ModelBox-博时特EC02上手指南

博时特EC02是一款体积小巧、功能强大的人工智能边缘计算盒，结合ModelBox开发框架和HiLens管理平台，开发者可以方便快速的进行AI应用的开发部署。

配置网络

使用博时特EC02进行ModelBox AI应用开发有两种方式，一是盒子连接显示器和键盘鼠标，安装Ubuntu桌面，直接在盒子上进行开发；二是使用远程连接工具（如VS Code中的`Remote-SSH`）从PC端登录盒子进行开发。这里我们推荐第二种方式，因为PC端可以使用功能更丰富、界面更友好的IDE。

PC连接盒子需要知道盒子的ip，但是盒子默认没有固定ip，此时也有两种方式，一是将PC和盒子连接到同一局域网中，使用盒子的动态ip进行登录；二是为盒子配置一个固定ip，把PC也配置到同一网段，二者用网线连接进行登录。

相同局域网方式1——使用无线路由器

1) PC连接（有线或无线均可）到无线路由器，盒子用网线也连接到该无线路由器；

2) PC登录到该无线路由器主页，通过终端列表查看盒子的ip（下图以华为无线路由器为例，红框中即为盒子ip）；

3) PC使用该ip即可SSH登录。

相同局域网方式2——连接相同WIFI

1) 盒子连接显示器和键盘鼠标，登录Ubuntu系统（账号密码均为`rock`，建议登陆后马上修改为自定义密码，以免密码泄露）；

3) 使用`nmcli`命令连接WIFI（安装WIFI模组后才能连接WIFI）；

查询可以连接到的WIFI：

```
nmcli device wifi list
```

连接wifi：

```
nmcli device wifi connect [WIFI名称] password [WIFI密码]
```

3) 使用`ifconfig`命令查看盒子ip，将PC也连接到同一个WIFI，即可SSH登录。

固定IP方式

1) 盒子连接显示器和键盘鼠标，通过网线连接网络，登录Ubuntu系统；

2) 安装`netplan`（需要连接公网，另外如果已经有的话不需要安装，判断方法：`sudo su`切换到`root`，查看是否有`/etc/netplan`目录）：

```
sudo apt-get install netplan.io
```

3) `ifconfig`命令查看有线网卡对应的网口名（一般是`eth0`）：

4) vim编辑`/etc/netplan`目录下的`config.yaml`文件（如果没有就新建`config.yaml`），配置网口及ip（可参考`netplan`官网示例，保存并退出（下面示例中盒子ip配置为192.168.2.55）：

```
network:
  version: 2
  renderer: networkd
  ethernets:
    eth0:
      dhcp4: false
      addresses: [192.168.2.55/23]
      gateway4: 192.168.2.1
      optional: true
      nameservers:
            addresses: [8.8.8.8]
```

5) 配置完成后root执行`netplan apply`使配置生效，重新用`ifconfig`命令查看网口是否有固定ip（即下图eth0下inet处的值），之后可以通过更改PC的ip为同一网段并通过网线连接电脑和盒子来登录（可参考HiLens Kit和PC的连接方式）。

```
netplan apply
ifconfig
```

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet [redacted]  netmask 255.255.254.0  broadcast [redacted]
        inet6 [redacted]  prefixlen 64  scopeid 0x20<link>
        ether [redacted]  txqueuelen 1000  (Ethernet)
        RX packets 5403466  bytes 2413014700 (2.4 GB)
        RX errors 0  dropped 727399  overruns 0  frame 0
        TX packets 2743417  bytes 778775312 (778.7 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 24

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1  (Local Loopback)
        RX packets 2145189  bytes 186397901 (186.3 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2145189  bytes 186397901 (186.3 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

