

Milk-V Duo开发板实战——基于YOLOv5的目标检测

1.配置docker开发环境

docker安装

在windows环境下，可以安装Docker Desktop for Windows, [docker下载地址](#)

Install Docker Desktop on Windows

Welcome to Docker Desktop for Windows. This page contains information about Docker Desktop for Windows system requirements, download URL, instructions to install and update Docker Desktop for Windows.

Docker Desktop for Windows

For checksums, see [Release notes](#)

i Docker Desktop terms

Commercial use of Docker Desktop in larger enterprises (more than 250 employees OR more than \$10 million USD in annual revenue) requires a paid subscription.

System requirements

You must meet the following requirements to successfully install Docker Desktop on Windows:

WSL 2 backend

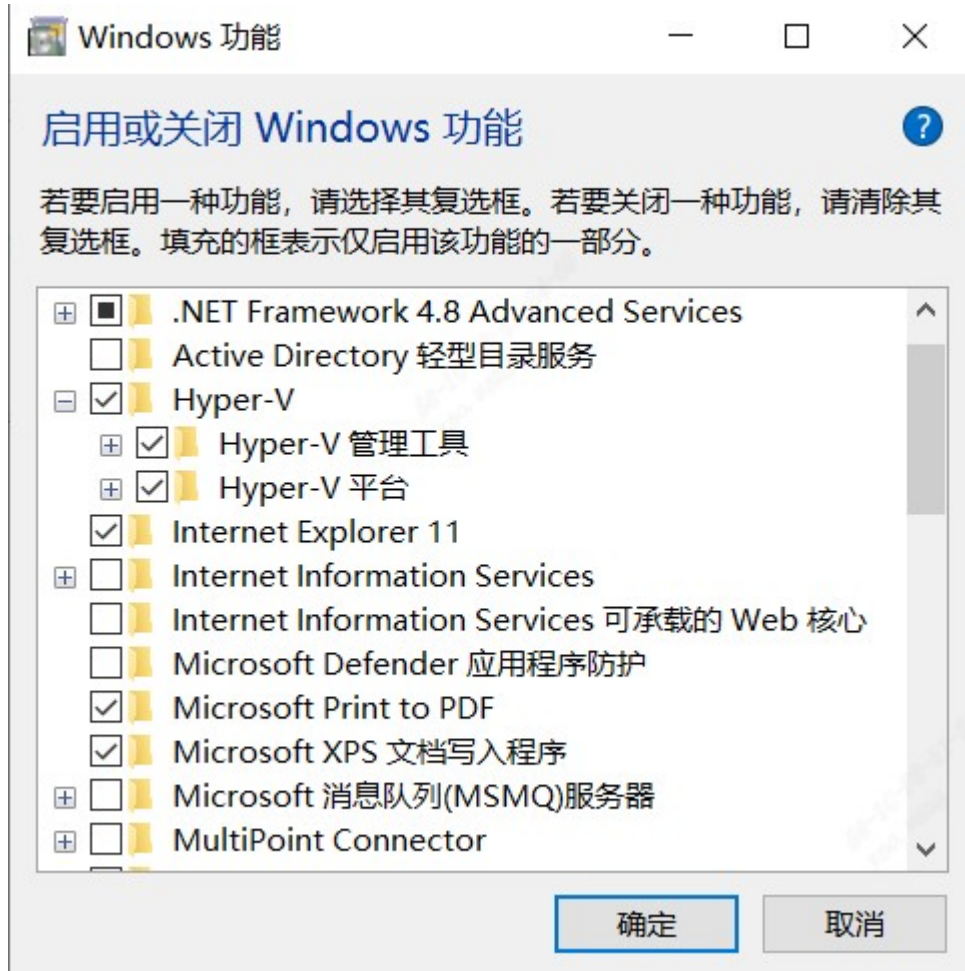
Hyper-V backend and Windows containers

在Windows下运行docker需要相关依赖，即如图中所示，需要使用WSL2后端或者Hyper-V后端作为运行依赖

Hyper-V后端的启用方式如下：

1. 控制面板 —— 程序 —— 启用或关闭Windows功能

2. 找到Hyper-V，勾选Hyper-V管理工具和Hyper-V平台，等待系统文件配置完成后重启电脑



然后即可安装下载好Docker Desktop for Windows，在安装指引中根据选择的后端进行相应的勾选

安装完成后，需要重启电脑，然后即可使用docker

拉取开发所需docker镜像

从docker hub获取镜像文件：

```
docker pull sophgo/tpuc_dev:v2.2
```

启动容器

```
docker run --privileged --name <container_name> -v /workspace -it  
sophgo/tpuc_dev:v2.2
```

