

NN SDK API 描述

Revision: 1.9.4


Release Date: 2022-02-10

Confidential!
nick@khadas.com
2022-03-17 19:26:37

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 with the written permission of Amlogic.

Trademarks

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

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

更改记录

版本 1.9.4 (2022-02-10)

第十二版。

- 1、第 1 章添加 nnsdk workflow 流程图。
- 2、添加第 3.5 小节模型性能 debug 接口介绍。
- 3、添加第 3.3.9 小节保存 NPU 输入信息

版本 1.8.2 (2021-06-30)

第十一版。

- 1、添加 aml_util_switchInputBuffer 和 aml_util_switchOutputBuffer API 描述
- 2、修改 nn_input 接口定义

版本 1.8.0 (2021-03-22)

第十版。

- 1、删除第 1 章 SDK 网络模型支持列表及 demo API 数据结构表
- 2、修改第 2.2 小节中 assign_user_address_t、aml_output_config_t、aml_perf_mode_t、input_info 描述。
- 3、删除原第三章。

版本 1.7.2 (2020-12-15)

第九版。

- 1、修改表 1.2 SDK API 接口表，添加表 1.3 SDK API 数据结构表，添加表 1.4 SDK Demo API 数据结构表
- 2、修改第 2.1.3, 2.1.4, 2.1.7, 2.1.10, 2.1.11, 2.1.13 小节示例代码
- 3、添加 aml_util_flushTensorHandle 和 aml_util_getHardwareStatusAPI 描述
- 4、修改第 3.1.1 小节 SDK 集成 demo 调用流程说明描述
- 5、添加第 3.1.2 小节 demo 框架类型描述
- 6、修改第 3.1.3 小节 opencv 依赖环境描述
- 7、修改第 3.1.4 小节参考代码
- 8、添加第 3.2.4 小节 mean 值不一致设置方式示例代码
- 9、添加第 3.2.5 小节 usb camera 示例代码
- 10、修改第 3.2.8 小节 SDK - HardwareStatus

11、修改第 3.2.11 小节参考代码

版本 1.7 (2020-09-25)

第八版。

- 1、添加 ffmpeg 解析视频接口介绍。
- 1、修改 aml_module_output_get 接口使用方式介绍。

版本 1.6 (2020-07-31)

第七版。

- 1、添加 setProfile 接口相关介绍。
- 2、添加 setPowerPolicy 接口相关介绍。
- 2、添加 usb_camera 使用方式介绍。
- 3、修改自定义模型调用方式介绍。
- 4、添加 tensor_info 和 info 结构介绍。

版本 1.5 (2020-06-23)

第六版。

- 1、添加 aml_util_getInputTensorInfo / aml_util_getOutputTensorInfo 接口相关介绍。
- 2、添加 aml_util_freeTensorInfo 接口相关介绍。
- 3、添加 aml_util_flushTensorHandle 接口相关介绍。
- 4、删除原第三四章，添加 SDK API 使用方法介绍。
- 5、支持 FACE_EMOTION、HEAD_DETECTION、PERSON_DETECT 模型

版本 1.4 (2020-05-25)

第五版。

- 1、修改 aml_config 定义。
- 2、修改 SDK Demo 和 CUSTOM_NETWORK 参考代码。
- 3、支持 FACE_LANDMARK_68 和 IMAGE_SEGMENTATION 模型

版本 1.3 (2020-04-22)

第四版。

- 1、修改 outBuf_t 定义。

2、修改 SDK Demo 和 CUSTOM_NETWORK 参考代码。

版本 1.2 (2020-03-24)

第三版。

- 1、添加 input dma swap 功能介绍。
- 2、添加 aml_util_swapInputBuffer 接口介绍。
- 3、修改 CUSTOM_NETWORK 基本调用流程介绍。
- 4、修改 Demo 和 CUSTOM_NETWORK 参考代码。

版本 1.1 (2020-02-27)

第二版。

- 1、添加 input/output dma 功能介绍。
- 2、添加 API aml_util_mallocAlignedBuffer 和 aml_util_freeAlignedBuffer 接口介绍。
- 3、添加 assign_user_address_t、aml_output_config_t、aml_output_format_t 数据结构介绍。
- 4、修改 face_gender_out_t 结构。
- 5、添加 micro-OpenCV 控制功能介绍。

版本 1.0 (2019-12-12)