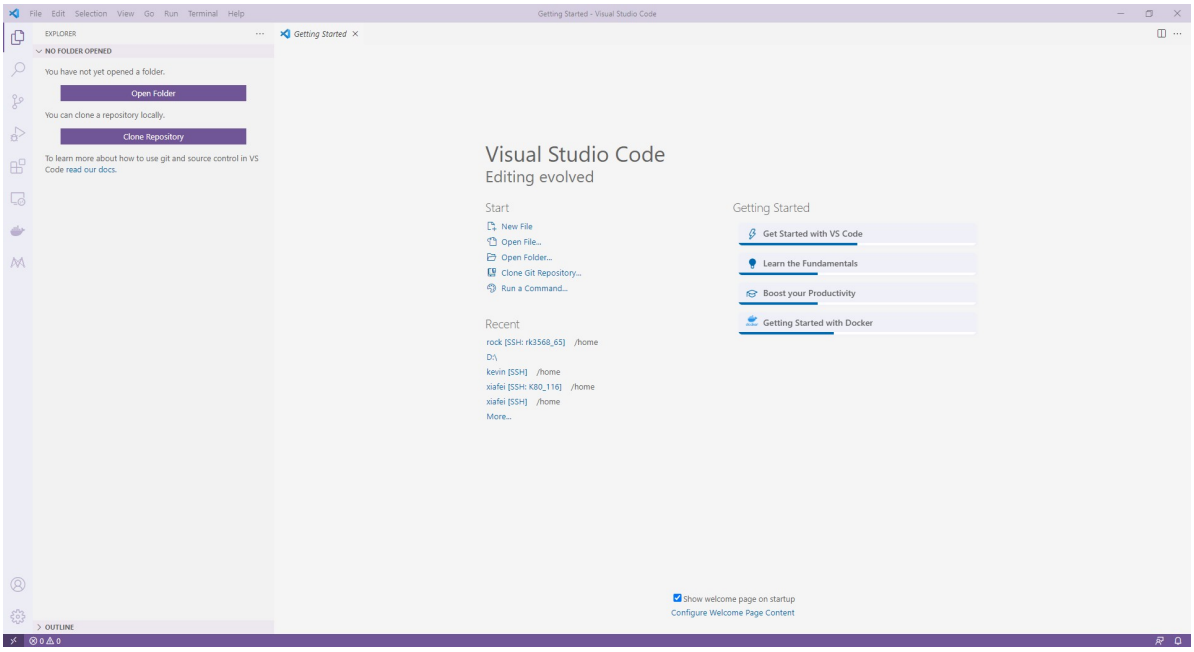
应用开发

1. 远程连接盒子

我们推荐在PC端使用VS Code远程连接盒子来对设备操作。

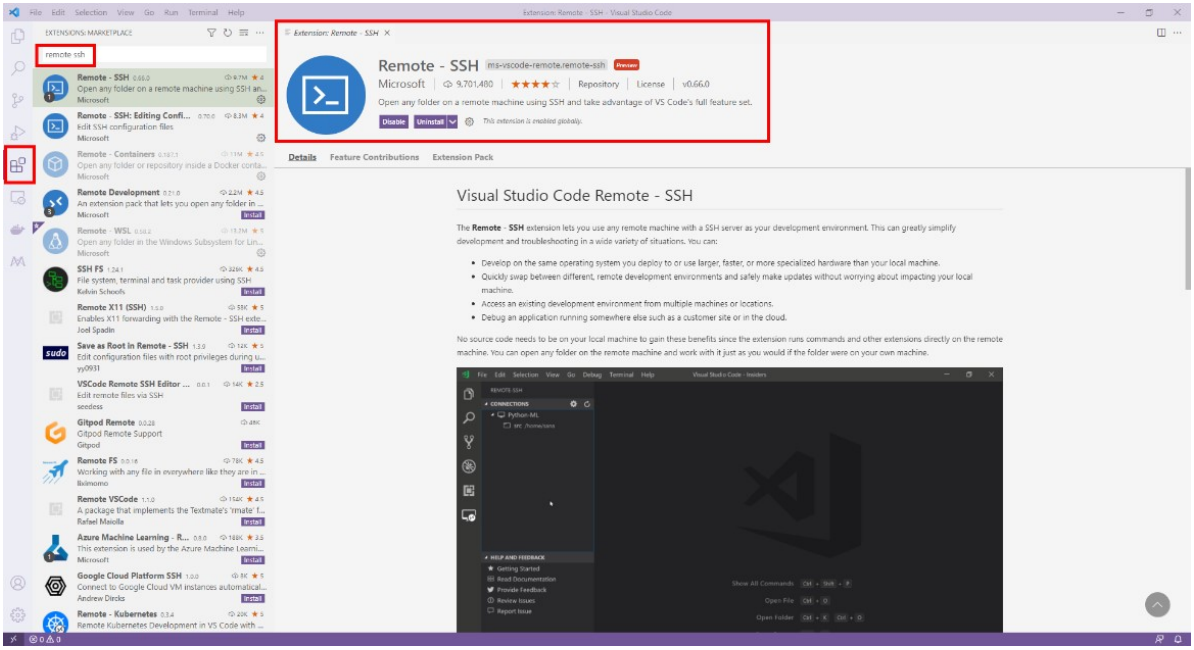
1）下载VS Code

VS Code官网下载安装对应版本的编辑器。



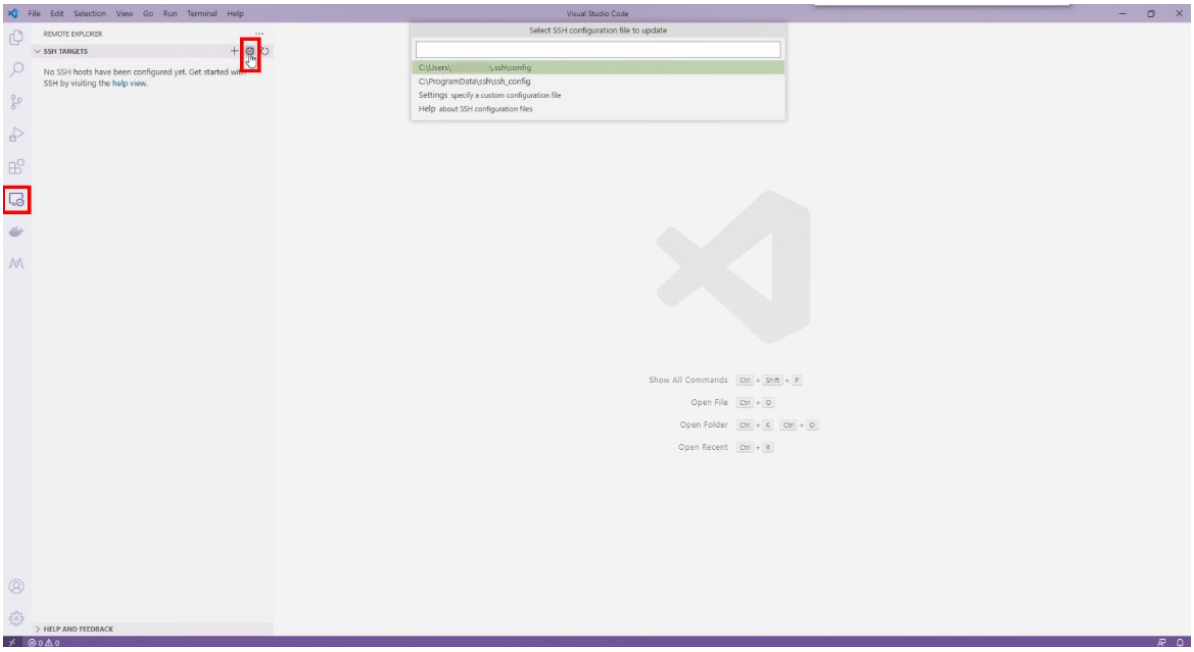
2）安装`Remote-SSH`

在VS Code中安装`Remote-SSH`插件，安装完成后侧边栏将出现远程连接`Remote Explorer`的图标。



