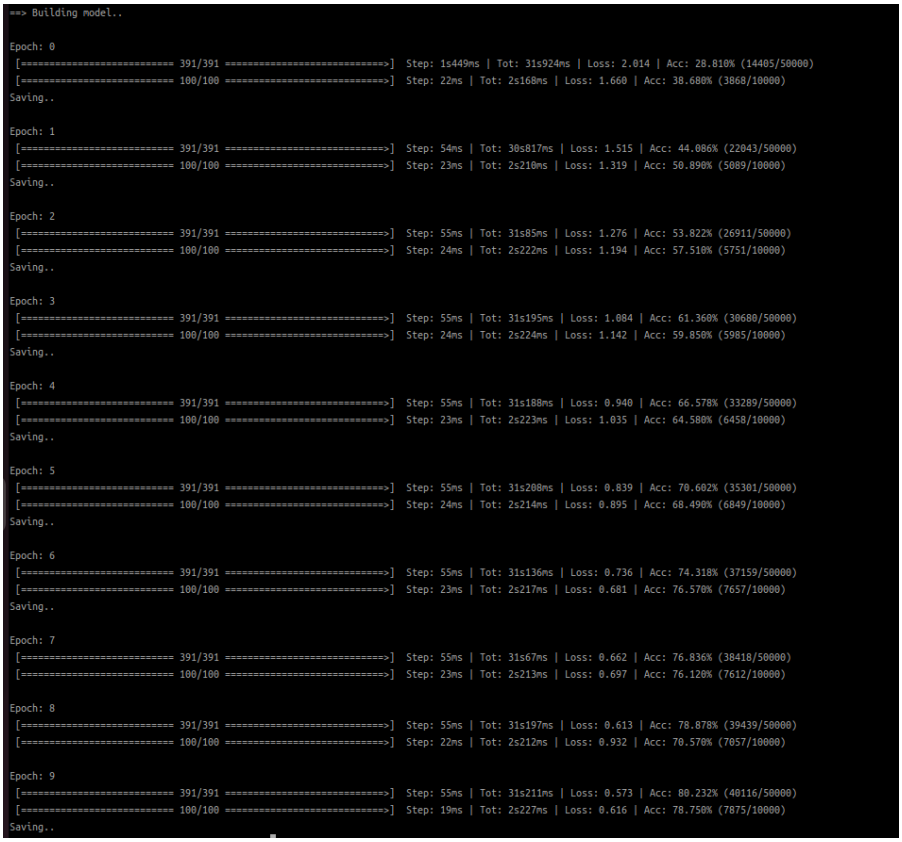
```
'''
import torch
import torch.nn as nn
import torch.nn.functional as F

from ddesigner_api.pytoch.xwn import torch_nn as cnn

class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, in_planes, planes, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(
            in_planes, planes, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3,
                               stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion*planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, self.expansion*planes,
                          kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(self.expansion*planes)
            )
```

※ Please change the existing nn.Conv2d code to → cnn.Conv2d.

