

Tachy-RT API Guidebook

Agenda

1. Overview

- 1-1) What is the Tachy-RT API?
- 1-2) Key Features and Advantages



2. Requirements

- 2-1) System Requirements
- 2-2) Support Platform and Environment

3. Use Example Guide

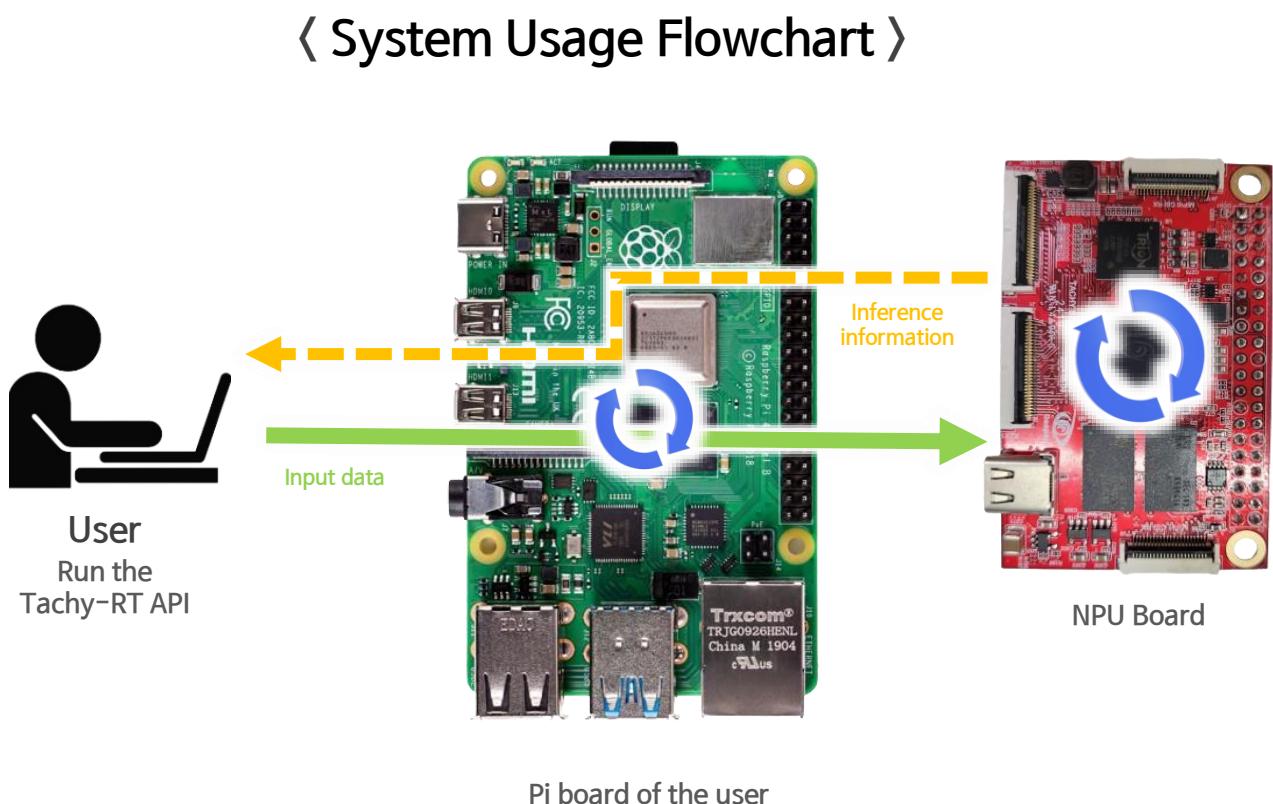
⟨ Preparations Before Running the Tachy-RT API ⟩

- 3-1) Importing required libraries and modules
- 3-2) Preparing for Tachy-RT Configuration
- 3-3) Loading USB Device Firmware
- 3-4) Load Model
- 3-5) Import input image
- 3-6) Inference execution
- 3-7) Perform post-processing

1. Overview

1-1) What is the Tachy-RT API?

The Tachy-RT API is a runtime tool designed to provide CLI-based control of model inference, sensor control, debugging, etc. in the Tachy Device. It operates in a Python environment and can be linked to external systems through various interfaces such as TCP/SPI/Local.