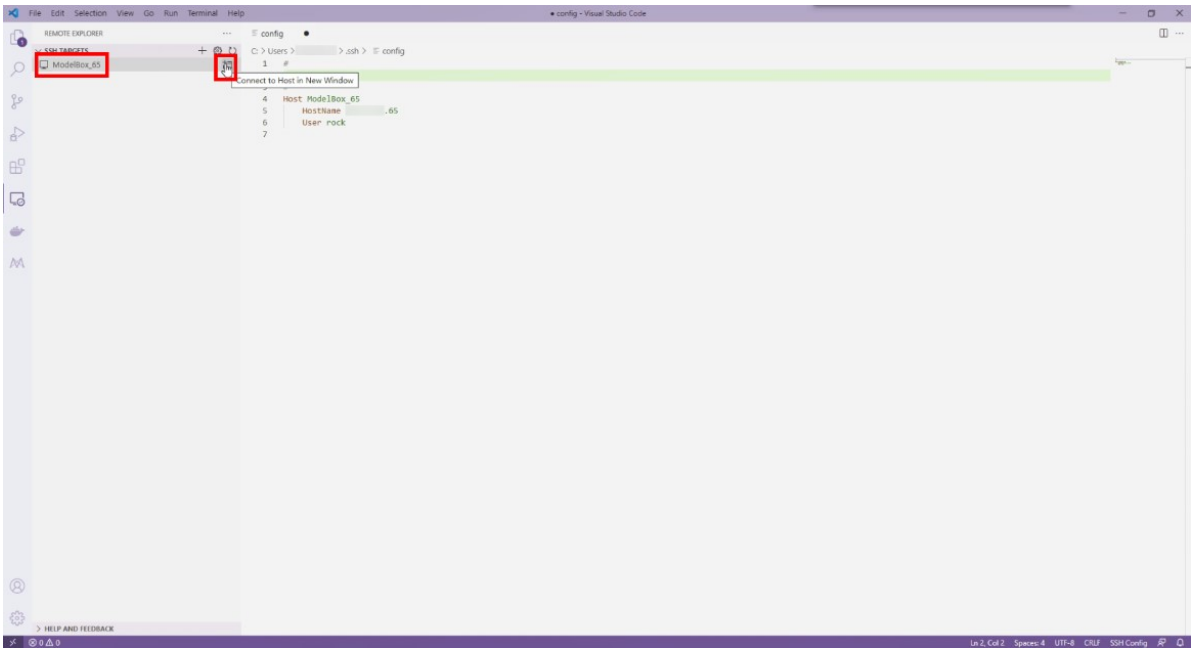
3）进入SSH配置文件

点击VS Code侧边栏的`Remote Explorer`图标，在弹出的`SSH TARGETS`界面中点击齿轮图标，选择一个SSH配置文件路径。



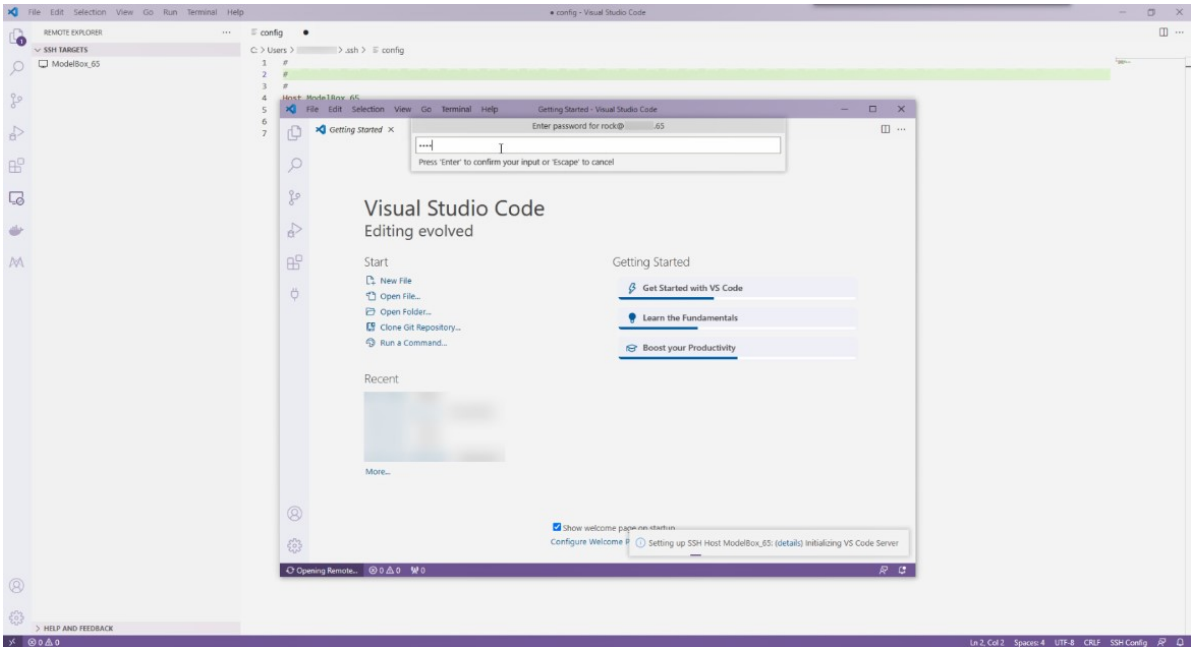
4）配置SSH连接

配置SSH连接，参考下图，其中Host是自己设置的盒子名称，HostName是盒子ip地址，User是SSH连接用户名。保存配置文件，`SSH TARGETS`将出现该Host图标，点击图标后面的连接图标。