其中，<container_name>为自己定义的容器名

获取开发工具包

可以从下载站台中获取开发工具包`tpu-mlir_xxxx.tar.gz`，`xxxx`为版本号，如`tpu-mlir_v1.2.89-g77a2268f-20230703`：

```
sftp://218.17.249.213
username: cvitek_mlir_2023
password: 7&2Wd%cu5k
```

另外，也可以从github上下载，[下载地址](#)

拷贝开发工具包

新建一个终端，并将开发工具包从windows拷贝到docker容器中

```
docker cp <path>/tpu-mlir_xxxx.tar.gz <container_name>:/tpu-
mlir_xxxx.tar.gz
```

其中，`<path>`为windows系统中开发工具包所在的文件目录，`<container_name>`为容器名

将工具包解压并添加环境变量

在docker容器中，解压工具包并添加环境变量

```
$ tar -zxvf tpu-mlir_xxxx.tar.gz
$ source ./tpu-mlir_xxxx/envsetup.sh
```

2.在windows下准备原始模型文件

准备yolov5开发工具包和yolov5n.pt文件

下载[yolov5开发工具包](#)以及[yolov5n.pt](#)文件，下载完成后将工具包解压，并将[yolov5n.pt](#)文件放在[yolov5-master](#)/目录下

配置conda环境

新建Anaconda Prompt终端，新建conda虚拟环境并安装3.9.0版本的python，然后可使用如下指令安装1.12.1版本的pytorch，具体安装指令可根据需求选择，后续过程只需要用到CPU即可

```
# CUDA 10.2
conda install pytorch==1.12.1 torchvision==0.13.1 torchaudio==0.12.1
                        cudatoolkit=10.2 -c pytorch
# CUDA 11.3
```

```
conda install pytorch==1.12.1 torchvision==0.13.1 torchaudio==0.12.1
  cudatoolkit=11.3 -c pytorch
# CUDA 11.6
conda install pytorch==1.12.1 torchvision==0.13.1 torchaudio==0.12.1
  cudatoolkit=11.6 -c pytorch
# CPU Only
conda install pytorch==1.12.1 torchvision==0.13.1 torchaudio==0.12.1
  cpuonly -c pytorch
```

然后将终端路径cd到开发工具包的yolov5-master/路径下，输入pip install -r requirements.txt安装其他依赖项

生成原始模型文件

在yolov5-master/目录下新建一个main.py文件，并在文件中写入如下代码：

```
import torch
from models.experimental import attempt_download
model = torch.load(attempt_download("./yolov5n.pt"),
  map_location=torch.device('cpu'))['model'].float()
model.eval()
model.model[-1].export = True
torch.jit.trace(model, torch.rand(1, 3, 640, 640),
  strict=False).save('./yolov5n_jit.pt')
```

然后找到/yolov5-master/models/yolo.py文件，将第63行到第79行的代码注释，并在第80行添加代码return x，如下图所示：

```
56 def forward(self, x):
57     z = [] # inference output
58     for i in range(self.nl):
59         x[i] = self.m[i](x[i]) # conv
60         bs, _, ny, nx = x[i].shape # x(bs,255,20,20) to x(bs,3,20,20,85)
61         x[i] = x[i].view(bs, self.na, self.no, ny, nx).permute(0, 1, 3, 4, 2).contiguous()
62
63         # if not self.training: # inference
64         #     if self.dynamic or self.grid[i].shape[2:4] != x[i].shape[2:4]:
65         #         self.grid[i], self.anchor_grid[i] = self._make_grid(nx, ny, i)
66         #
67         # if isinstance(self, Segment): # (boxes + masks)
68         #     xy, wh, conf, mask = x[i].split((2, 2, self.nc + 1, self.no - self.nc - 5), 4)
69         #     xy = (xy.sigmoid() * 2 + self.grid[i]) * self.stride[i] # xy
70         #     wh = (wh.sigmoid() * 2) ** 2 * self.anchor_grid[i] # wh
71         #     y = torch.cat((xy, wh, conf.sigmoid(), mask), 4)
72         # else: # Detect (boxes only)
73         #     xy, wh, conf = x[i].sigmoid().split((2, 2, self.nc + 1), 4)
74         #     xy = (xy * 2 + self.grid[i]) * self.stride[i] # xy
75         #     wh = (wh * 2) ** 2 * self.anchor_grid[i] # wh
76         #     y = torch.cat((xy, wh, conf), 4)
77         #     z.append(y.view(bs, self.na * nx * ny, self.no))
78         #
79         # return x if self.training else (torch.cat(z, 1), ) if self.export else (torch.cat(z, 1), x)
80     return x
```