第一版。

目录

更改记录.....	2
目录.....	5
1. 简介	7
2. SDK API 及数据结构介绍	10
2.1 Amlogic SDK API 详细说明	10
2.1.1 aml_module_create.....	10
2.1.2 aml_module_input_set.....	11
2.1.3 aml_module_output_get	11
2.1.4 aml_module_output_get_simple.....	12
2.1.5 aml_module_destroy.....	12
2.1.6 aml_util_mallocAlignedBuffer.....	12
2.1.7 aml_util_freeAlignedBuffer.....	13
2.1.8 aml_util_swapInputBuffer.....	13
2.1.9 aml_util_swapOutputBuffer.....	14
2.1.10 aml_util_flushTensorHandle	14
2.1.11 aml_util_getInputTensorInfo	15
2.1.12 aml_util_getOutputTensorInfo.....	15
2.1.13 aml_util_freeTensorInfo	16
2.1.14 aml_util_setProfile.....	16
2.1.15 aml_util_setPowerPolicy.....	17
2.1.16 aml_util_getHardwareStatus.....	17
2.1.17 aml_util_switchInputBuffer	18
2.1.18 aml_util_switchOutputBuffer	18
2.2 Amlogic SDK 数据结构详细说明	19
2.2.1 aml_config.....	19
2.2.2 assign_user_address_t.....	20
2.2.3 aml_output_config_t.....	21
2.2.4 aml_perf_mode_t.....	21
2.2.5 aml_output_format_t.....	22
2.2.6 image_out_t	22
2.2.7 tensor_info.....	22
2.2.8 info.....	23

2.2.9 detBox	24
2.2.10 point_t.....	24
2.2.11 nn_output	24
2.2.12 outBuf_t.....	25
2.2.13 nn_buffer_params_t.....	25
2.2.14 nn_input.....	27
3. SDK API 使用方法介绍	28
3.1 SDK 自定义模型调用流程.....	28
3.2 SDK 基本调用流程说明	29
3.2.1 模型初始化.....	29
3.2.2 设置输入	31
3.2.3 获取模型结果	32
3.3 SDK 拓展功能介绍.....	33
3.3.1 SDK - DMA 功能介绍	33
3.3.2 SDK - GetTensorInfo 功能介绍	36
3.3.3 SDK - mean 值不一致设置方式介绍	37
3.3.4 SDK - usb camera 功能介绍	37
3.3.5 SDK - profile 控制方式介绍	38
3.3.6 SDK - PowerPolicy 控制方式介绍	39
3.3.7 SDK - HardwareStatus 信息获取方式介绍.....	39
3.3.8 ffmpeg 解析视频接口介绍	39
3.3.9 SDK - 保存 NPU 输入信息.....	40
3.4 参考代码	40
3.5 模型性能 debug 接口	42

1. 简介

Amlogic SDK 是一套已经集成多个常用模型，可以提供接口给客户直接调用并获取对应结果，且支持客户添加自定义模型的开发包。

本文介绍 Amlogic SDK API 接口以及相关的数据结构，用以指导 amlogic 相关开发人员以及 amlogic 客户。

SDK API 接口信息如表 1.1 所示，SDK API 数据结构信息如表 1.2 所示。SDK workflow 如图 1.1 所示。

第一章：简介，介绍本文档的结构信息和主要内容。

第二章：SDK API 及数据结构介绍，详细介绍 SDK API 的功能、参数及返回值；详细介绍 SDK 定义的数据结构。

第三章：SDK API 使用方法介绍，介绍 SDK API 的使用方式并附上一段示例代码以供参考。

表 1.1 SDK API 接口表

SDK API	<code>void* aml_module_create(aml_config* config);</code>
	<code>int aml_module_input_set(void* context, nn_input *pInput);</code>
	<code>void* aml_module_output_get(void* context, aml_output_config_t outconfig)</code>
	<code>void* aml_module_output_get_simple(void* context);</code>
	<code>int aml_module_destroy(void* context);</code>
	<code>unsigned char * aml_util_mallocAlignedBuffer(int mem_size);</code>
	<code>void aml_util_freeAlignedBuffer(unsigned char *addr);</code>
	<code>int aml_util_swapInputBuffer(void *context, void *newBuffer, unsigned int inputId);</code>
	<code>int aml_util_swapOutputBuffer(void *context, void *newBuffer, unsigned int outputId);</code>
	<code>tensor_info* aml_util_getInputTensorInfo(const char* nbldata);</code>
	<code>tensor_info* aml_util_getOutputTensorInfo(const char* nbldata);</code>
	<code>void aml_util_freeTensorInfo(tensor_info* tinfo);</code>
	<code>int aml_util_flushTensorHandle(void* context, aml_flush_type_t type);</code>
	<code>int aml_util_setProfile(aml_profile_type_t type, const char *savepath);</code>