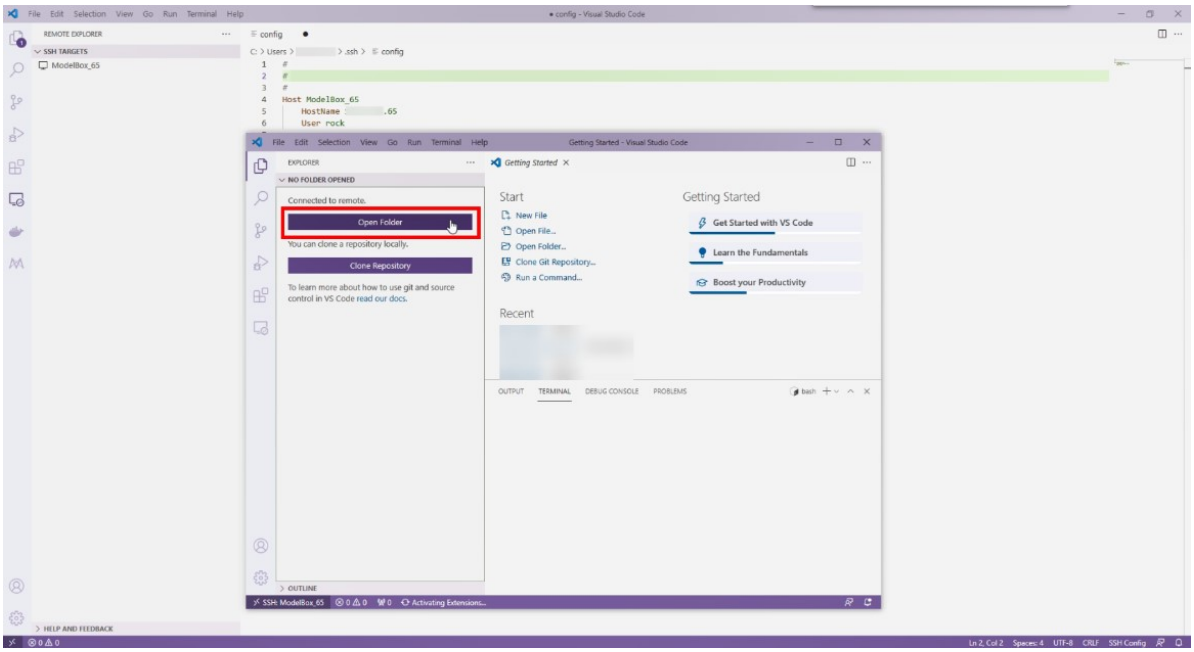
5）输入密码

在弹出的连接界面中输入密码。



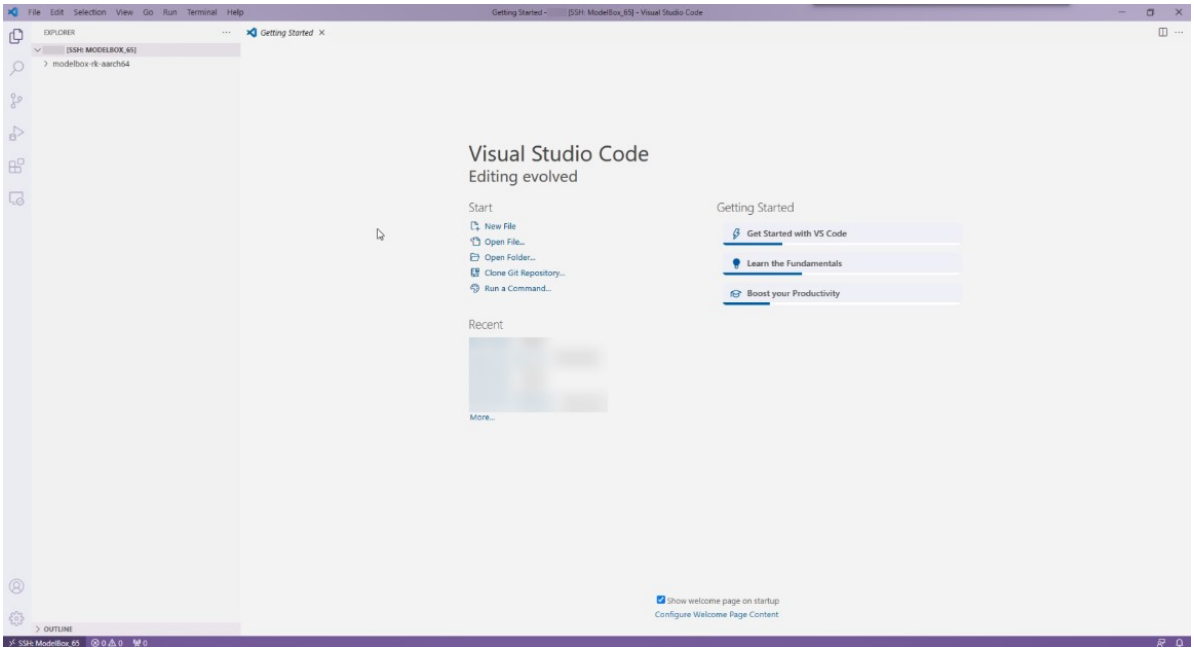
6）打开文件夹

点击`Open Folder`。



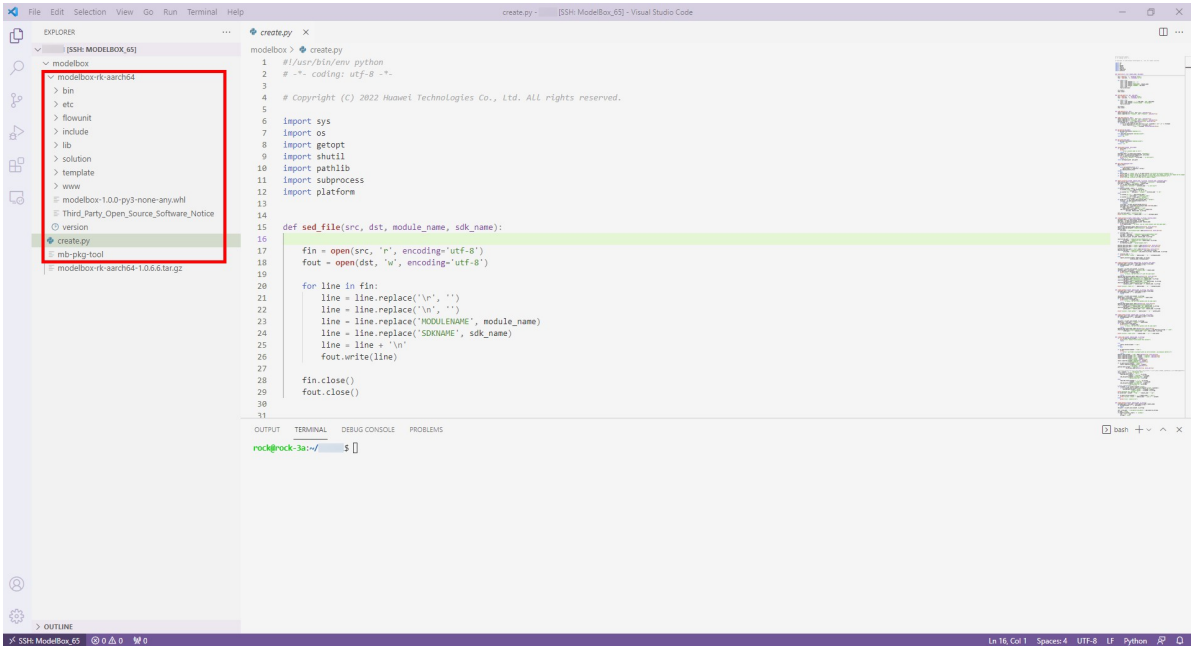
7）进入开发目录

选择一个路径作为ModelBox SDK和应用存放的目录，再次输入密码，即可连接到盒子的该路径下进行应用开发。



2. 下载ModelBox sdk

联系我们获取RK系列对应的ModelBox sdk。
将sdk拷贝到盒子上（拖动文件到VS Code界面中即可），解压可得到sdk如下：



其中`modelbox-rk-aarch64`文件夹即为sdk，包含modelbox运行环境、内置的功能单元等，`create.py`为创建modelbox工程、创建功能单元、编译运行等的辅助工具。进入sdk目录，执行`create.py`可看到辅助工具的用法介绍（需使用python3.8版本进行编码开发）：

```
rock@rock-3a:~/███/modelbox$ ./create.py

Usage: Create ModelBox project and flowunit

NOTE : you must firstly use bellow cmd to create a project
    create.py -t project -n your_proj_name {option: -s name, create this project
from a solution}
AND  : use bellow cmd to create  [c++|python|infer]  flowunit in this project
    create.py -t c++ -n your_flowunit_name -p your_proj_name
AND  : call build_project.sh to build your project, call run/main.sh[bat] to run
FINAL: create.py -t rpm -n your_proj_name  to package your project if upload to
hilens

NOTE: create.py -t editor -n your_proj_name to start web ui editor to edit your
graph

-h or --help: show help
-t or --template [c++|python|infer|project|rpm|editor]  create a template or
package to rpm
-n or --name [your template name]
-p or --project  [your project name when create c++|python|infer]
-s or --solution [the solution name when create project] create a project from
solution
-v or --version: show sdk version

rock@rock-3a:~/███/modelbox$
```

3. 开发Hello World应用

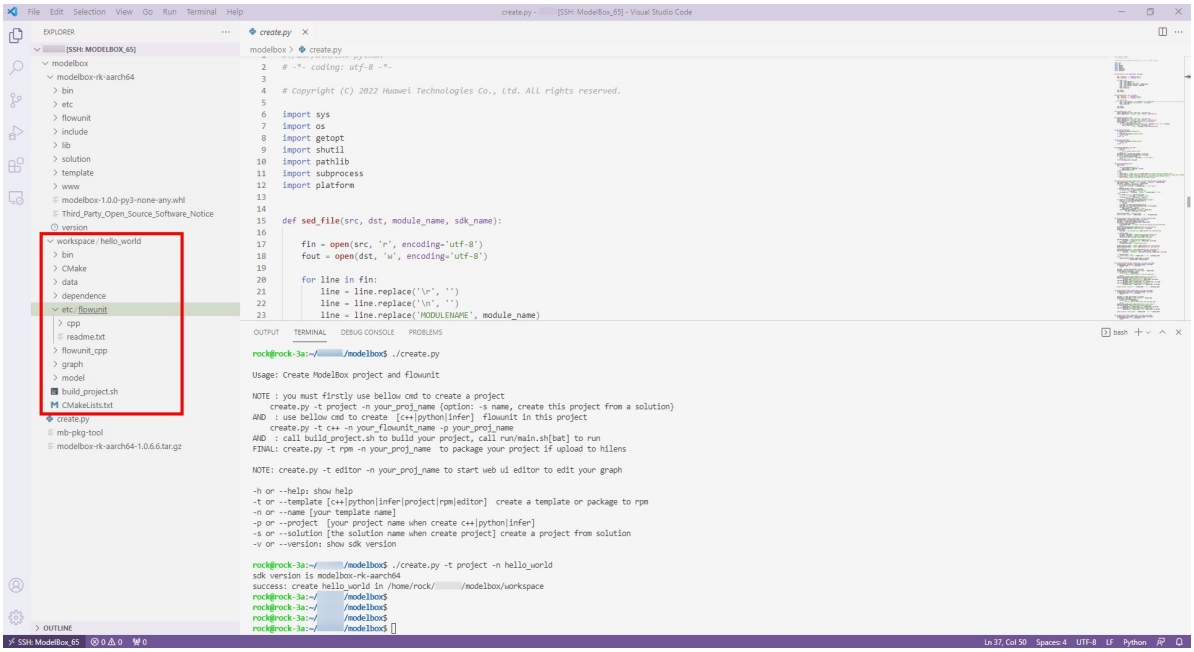
接下来我们用Python开发一个最简单的ModelBox应用：打开一个视频文件，在画面左上方写上“Hello World”，再输出到另一个视频文件中。

1）创建工程

使用`create.py`创建`hello_world`工程

```
rock@rock-3a:~/███/modelbox$ ./create.py -t project -n hello_world
sdk version is modelbox-rk-aarch64
success: create hello_world in /home/rock/███/modelbox/workspace
```

