Amlogic

NNEngine Tool Introduction

Revision: 0.1

Release Date: 2022-02-10

Copyright

© 2022 Amlogic. All rights reserved. No part of this document may be reproduced, transmitted, transcribed, or translated into any language in any form or by any means without the written permission of Amlogic.

Trademarks

Amlogic, and other Amlogic icons are trademarks of Amlogic companies. All other trademarks and registered trademarks are property of their respective holders.

Disclaimer

Amlogic may make improvements and/or changes in this document or in the product described in this document at any time.

This product is not intended for use in medical, life saving, or life sustaining applications.

Circuit diagrams and other information relating to products of Amlogic are included as a means of illustrating typical applications. Consequently, complete information sufficient for production design is not necessarily given. Amlogic makes no representations or warranties with respect to the accuracy or completeness of the contents presented in this document.

Contact Information

- Website: www.amlogic.com
- Pre-sales consultation: contact@amlogic.com
- Technical support: support@amlogic.com

Revision History

Issue 0.1 (2022-02-10)

This is the Initial release.

Contents

1. Introduction

nnEngine is a set of model reasoning engine based on nnsdk development kit developed by Amlogic. It uses ADB protocol to realize data transmission and model operation control between embedded device and server. Users can use the provided aml_nn_engine python library to implement model inference and performance index acquisition functions.

https://github.com/Amlogic-NN/AML_NN_SDK/tree/master/Tools/nnengine

For related introduction about nnsdk development kit, please see <DDK_6.4.8.7_SDK_V1.9.4 API Description.docx>

2. Introduction to nnEngine usage

2.1 import nn_engine python library

```
import sys
sys.path.append('aml_nn_engine_37')    #Specify aml_nn_engine library file path
```

Note：Need to select the aml_nn_engine library file corresponding to the python version. If the version does not match, the import may fail.

2.2 nn_engine API introduction

1. Init amlnn class

   amlnn = AMLNN()

2. Model init

   amlnn.init_runtime(config.linux_path, config.nb_path, config.model_type)

3. Set model input
   amlnn.set_input(config.input_name,config.input_data,config.input_type,config.input_w, config.input_h,config.input_c,config.run_cyc,config.modetype,config.profile)
4. Model forward processand obtain output results

   output=amlnn.inference()

5. Model destroy

   amlnn.destroy()

2.2.1 amlnn = AMLNN()

API	amlnn = AMLNN()
Function	Init amlnn class

2.2.2 amlnn.init_runtime()

API	amlnn.init_runtime(config.linux_path,config.nb_path,config.model_type,config.device_type)
Function	model init
Parameter	config.linux_path: set board work space
	config.nb_path: model nbg file path and name
	config.model_type: Set the model framework type, currently supportedcaffe, tensorflow, tensorflowlite, darknet, onnx, keras, pytorch
	config.device_type: Specify the device operating system type, divided into linux and android

example：

amlnn.init_runtime(“/media/nn/”,mobilenetv1_be.nb,tensorflow,linux)

2.2.3 amlnn.set_input()

API	amlnn.set_input(config.input_name,config.input_data,config.input_type,config.input_w,config.input_h,config.input_c,config.run_cyc,config.modetype,config.profile)
Function	Set model input and related control parameters
Parameter	config.input_name: Input file path and name, support jepg, tensor and binary format files
	config.input_data: Input raw data, such as using opencv to read the data of bmp images
	config.input_type:Data preprocessing type, rgb means model preprocessing operation using RGB24; tensor means model preprocessing operation using tensor; raw means no data preprocessing is performed, and the input is directly used as the input of the model
	config.input_w: input data width info
	config.input_h: input data high info
	config.input_c: input data channel info
	config.run_cyc: The number of model run_graph cycles, usually used to test the running time of the model
	config.modetype: For the custom model, please select 99 to return the model output data directly, and the user can perform post-processing operations on the python side
	config.profile: Get profile information, you can get fps and bandwidth information by setting to'all', set'runtime' to get model fps information, and set'bw' to get model bandwidth information

- *Note：*
- *1. Set_input supports jpeg, tensor, binary format file input, at this time config.input_name: set to the corresponding file name, set config.input_data to'None', set config.input_type to the specified format code, and set it up Enter the width/height/channel information of the data.*
- *2. Set_input supports the input of parsed data in various formats. At this time, config.input_name: is set to "None", and config.input_data is set to input data. The user can perform data pre-processing and set the input type to rawdata. If the user wants to use the preprocessing scheme of nnsdk, the input image can be parsed into RGB24 or tensor format data, and the input type is set to RGB24 or tensor.*
- *3.This version of the android platform does not currently support obtaining model bandwidth information. This function will be added in the next version.*
- *Example：*
- *amlnn.set_input('2242243.jpg',None,'rgb',224,224,3,1,0,all)*

2.2.4 amlnn.inference()

API	output=amlnn.inference(out_size)
Function	Get model inference results
Parameter	out_size：sockect transmission data size information
Return value	output：model output data

The output is the calculation result of the model through npu inference, and it can be verified whether the accuracy of the model is normal based on adding post-processing code on the python side. The out_size parameter defaults to 4096. If the output size is greater than 4096, you need to specify specific data. If it is a multi-output model, you need to set the total output size.

2.2.5 amlnn.destroy()

API	amlnn.destroy()

Function	Model destroy，free resource

3. Introduce how to use nnengine

Currently nnEngine supports running on windows10 operating system and python3.7 environment.

Windows environment configuration:

1. The PC terminal and the development board are connected through the USB interface, and ensure that the specified device can be recognized by ADB normally.

Execute the command in the shell: adb devices -l, the target device can be displayed normally; execute the command: adb shell

And use cmd or powershell to execute adb shell to ensure that adb functions normally

2. Configure the python environment,

```
        pip3 install numpy==1.20.1
        pip3 install opencv-python==4.5.4.58
        pip3 install easydict==1.9
```

a) windows platform environment preparation

In the windows environment, use cmd or powershell to enter the demo storage path, and execute python demo.py to get the final result of the classification model in the shell.

Before performing operations, you need to create a'/media/nn' folder on the side of the board, and place the nnEngine executable file in this path.

b) Android platform preparation

Before executing the operation, add corresponding permissions for android in the shell, then create a'/media/nn' folder on the board side, and place the nnEngine executable file in this path.

```
adb root
adb remount
adb shell
setenforce 0
```

c) Start NNEngine service(choose one of them)

Method 1: Board end execution: ./nnEngine

Method 2: PC side execution: python .\server.py

d) Run test demo

Execute on the PC side: python .\demo.py to get the output result

Image classify demo

```
output=amlnn.inference(4096)
for j in range(10+4,10+4+5):
      print("pcreceived top",j-4," is:",struct.unpack('f', struct.pack('4B', output[4*j], output[4*j+1],
output[4*j+2],output[4*j+3]))[0])
for i in range(4,7):
   print("profile",i-4,"is:",struct.unpack('f',struct.pack('4B', output[4*i], output[4*i+1], output[4*i +2],
output[4*i+3]))[0])
```

Object detect demo

```
output=amlnn.inference(425*13*13)      #output size is 425*13*13=71825
list_out=list(output[(4+10)*4:(71825+10+4)*4])#10:Store profile related data, 4: sockect information header
lista = []
out_lenth = (int)((len(list_out) )/4)
for i in range(out_lenth):
lista.append(struct.unpack('f',struct.pack('4B',list_out[4*i],list_out[4*i+1],list_out[4*i+2],list_out[4*i+3]))[0])
```