修改完成后，运行`main.py`文件，就会在`yolov5-master`目录下生成`yolov5n_jit.pt`文件，该文件即为所需的原始模型文件

3.在docker中准备工作目录

建立`yolov5n_torch`工作目录，注意是与`tpu-mlir_xxxx`同级的目录，并将模型文件和图片文件都放入该目录下

```
$ mkdir yolov5n_torch && cd yolov5n_torch
```

新建一个终端，将`yolov5n_jit.pt`从windows拷贝到docker中

```
docker cp <path>/yolov5-master/yolov5n_jit.pt  
<container_name>:/workspace/yolov5n_torch/yolov5n_jit.pt
```

其中，`<path>`为windows系统中yolov5开发工具包所在的文件目录，`<container_name>`为容器名

再将图片文件放入`yolov5n_torch/`目录下并建立`work`目录

```
$ cp -rf $TPUC_ROOT/regression/dataset/COC02017 .  
$ cp -rf $TPUC_ROOT/regression/image .  
$ mkdir work && cd work
```

这里的`$TPUC_ROOT`是环境变量，对应`tpu-mlir_xxxx`目录

4.TORCH转MLIR

本例中，模型是RGB输入，`mean`和`scale`分别为`0,0,0`和`0.0039216,0.0039216,0.0039216`将torch模型转换为mlir模型的命令如下：

```
$ model_transform.py \  
  --model_name yolov5n \  
  --model_def ../yolov5n_jit.pt \  
  --input_shapes [[1,3,640,640]] \  
  --pixel_format "rgb" \  
  --keep_aspect_ratio \  
  --mean 0,0,0 \  
  --scale 0.0039216,0.0039216,0.0039216 \  
  --test_input ../image/dog.jpg \  
  --test_result yolov5n_top_outputs.npz \  
  --output_names 1219,1234,1249 \  
  --mlir yolov5n.mlir
```

运行成功效果示例：

```
[1219          ] SIMILAR [PASSED]
(1, 255, 80, 80) float32
cosine_similarity = 1.000000
euclidean_similarity = 1.000000
sqnr_similarity = 119.358044
[1234          ] SIMILAR [PASSED]
(1, 255, 40, 40) float32
cosine_similarity = 1.000000
euclidean_similarity = 1.000000
sqnr_similarity = 118.279133
[1249          ] SIMILAR [PASSED]
(1, 255, 20, 20) float32
cosine_similarity = 1.000000
euclidean_similarity = 1.000000
sqnr_similarity = 118.587103
200 compared
200 passed
1 equal, 3 close, 196 similar
0 failed
0 not equal, 0 not similar
min_similarity = (0.999998211860657, 0.999981805550603, 114.30965423583984)
Target yolov5n_top_outputs.npz
Reference yolov5n_ref_outputs.npz
npz compare PASSED.
compare 1249: 100% 200/200 [00:02<00:00, 76.38it/s]
[Success]: npz_tool.py compare yolov5n_top_outputs.npz yolov5n_ref_outputs.npz --tolerance 0.99,0.99 --except -vv
```

转成mlir模型后，会生成一个**yolov5n.mlir**文件，该文件即为mlir模型文件，还会生成一个**yolov5n_in_f32.npz**文件，该文件是后续转模型的输入文件

```
root@fdb7a81ccdb3:/workspace/yolov5n_torch/work# ls
yolov5n.mlir      yolov5n_origin.mlir      yolov5n_top_f32_all_weight.npz
yolov5n_in_f32.npz yolov5n_top_f32_all_origin_weight.npz yolov5n_top_outputs.npz
```

5.MLIR转INT8模型

生成校准表

在转int8模型之前需要先生成校准表，这里用现有的100张来自COCO2017的图片举例，执行calibration：

```
$ run_calibration.py yolov5n.mlir \
  --dataset ../COCO2017 \
  --input_num 100 \
  -o ./yolov5n_cali_table
```