可以看到，第一次创建工程时，在modelbox sdk目录下，自动生成了`workspace`文件夹，此文件夹将作为modelbox应用的默认目录。`workspace`目录下创建了`hello_world`工程：

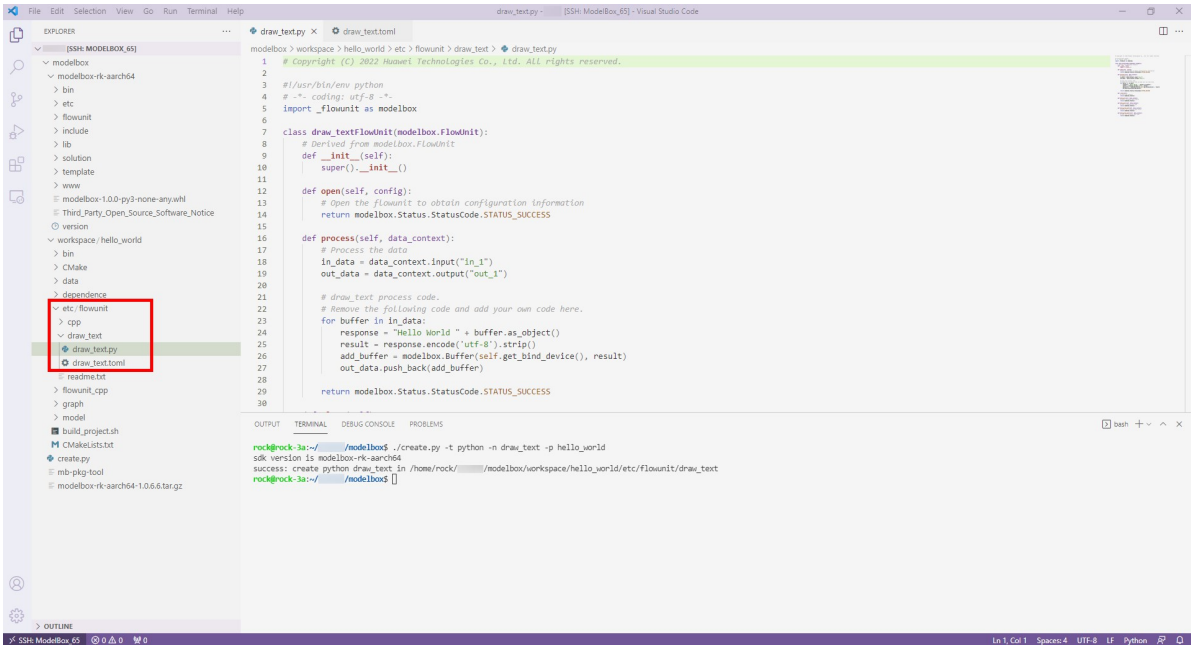


2）创建功能单元

为`hello_world`工程创建python的`draw_text`功能单元：

```
rock@rock-3a:~/███/modelbox$ ./create.py -t python -n draw_text -p hello_world
sdk version is modelbox-rk-aarch64
success: create python draw_text in /home/rock/
███/modelbox/workspace/hello_world/etc/flowunit/draw_text
```

可以看到，在`hello_world`工程的`etc/flowunit`目录（此目录将作为Python功能单元的默认存放目录）下，生成了`draw_text`文件夹，其中自动生成了Python功能单元所需的py文件和toml配置文件：



3）修改功能单元

`draw_text.toml`中配置该功能单元的名称、类别、输入输出端口等信息，当前不用修改；`draw_text.py`中描述了该功能单元的处理逻辑，这里我们增加OpenCV与NumPy包的引用（需要事先用pip安装OpenCV与NumPy的Python库），修改其中的`process`函数如下：

```
import cv2
import numpy as np
import _flowunit as modelbox

…

    def process(self, data_context):
        # Process the data
        in_data = data_context.input("in_1")
        out_data = data_context.output("out_1")

        # draw_text process code.
        # Remove the following code and add your own code here.
        for buffer_img in in_data:
            width = buffer_img.get('width')
            height = buffer_img.get('height')
            channel = buffer_img.get('channel')

            img_data = np.array(buffer_img.as_object(), copy=False)
            img_data = img_data.reshape((height, width, channel))

            cv2.putText(img_data, 'Hello World', (50, 50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)

            out_buffer = self.create_buffer(img_data)
            out_buffer.copy_meta(buffer_img)
            out_data.push_back(out_buffer)

        return modelbox.Status.StatusCode.STATUS_SUCCESS
```

4）修改流程图

`hello_world`工程`graph`目录下默认生成了一个`hello_world.toml`流程图，修改其中的流程定义graphconf如下：

```
graphconf = """digraph hello_world {
    node [shape=Mrecord];
    queue_size = 4
    batch_size = 1
    video_input[type=flowunit, flowunit=video_input, device=cpu, deviceid=0,
source_url="xxx/xxx.mp4"]
    video_demuxer[type=flowunit, flowunit=video_demuxer, device=cpu, deviceid=0]
    video_decoder[type=flowunit, flowunit=video_decoder, device=rknpu,
deviceid=0, pix_fmt=bgr]
    draw_text[type=flowunit, flowunit=draw_text, device=cpu, deviceid=0]
    video_encoder[type=flowunit, flowunit=video_encoder, device=cpu, deviceid=0,
default_dest_url="xxx/xxx.mp4", format=mp4]

    video_input:out_video_url -> video_demuxer:in_video_url

    video_demuxer:out_video_packet -> video_decoder:in_video_packet

    video_decoder:out_video_frame -> draw_text:in_1
    draw_text:out_1 -> video_encoder:in_video_frame
}"""
```

我们需要准备一个mp4文件拷贝到`hello_world`工程中（可以使用sdk目录下的`solution/common/car_test_video.mp4`），然后将`video_input`单元中的`source_url`属性内容修改为实际的mp4文件路径，将`video_encoder`单元`default_dest_url`属性内容修改为保存应用结果的mp4文件路径。

5）构建工程

在`hello_world`工程路径下执行`build_project.sh`进行工程构建：

```
rock@rock-3a:~/███/modelbox/workspace/hello_world$ ./build_project.sh

build success: you can run main.sh in ./run folder

rock@rock-3a:~/███/modelbox/workspace/hello_world $
```