1-2) Key Features and Advantages

- Support for a variety of interfaces
- Real-time inference of trained AI models
- Fast responsiveness and efficient resource use in embedded environments

2. Requirements

2-1) System Requirements

- CPU: Intel i5 and up or AMD rating
- RAM: 8GB or more
- GPU (Optional): NVIDIA CUDA 11.x or later
- OS: Ubuntu 20.04 / Windows 10 or later

2-2) Support Platform and Environment

- Linux (Ubuntu 18.04/20.04)
- Windows 10/11 (64bit)
- Python 3.7 and later, C++ support

practice question) Hello Image Detection

This example uses Tachy-RT to set up the device → load the model → input the image → inference and postprocessing → This is a hands-on guide covering the entire AI-based Object Detection inference process, from visualization of detection results.

⟨Preparations Before Running the Tachy RT API⟩ ⟨ Board preparation ⟩

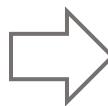
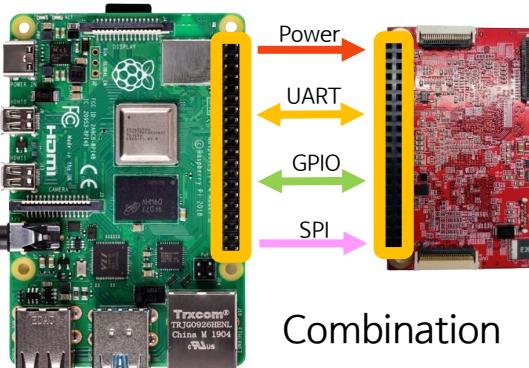


NPU Board

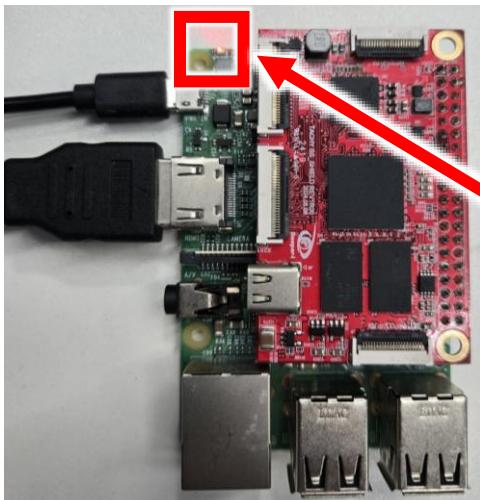


Pi Board

⟨ NPU board, PI combined ⟩



⟨ Ethernet wire connection ⟩



Cheak
Red LED Light

→ CONNECTION ON

※ This is an example photo for your understanding.

3-2) Importing required libraries and modules

```
import sys
sys.path.append('..../post')
sys.path.append('..../utils')
import cv2
import argparse
import numpy as np
from yolov4 import Decoder
from functions import *
import tachy_rt.core.functions as rt_core
```

3-3) Preparing for Tachy-RT Configuration

```
config = @@
{
    "INTERFACE": {
        # # On board
        # "mode": "local",

        # # On host (connected with ethernet)
        # "mode": "ethernet",
        # "ip": "192.168.3.121",

        # On host (connected with usb)
        "mode": "spi:ftdi"
    },
    "GLOBAL": {
        "bs_ver": 2,
        "std": 255.0,
        "mean": 0.0,
        "input_dump": True
    },
    "NETWORK": {
        "network_path": "./model/object_detection_yolov40-20220918-
288x160-bs2.tachyrt"
    }
}
```

3-4) Loading USB Device Firmware

```
if "ftdi" in config["INTERFACE"]["mode"]:  
    ret = rt_core.boot(path='../../firmware', spi_type='ftdi')  
    if ret:  
        print("Success to boot. Check the status via uart or  
other api")  
    else:  
        print("Failed to boot")  
        exit(-1)
```

3-5) Load Model

```
engine = rt_core.make_engine(config)  
if engine is None:  
    print("make engine fail")  
    exit()
```

3-6) Import input image

```
# The Object Detection model expects images  
in RGB format.  
image = cv2.imread("./image/image_3cls.jpg")  
org_h, org_w, _ = image.shape  
  
# Input size of Object Detection model is  
# 288x160(h x w)  
image_input = cv2.resize(image, (288,  
160)) [..., ::-1]  
  
# Shape of input must be (B, H, W, D)  
input_image = np.expand_dims(image_input, 0)  
display_img_plt(image)
```

3-7) Inference execution

```
engine.process(input_image)
ret = engine.get_result()
```