运行完成后，会生成**yolov5n_cali_table**文件，该文件用于后续编译int8模型

```
root@fdb7a81ccdb3:/workspace/yolov5n_torch/work# ls
yolov5n.mlir      yolov5n_in_f32.npz      yolov5n_top_f32_all_origin_weight.npz  yolov5n_top_outputs.npz
yolov5n_cali_table yolov5n_origin.mlir      yolov5n_top_f32_all_weight.npz
```

编译为int8模型

将mlir模型转换为int8模型的命令如下：

```
$ model_deploy.py \
  --mlir yolov5n.mlir \
```



```

--quantize INT8 \
--calibration_table ./yolov5n_cali_table \
--chip cv180x \
--test_input ../image/dog.jpg \
--test_reference yolov5n_top_outputs.npz \
--compare_all \
--tolerance 0.96,0.72 \
--fuse_preprocess \
--debug \
--model yolov5n_int8_fuse.cvimodel

```

编译成功效果示例:

```

[1219_f32 ] EQUAL [PASSED]
(1, 255, 80, 80) float32
[1234_f32 ] EQUAL [PASSED]
(1, 255, 40, 40) float32
[1249_f32 ] EQUAL [PASSED]
(1, 255, 20, 20) float32
[1249_f32 ] EQUAL [PASSED]
(1, 255, 20, 20) float32
9 compared
9 passed
9 equal, 0 close, 0 similar
0 failed
0 not equal, 0 not similar
min_similarity = (1.0, 1.0, inf)
Target yolov5n_cv180x_int8_sym_model_outputs.npz
Reference yolov5n_cv180x_int8_sym_tpu_outputs.npz
npz compare PASSED.
compare 1249_f32: 100% 9/9 [00:00<00:00, 79.61it/s]
[Success]: npz_tool.py compare yolov5n_cv180x_int8_sym_model_outputs.npz yolov5n_cv180x_int8_sym_tpu_outputs.npz --tolerance 0.99,0.90 --except -vv

```

编译完成后, 会生成yolov5n_int8_fuse.cvimodel文件

```

root@fdb7a81ccdb3:/workspace/yolov5n_torch/work# ls
_weight_map.csv          yolov5n_int8_fuse.cvimodel
yolov5n.mlir             yolov5n_origin.mlir
yolov5n_cali_table       yolov5n_top_f32_all_origin_weight.npz
yolov5n_cv180x_int8_sym_final.mlir  yolov5n_top_f32_all_weight.npz
yolov5n_cv180x_int8_sym_model_outputs.npz  yolov5n_top_outputs.npz
yolov5n_cv180x_int8_sym_tpu.mlir  yolov5n_tpu_addressed_cv180x_int8_sym_weight.npz
yolov5n_cv180x_int8_sym_tpu_outputs.npz  yolov5n_tpu_addressed_cv180x_int8_sym_weight_fix.npz
yolov5n_in_f32.npz        yolov5n_tpu_lowered_cv180x_int8_sym_weight.npz
yolov5n_in_ori.npz

```

6.在Duo开发板上进行验证

连接Duo开发板

根据前面的教程完成duo开发板与电脑的连接, 并使用mobaxterm开启终端操作Duo开发板

获取cvitek_tpu_sdk

可以从下载站台中获取开发工具包cvitek_tpu_sdk_cv180x_musl_riscv64_rvv.tar.gz, 注意需要选择cv180x的工具包, 下载站台如下:

```
sftp://218.17.249.213
username: cvitek_mlir_2023
password: 7&2Wd%cu5k
```

下载完成后，拷贝到docker中并在docker中进行解压

```
$ tar -zxvf cvitek_tpu_sdk_cv180x_musl_riscv64_rvv.tar.gz
```

解压完成后会生成cvitek_tpu_sdk文件夹

将开发工具包和模型文件拷贝到开发板上

在duo开发板的终端中，新建文件目录/home/milkv/

```
$ mkdir /home/milkv && cd /home/milkv
```

在docker的终端中，将开发工具包和模型文件拷贝到开发板上

```
$ scp -r cvitek_tpu_sdk root@192.168.42.1:/home/milkv
$ scp /workspace/yolov5n_torch/work/yolov5n_int8_fuse.cvimodel
root@192.168.42.1:/home/milkv/cvitek_tpu_sdk
```