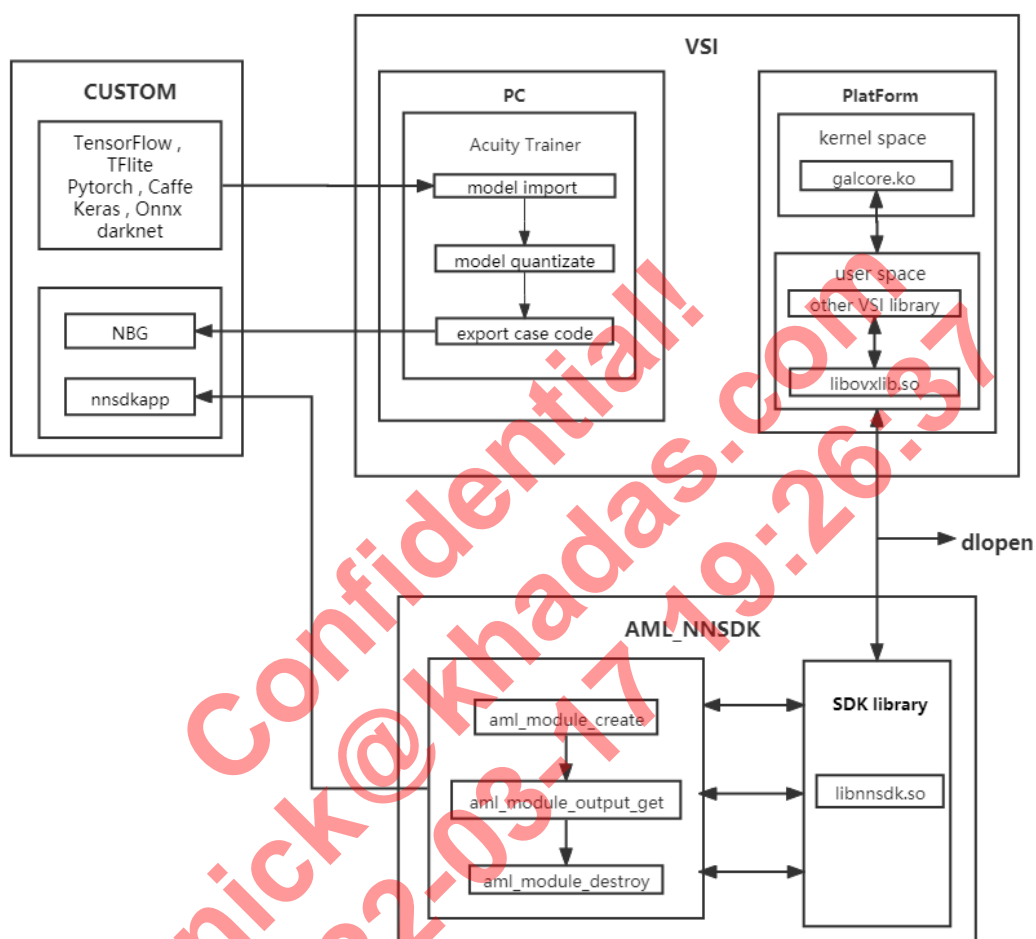
	int aml_util_getHardwareStatus(int *customID,int *powerStatus);
	int aml_util_switchInputBuffer(void *context,void *newBuffer,unsigned int inputId);
	int aml_util_switchOutputBuffer(void *context,void *newBuffer,unsigned int outputId);
	int aml_util_getHardwareStatus(int *customID,int *powerStatus);

表 1.2 SDK API 数据结构表

结构体	功能
aml_config	输出 config 信息
assign_user_address_t	设置用户定义内存地址
aml_output_config_t	设置输出模型及数据类型信息
aml_output_format_t	设置输出数据类型
amlInn_input_mode_t	设置输入模式
amlInn_model_type	设置框架类型
amlInn_nbg_type	设置 nbg 文件读取方式
amlInn_input_type	设置输入数据类型
amlInn_query_cmd	设置 query 类型
nn_buffer_format_e	设置输出 buffer 类型
nn_buffer_quantize_format_e	设置量化类型
aml_module_t	设置 demo 类型
aml_profile_type_t	设置 profile 类型
aml_policy_type_t	设置 policy 类型
aml_flush_type_t	设置 flush tensor 类型
assign_user_address_t	设置分配地址类型