3-8) Perform post-processing

1. Prepare post processing configuration
2. Create post processing instance
3. Do post processing

```
# load post configuration
post_config =
read_json('./config/post_3cls.json')

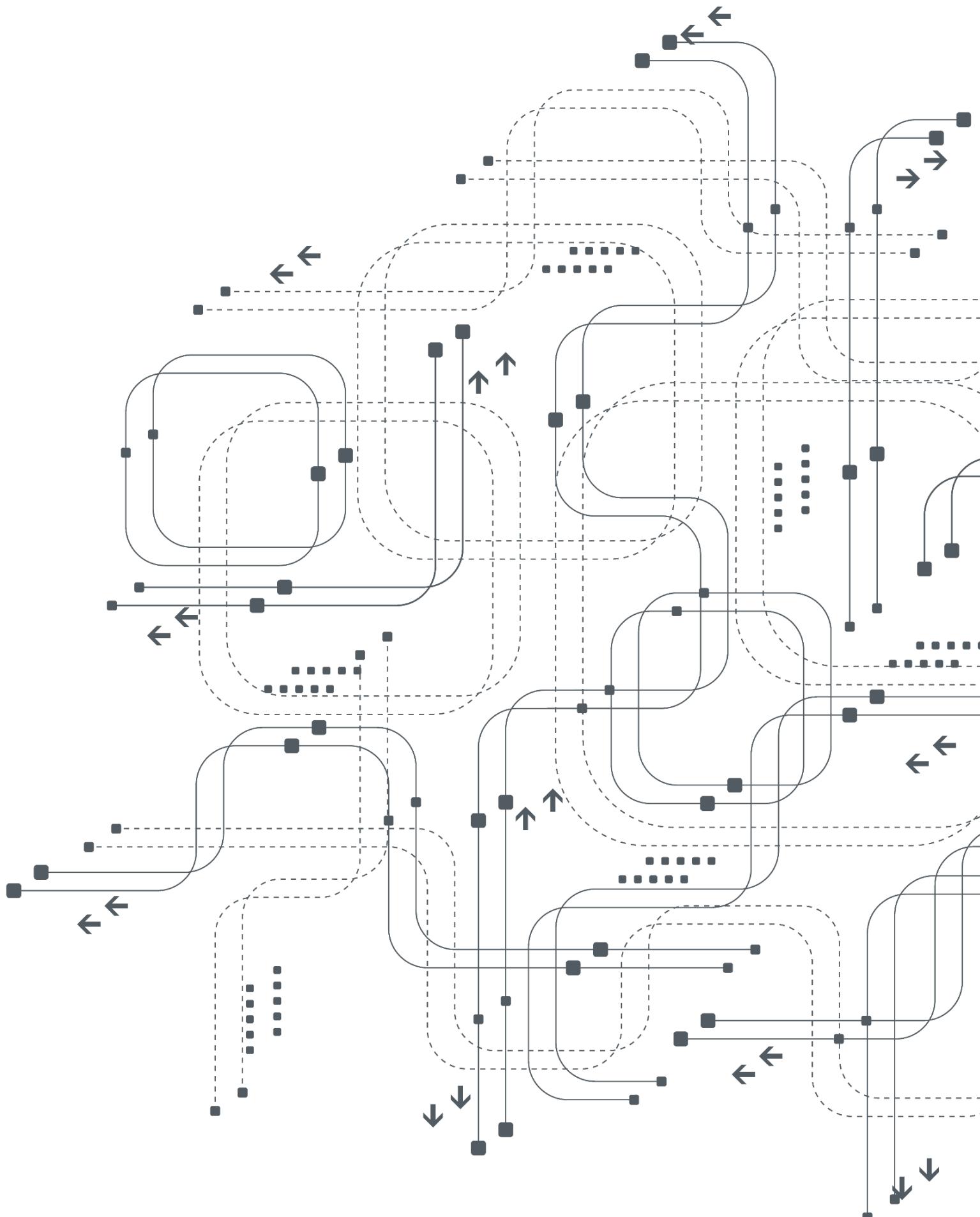
# Init post decoder
output_depth =
int(np.mean(np.asarray(post_config["INF_SHAPES_OUTPUT"]))[..., -1]))
post_processing = Decoder(post_config)

# Do post process for annotation
anno =
post_processing.main(ret['buf'].reshape(-1,
output_depth), np.array([[0, 0, org_w-1,
org_h-1]], dtype=np.float32))
```

Draw Annotations

1. Prepare labels for detection result
2. Draw detection

```
# Draw box and show image
labels = read_json('./label/label_3cls.json')
img = draw_detection(image, anno, labels)
display_img_plt(img)
```



Customer Support

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