如果执行过程中`dos2unix`报错，请使用`sudo apt-get install dos2unix`安装：

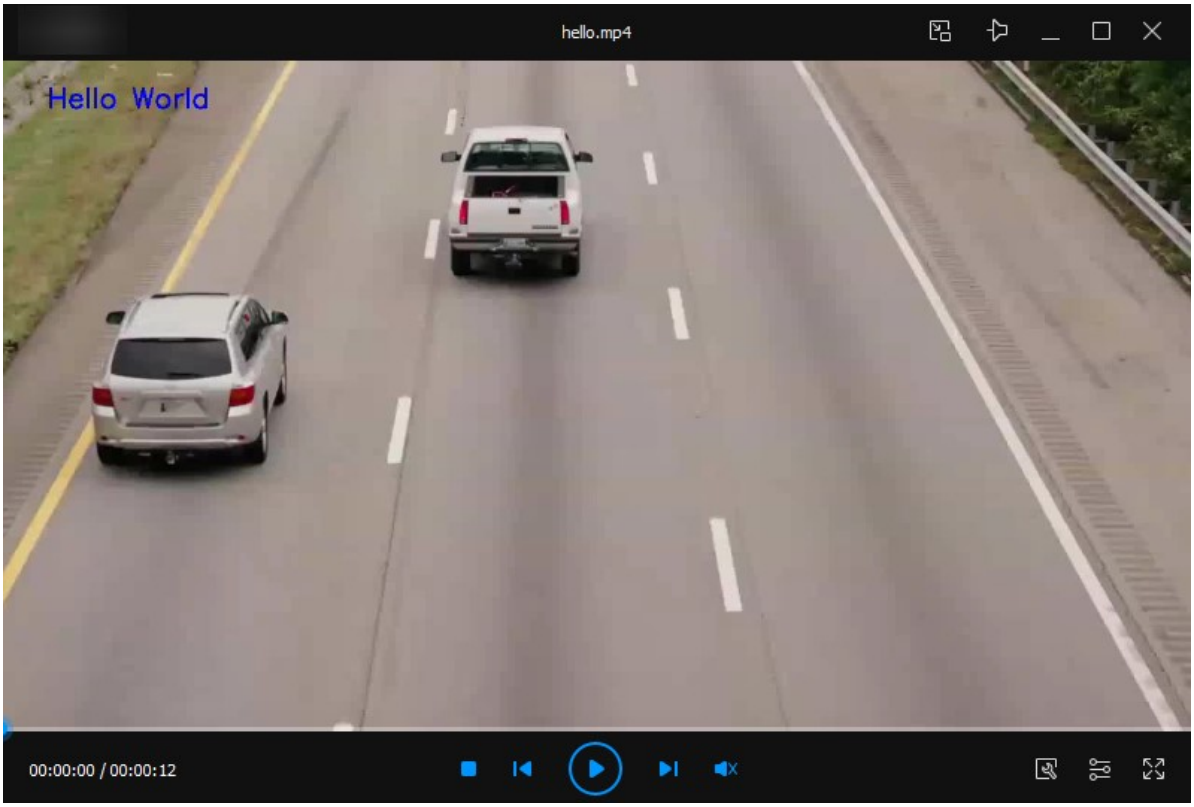
```
./build_project.sh: line 26: dos2unix: command not found
```

6）运行应用

构建操作将在`hello_world`工程路径下生成`run`文件夹，其中包含应用执行脚本和应用流程图，执行`run/main.sh`运行应用（如果执行过程中报错，可尝试切换到root用户再运行）：

```
rock@rock-3a:~/███/modelbox/workspace/hello_world$ ./run/main.sh
[2022-03-18 03:02:16,552][ INFO][          flow.cc:97  ] run flow /home/rock/
███/modelbox/workspace/hello_world/run/etc/graph/hello_world.toml
[2022-03-18 03:02:16,629][ INFO][        driver.cc:549 ] Gather scan info
success, drivers count 47
[2022-03-18 03:02:16,635][ INFO][        driver.cc:357 ] load success drivers:
count 47, show detail in debug level
…
[2022-03-18 03:02:21,577][ INFO][ ffmpeg_writer.cc:67  ] Open url /home/rock/
███/modelbox/workspace/hello_world/xxx.mp4, format mp4 success
…
[2022-03-18 03:02:33,849][ INFO][video_decoder_flowunit.cc:83  ] Video decoder
finish
[2022-03-18 03:02:33,912][ INFO][          node.cc:1142] video_decoder data_ctx
finished se id:e21455b0-7957-49ca-8128-8c549f4a184b
[2022-03-18 03:02:34,189][ INFO][          node.cc:1142] video_encoder data_ctx
finished se id:e21455b0-7957-49ca-8128-8c549f4a184b
[2022-03-18 03:02:34,196][ INFO][session_context.cc:41  ] session context finish
se id:e21455b0-7957-49ca-8128-8c549f4a184b
```

运行结束后将在`video_encoder`单元的`default_dest_url`目录下生成视频文件，可以下载到PC端查看。



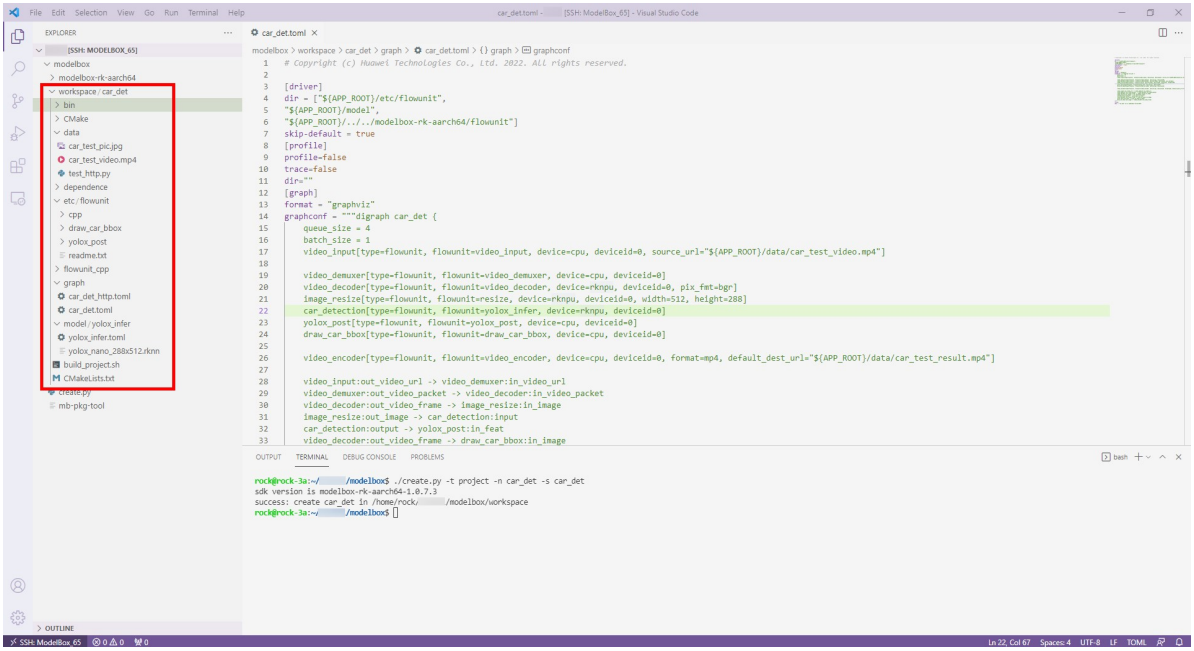
4. 开发第一个AI应用

接下来我们开发一个车辆检测应用：打开一个视频文件，使用检测模型检测出画面中车辆并画框，再输出到另一个视频文件中。本应用作为模板案例已内置在sdk中，不需要另外下载。

1）创建工程

使用车辆检测模板创建`car_det`工程（注意与创建`hello_world`工程的区别）：

```
rock@rock-3a:~/███/modelbox$ ./create.py -t project -n car_det -s car_det
sdk version is modelbox-rk-aarch64
success: create car_det in /home/rock/███/modelbox/workspace
```



2）查看推理功能单元

这个应用使用到了模型推理，需要用到推理功能单元，可以看到，在`car_det`工程目录的`model`文件夹下，存在`yolox_infer`推理功能单元文件夹，里面有yolox模型文件（`yolox_nano_288x512.rknn`）和模型配置文件（`yolox_infer.toml`），模型配置文件内容如下：

```
# Copyright (C) 2022 Huawei Technologies Co., Ltd. All rights reserved.

[base]
name = "yolox_infer"
device = "rknpu"
version = "1.0.0"
description = "car detection"
entry = "./yolox_nano_288x512.rknn"  # model file path, use relative path
type = "inference"
virtual_type = "rknpu" # inference engine type: rockchip now support rknpu,
rknpu2(if exist)
group_type = "Inference"  # flowunit group attribution, do not change
is_input_contiguous = "false"  # input data attribution, do not change

# input port description, suporrt multiple input ports
[input]
[input.input1]
name = "input"
type = "uint8"
device = "rknpu"

# output port description, suporrt multiple output ports
[output]
[output.output1]
name = "output"
type = "float"
```