☐ Run python main.py file

```
python main.py
```

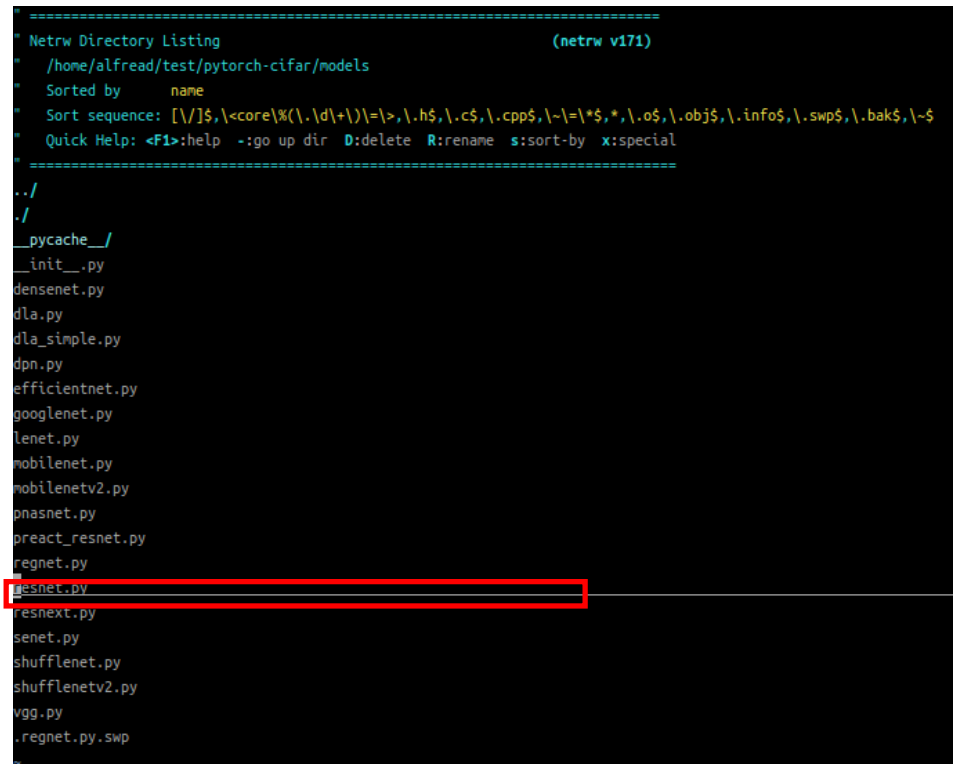
execution screen



3) Step 3 – XWN Applied Learning

→ XWN-based lightweight application

Run vim models



Run the resnet.py file

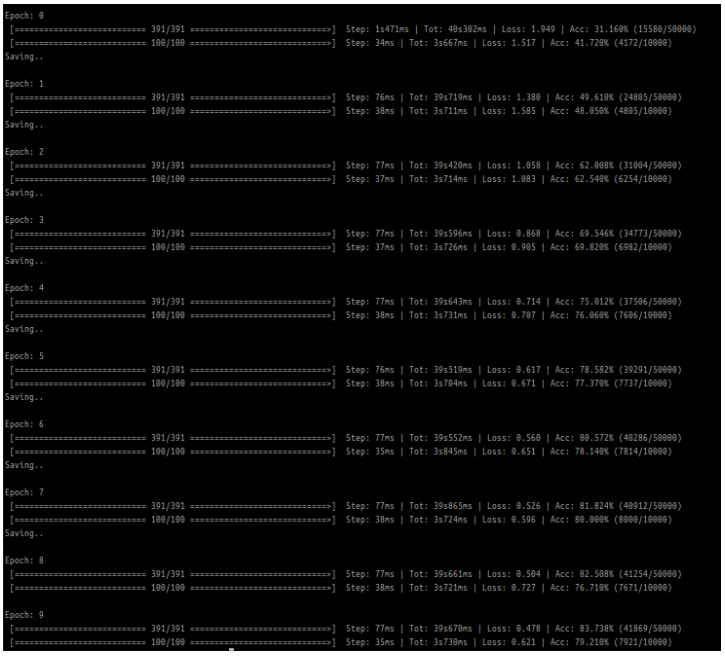
use_transform=True, max_scale=4.0, use_pruning=True, prun_weight=0.50, code writing

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

from ddesigner_api.pytorch.xwn import torch_nn as cnn

class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, in_planes, planes, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = cnn.Conv2d(
            in_planes, planes, kernel_size=3, stride=stride, padding=1, bias=False,
            use_transform=True, max_scale=4.0, use_pruning=True, prun_weight=0.50,
            )
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = cnn.Conv2d(planes, planes, kernel_size=3,
                                stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion*planes:
            self.shortcut = nn.Sequential(
                cnn.Conv2d(in_planes, self.expansion*planes,
                           kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(self.expansion*planes)
            )
```

Save to :wq, then exit

☐ **Run python main.py**

```
python main.py
```



Comparison of learning outcomes for each step

Step 1 Learning Results

```
Epoch: 9
 [========================== 391/391 ==========================>]  Step: 65ms | Tot: 35s102ms | Loss: 0.517 | Acc: 82.278% (41139/50000)
 [========================== 100/100 ==========================>]  Step: 30ms | Tot: 2s846ms | Loss: 0.688 | Acc: 76.370% (7637/10000)
```

Acc: 82%

Step 2 Learning Results

```
Epoch: 9
 [========================== 391/391 ==========================>]  Step: 57ms | Tot: 31s813ms | Loss: 0.510 | Acc: 82.402% (41201/50000)
 [========================== 100/100 ==========================>]  Step: 19ms | Tot: 2s341ms | Loss: 0.581 | Acc: 80.790% (8079/10000)
```

※ If the learning results of Step 1 and Step 2 are different, it is necessary to reconfirm whether DDesigner API is linked or not

Acc: 82%

Step 3 Learning Results