设置环境变量

在duo开发板的终端中，进行环境变量的设置

```
$ cd ./cvitek_tpu_sdk
$ source ./envs_tpu_sdk.sh
```

进行目标检测

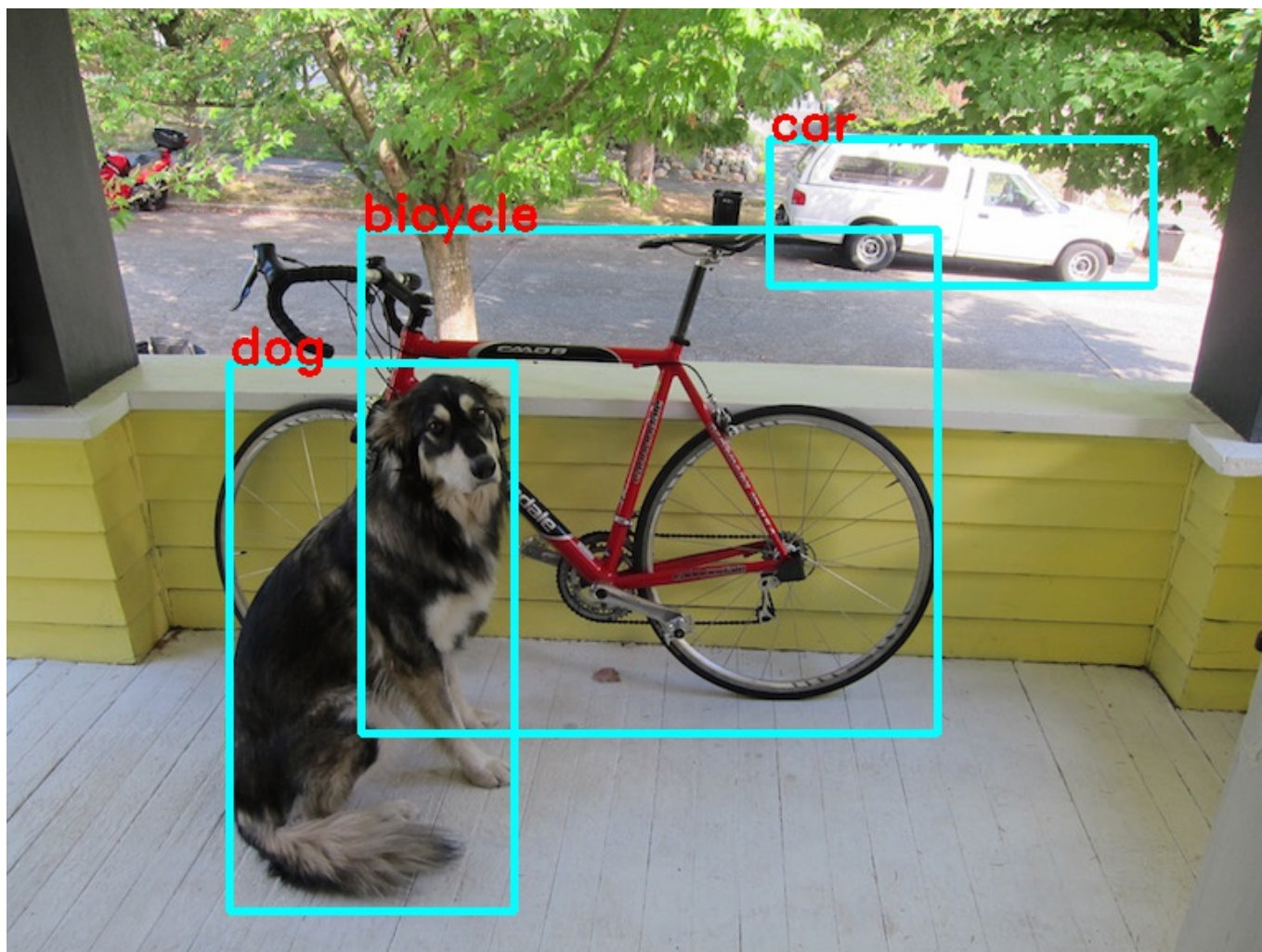
在duo开发板的终端中，输入如下命令进行目标检测

```
$ ./samples/samples_extra/bin/cvi_sample_detector_yolo_v5_fused_preprocess \
./yolov5n_int8_fuse.cvimodel \
./samples/samples_extra/data/dog.jpg \
yolov5n_out.jpg
```