可以看到该模型有1个输入节点，1个输出节点。需要注意其中的`virtual_type`配置与npu类别有关，RK1808需配置为`rknpu`；输入节点的`device`配置建议设为与该推理功能单元的上一个功能单元相同。**ModelBox内置了rknn推理引擎和推理逻辑，开发者只需要准备好模型、编辑好配置文件，即可使用该模型进行推理，无需编写推理代码。**

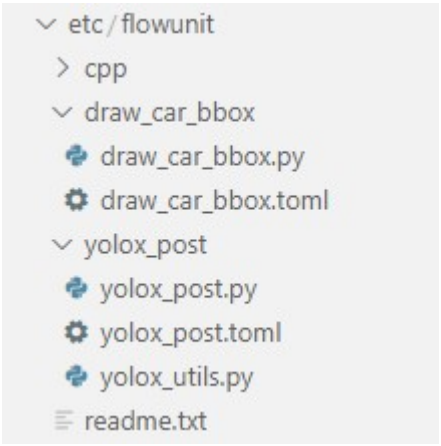
如果想要创建另外的推理功能单元，可以使用如下命令，推理功能单元默认创建在工程目录的`model`文件夹下：

```
rock@rock-3a:~/███/modelbox$ ./create.py -t infer -n my_model -p car_det
sdk version is modelbox-rk-aarch64
success: create infer my_model in /home/rock/
███/modelbox/workspace/car_det/model/my_model
```

另外，本案例使用的车辆检测模型是由PyTorch框架训练得到，我们事先使用rknn-toolkit工具将它转换为RK1808（博时特EC02中的npu型号）支持的模型格式，感兴趣的话可以在RK1808模型转换验证案例中查看模型转换过程。

3）查看其他功能单元

车辆检测模型推理后需要做一些后处理操作得到检测框，再把检测框添加到原始画面中，我们已经准备好了对应的功能单元`yolox_post`和`draw_car_bbox`：



4）查看执行脚本

`car_det`工程`graph`目录下带有两个流程图，执行哪个流程图是由`car_det/bin/main.sh`脚本决定的，打开该脚本看到其内容为：

```
#!/bin/bash
# Copyright (C) 2022 Huawei Technologies Co., Ltd. All rights reserved.

export PATH=${PATH}:${APP_ROOT}/../../modelbox-rk-aarch64/bin
export LD_LIBRARY_PATH=${APP_ROOT}/../../modelbox-rk-
aarch64/lib:${APP_ROOT}/dependence
if [ "$1" == "default" -o "$1" == "" ]; then
  GRAPH_TYPE=""
else
  GRAPH_TYPE="_$1"
fi

modelbox-tool -verbose -log-level INFO flow -run
${APP_ROOT}/run/etc/graph/car_det${GRAPH_TYPE}.toml
```

即默认执行与工程同名的流程图。

5）查看默认流程图

与工程同名的`car_det.toml`流程图中的流程定义graphconf如下：

```
graphconf = """digraph car_det {
    node [shape=Mrecord];
    queue_size = 4
    batch_size = 1
    video_input[type=flowunit, flowunit=video_input, device=cpu, deviceid=0,
source_url="${APP_ROOT}/data/car_test_video.mp4"]

    video_demuxer[type=flowunit, flowunit=video_demuxer, device=cpu, deviceid=0]
    video_decoder[type=flowunit, flowunit=video_decoder, device=rknpu,
deviceid=0, pix_fmt=bgr]
    image_resize[type=flowunit, flowunit=resize, device=rknpu, deviceid=0,
width=512, height=288]
    car_detection[type=flowunit, flowunit=yolox_infer, device=rknpu, deviceid=0]
    yolox_post[type=flowunit, flowunit=yolox_post, device=cpu, deviceid=0]
    draw_car_bbox[type=flowunit, flowunit=draw_car_bbox, device=cpu, deviceid=0]

    video_encoder[type=flowunit, flowunit=video_encoder, device=cpu, deviceid=0,
format=mp4, default_dest_url="${APP_ROOT}/data/car_test_result.mp4"]

    video_input:out_video_url -> video_demuxer:in_video_url
    video_demuxer:out_video_packet -> video_decoder:in_video_packet
    video_decoder:out_video_frame -> image_resize:in_image
    image_resize:out_image -> car_detection:input
    car_detection:output -> yolox_post:in_feat
    video_decoder:out_video_frame -> draw_car_bbox:in_image
    yolox_post:out_data -> draw_car_bbox:in_bbox
    draw_car_bbox:out_image -> video_encoder:in_video_frame
}"""
```

该流程图使用`${APP_ROOT}/data/car_test_video.mp4`文件进行车辆检测，检测结果绘制后保存为`${APP_ROOT}/data/car_test_result.mp4`文件。环境变量`${APP_ROOT}`在编译构建时将自动替换为当前工程的实际路径。

6）运行默认应用

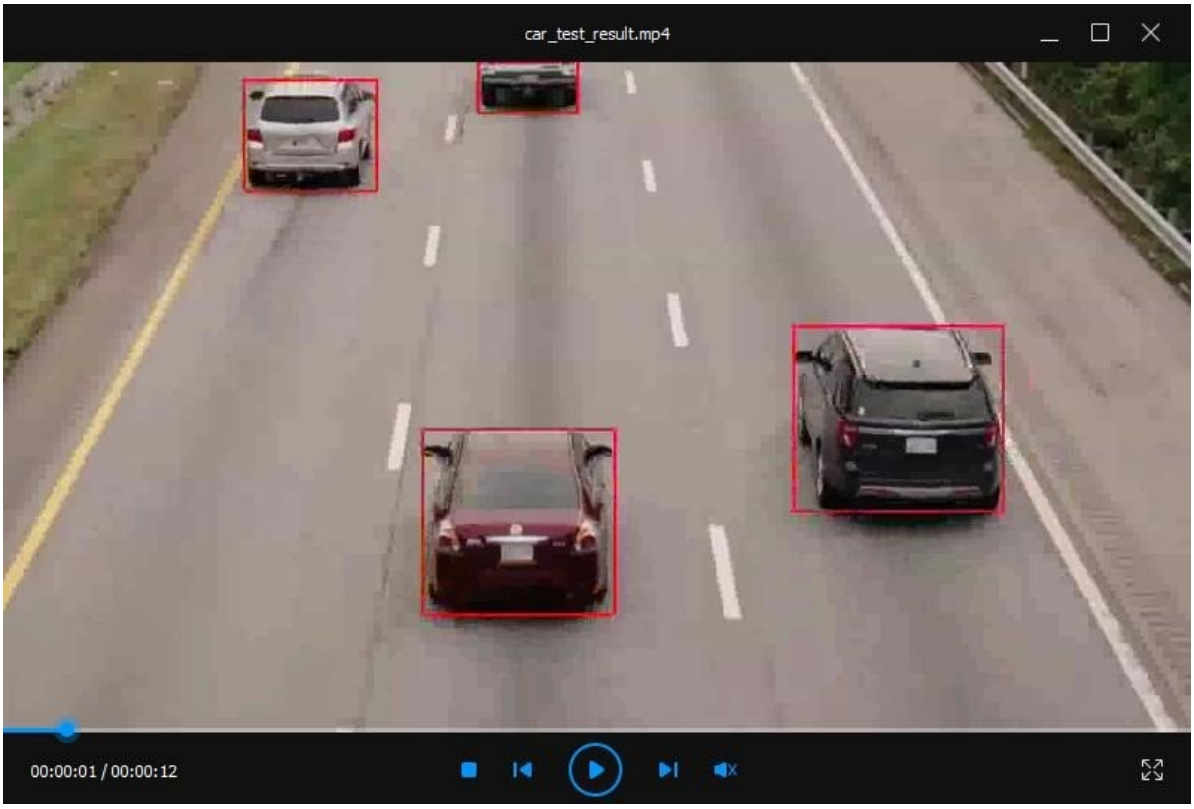
在`car_det`工程路径下执行`build_project.sh`进行工程构建：

```
rock@rock-3a:~/███/modelbox/workspace/car_det$ ./build_project.sh

build success: you can run main.sh in ./run folder

rock@rock-3a:~/███/modelbox/workspace/car_det$
```