```
Epoch: 9
 [========================== 391/391 ==========================>]  Step: 77ms | Tot: 39s670ms | Loss: 0.478 | Acc: 83.738% (41869/50000)
 [========================== 100/100 ==========================>]  Step: 35ms | Tot: 3s730ms | Loss: 0.621 | Acc: 79.210% (7921/10000)
```

※ Comparing the two-step learning results, it is necessary to check the XWN optimization learning results (Similar figures to the second-stage study results must be obtained)

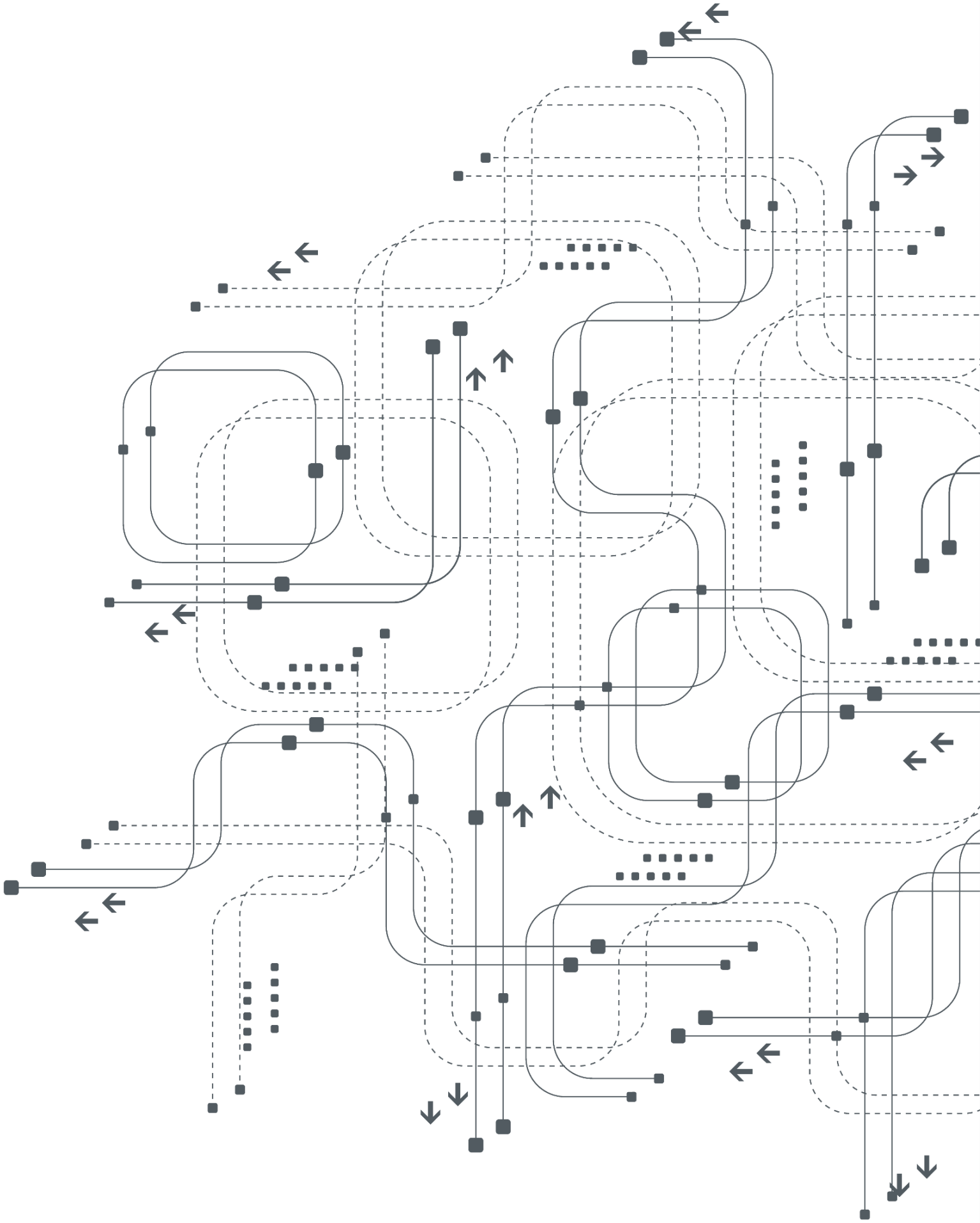
Acc: 83%

4) Accuracy comparison and normal operation verification criteria

Step	Accuracy Expectations	Verification Criteria
Step 1	ex) 0.82%	Initial Validation Criteria
Step 2	ex) 0.82%	normal at the same time
Step 3	ex) 0.83%	Normal if there is no significant difference

0.82% ↔ 0.83% = normal

※ The number of epochs in this guide is set to 9 arbitrary times
As a result, the Acc value may vary from case to case



Customer Support

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