检测成功结果示例:

```
CVI_NN_RegisterModel succeeded
get detection num: 3
obj 0: [222.069870 381.771240 173.271393 333.260345] score:0.733817 cls:dog
obj 1: [580.405090 123.967697 233.020157 89.783295] score:0.687424 cls:car
obj 2: [390.363068 287.245392 350.024567 305.919586] score:0.517759 cls:bicycle
-----
3 objects are detected
-----
CVI_NN_CleanupModel succeeded
```

运行成功后，会生成检测结果文件yolov5n_out.jpg



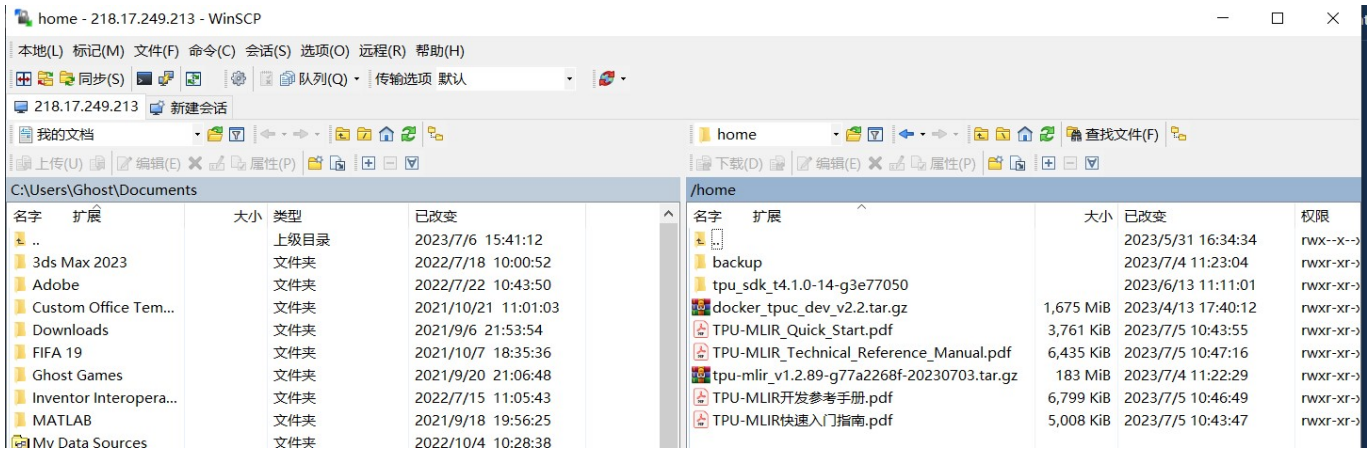
附录

正文涉及到的文件总结如下：

- TPU-MLIR模型转换工具链：tpu-mlir_v1.2.89-g77a2268f-20230703.tar.gz
- TPU SDK开发工具包：cvitek_tpu_sdk_cv180x_musl_riscv64_rvv.tar.gz
- （附）Sample测试例程源码：cvitek_tpu_samples.tar.gz
- （附）转换好的cvimodel包：cvimodel_samples_cv180x.tar.gz

正文提到的TPU开发所需的包文件可在下面sftp站点获取：

```
sftp://218.17.249.213 user: cvitek_mlir_2023 password: 7&2Wd%cu5k
```



或者直接使用wget获取：

TPU-MLIR模型转换工具链

```
wget --user='cvitek_mlir_2023' --password='7&2Wd%cu5k'
ftp://218.17.249.213/home/tpu-mlir_v1.2.89-g77a2268f-20230703.tar.gz
```

TPU SDK开发工具包

```
wget --user='cvitek_mlir_2023' --password='7&2Wd%cu5k'
ftp://218.17.249.213/home/tpu_sdk_t4.1.0-14-
g3e77050/cvitek_tpu_sdk_cv180x_musl_riscv64_rvv.tar.gz
```

(附) Sample测试例程源码

```
wget --user='cvitek_mlir_2023' --password='7&2Wd%cu5k'
ftp://218.17.249.213/home/tpu_sdk_t4.1.0-14-
g3e77050/cvitek_tpu_samples.tar.gz
```

(附) 转换好的cvimodel包

```
wget --user='cvitek_mlir_2023' --password='7&2Wd%cu5k'
ftp://218.17.249.213/home/tpu_sdk_t4.1.0-14-
g3e77050/cvimodel_samples_cv180x.tar.gz
```