切换到`root`账号，执行`run/main.sh`运行应用，运行结束后在`data`目录下生成了`car_test_result.mp4`文件，可以下载到PC端查看。



7）查看HTTP流程图

除了开发视频推理类应用，我们还可以使用ModelBox开发HTTP服务类应用，打开`car_det/graph/car_det_http.toml`，看到流程定义graphconf如下：

```
graphconf = """digraph car_det {
    node [shape=Mrecord];
    queue_size = 4
    batch_size = 1
    httpserver_sync_receive[type=flowunit, flowunit=httpserver_sync_receive,
device=cpu, deviceid=0, time_out_ms=5000,
endpoint="http://0.0.0.0:8083/v1/car_det", max_requests=100]

    image_decoder[type=flowunit, flowunit=image_decoder, device=rknpu,
deviceid=0, key="image_base64"]
    image_resize[type=flowunit, flowunit=resize, device=rknpu, deviceid=0,
width=512, height=288]
    car_detection[type=flowunit, flowunit=yolox_infer, device=rknpu, deviceid=0]
    yolox_post[type=flowunit, flowunit=yolox_post, device=cpu, deviceid=0]

    httpserver_sync_reply[type=flowunit, flowunit=httpserver_sync_reply,
device=cpu, deviceid=0]

    httpserver_sync_receive:out_request_info -> image_decoder:in_encoded_image
    image_decoder:out_image -> image_resize:in_image
    image_resize:out_image -> car_detection:input
    car_detection:output -> yolox_post:in_feat
    yolox_post:out_data -> httpserver_sync_reply:in_reply_info
}"""
```

该流程图从HTTP请求中接收一张图片进行车辆检测，并返回检测结果。

8）运行HTTP应用

我们使用root账号运行`car_det_http.toml`流程图（注意命令后的http参数）：

```
root@rock-3a:/home/rock/███/modelbox/workspace/car_det# ./run/main.sh http
[2022-03-26 08:18:45,882][ INFO][          flow.cc:97  ] run flow /home/rock/
███/modelbox/workspace/car_det/run/etc/graph/car_det_http.toml
[2022-03-26 08:18:45,917][ INFO][        driver.cc:549 ] Gather scan info
success, drivers count 47
[2022-03-26 08:18:45,923][ INFO][        driver.cc:357 ] load success drivers:
count 47, show detail in debug level
…
[2022-03-26 08:18:48,350][ INFO][         graph.cc:1094] port connect,
image_decoder:out_image -> yolox_post:in_image
[2022-03-26 08:18:48,351][ INFO][flow_scheduler.cc:69  ] init scheduler with 12
threads, max 384
```

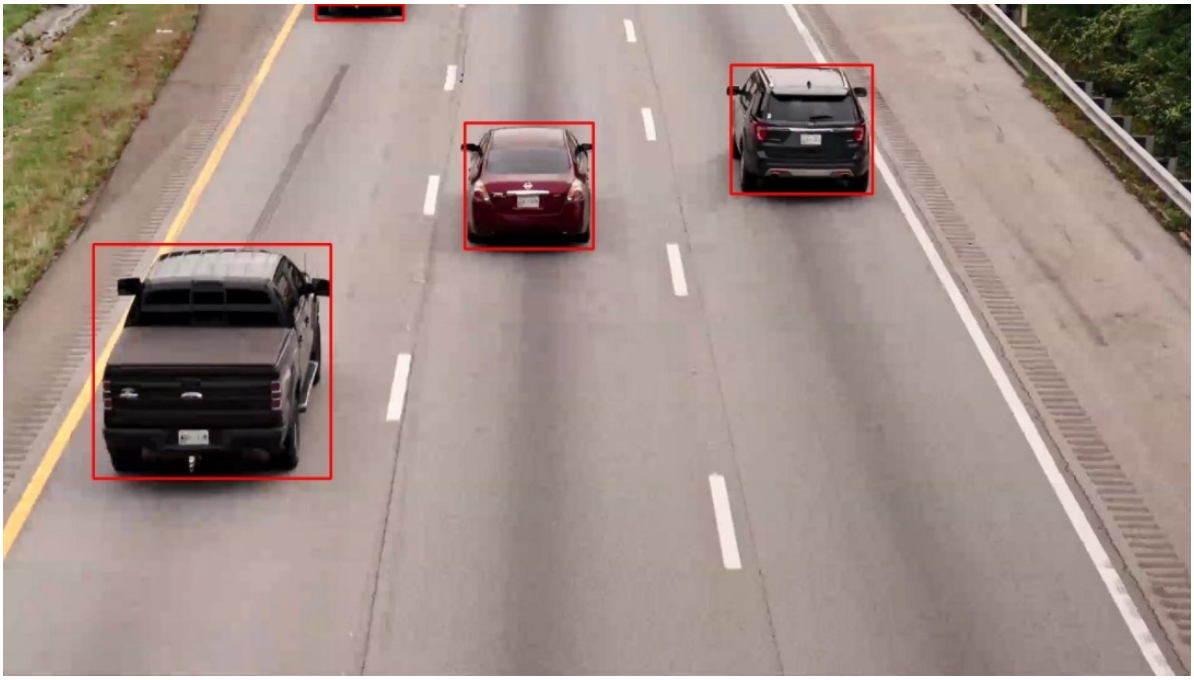
此时HTTP服务已启动，等待调用。

9）调用HTTP服务

在`car_det/data`目录下我们准备了HTTP调用的测试脚本`test_http.py`和测试图片`car_test_pic.jpg`，可以看到默认是在本机发起调用请求：

```
…
if __name__ == "__main__":
    port = 8083
    ip = "127.0.0.1"
    url = "/v1/car_det"
    img_path = "./car_test_pic.jpg"
    test_image(img_path, ip, port, url)
```

我们在VS Code中打开另一个终端，执行该脚本，将在`car_det/data`下生成测试图片的推理结果`car_test_pic.jpg`：



当然，也可以将测试脚本和图片拷贝到PC上，并将`test_http.py`中的ip变量修改为盒子的ip进行远程调用测试。

至此，我们的第一个AI应用就已经开发好了。
有关ModelBox核心概念、功能单元和流程图开发的更多介绍，可查看ModelBox手册。