image_out_t	输出图像结果
tensor_info	输出 tensor 信息

图 1.1 SDK workflow 流程图



2. SDK API 及数据结构介绍

2.1 Amlogic SDK API 详细说明

```
void* aml_module_create(aml_config* config);
int aml_module_input_set(void* context, nn_input *pInput);
void* aml_module_output_get(void* context, aml_output_config_t outconfig);
void* aml_module_output_get_simple(void* context);
int aml_module_destroy(void* context);
unsigned char * aml_util_mallocAlignedBuffer(int mem_size);
void aml_util_freeAlignedBuffer(unsigned char *addr);
int aml_util_swapInputBuffer(void *context, void *newBuffer, unsigned int inputId);
int aml_util_swapOutputBuffer(void *context, void *newBuffer, unsigned int outputId);
int aml_util_switchInputBuffer(void *context, void *newBuffer, unsigned int inputId);
int aml_util_switchOutputBuffer(void *context, void *newBuffer, unsigned int outputId);
tensor_info* aml_util_getInputTensorInfo(const char* nbldata);
tensor_info* aml_util_getOutputTensorInfo(const char* nbldata);
void aml_util_freeTensorInfo(tensor_info* tinfo);
int aml_util_flushTensorHandle(void* context, aml_flush_type_t type);
int aml_util_setProfile(aml_profile_type_t type, const char *savepath);
int aml_util_setPowerPolicy(aml_policy_type_t type);
int aml_util_getHardwareStatus(int *customID, int *powerStatus);
```

2.1.1 aml_module_create

API	void* aml_module_create(aml_config* config);
功能	初始化句柄，加载生成的 nbg 模型
参数	aml_config* config: nbg 模型的路径及类型信息
返回值	void*: 句柄 context 对象指针

示例代码：

```
void *context;
context = aml_module_create(&config);
```

2.1.2 aml_module_input_set

API	int aml_module_input_set(void* context, nn_input *pInput);
功能	根据 nbg 模型设置输入数据
参数	void* context: 句柄 context 指针
	nn_input *pInput: input 数据, 当前单次只能设置一个 input, 如果是多输入模型, 暂时需要调用多次 amlnn_inputs_set 接口, 后面会修正。
返回值	int: 成功返回零, 失败返回非零。

示例代码:

```
nn_input inData;
unsigned char *rawdata;

rawdata = get_jpeg_rawData(jpeg_path, 224, 224);
inData.input_index = 0; //this value is index of input, begin from 0
inData.size = 224*224*3;
inData.input = rawdata;
inData.input_type = RGB24_RAW_DATA;
ret = aml_module_input_set(context, &inData);
```

2.1.3 aml_module_output_get

API	void* aml_module_output_get(void* context, aml_output_config_t outconfig)	
功能	获取一次推理的结果	
参数	void* context: 句柄 context 指针	
	aml_output_config_t outconfig: 模型类型及输出类型	
	aml_module_t mdType: 模型类型	aml_output_format_t format: 输出类型
返回值	void*: 推理结果结构体指针	

示例代码:

```
img_classify_out_t *cls_out = NULL;
modelType = IMAGE_CLASSIFY;
outconfig.format = AML_OUTDATA_FLOAT32;
outdata = (nn_output*)aml_module_output_get(context, outconfig);
cls_out = (img_classify_out_t*)post_process_all_module(modelType, outdata);
for(i = 0; i < 5; i++){
    printf("top %d: score--%f, class--%d\n", i, cls_out->score[i], cls_out->topClass[i]);
}
```

}

2.1.4 aml_module_output_get_simple

API	void* aml_module_output_get_simple(void* context);
功能	获取一次推理的结果，输出类型为 float32
参数	void* context: 句柄 context 指针
返回值	void*: 推理结果结构体指针

示例代码：

```
nn_output *outdata = NULL;
outdata = (nn_output *)aml_module_output_get_simple(qcontext);
process_top5((float*)outdata->out[0].buf,outdata->out[0].size/sizeof(float),NULL);
```

2.1.5 aml_module_destroy

API	int aml_module_destroy(void* context);
功能	卸载 nbg 模型并销毁 context 以及相关资源
参数	void* context: 句柄 context 指针
返回值	int: 成功返回零，失败返回非零。

示例代码：

```
int ret = aml_module_destroy(context);
```

2.1.6 aml_util_mallocAlignedBuffer

API	unsigned char * aml_util_mallocAlignedBuffer(int mem_size);
功能	为 inbuf 或 outbuf 申请内存空间,以使用 input/output DMA 功能。详见示例
参数	int mem_size: 分配内存空间大小。
返回值	unsigned char *: 返回申请地址的指针。

示例代码：

```
unsigned char * inbuf = aml_util_mallocAlignedBuffer(224*224*3);
```

```

unsigned char * outbuf = aml_util_mallocAlignedBuffer(1000*2);
//设置 input 与 output dma 地址, 设置完成后, 输入数据可直接拷贝到 inbuf 中,
inbuf 会直接映射到 npu 空间, 不需要额外操作; 同样, outbuf 设置后, 在
aml_module_output_get 的时候, 系统会直接解析 outbuf 的数据, 省去 output 数据从 npu
拷贝到 cpu 的过程, 达到减少系统整体运行时间的目的。
config.inOut.outAddr[0] = outbuf;
config.inOut.inAddr[0] = inbuf;
config.modelType = TENSORFLOW;
context = aml_module_create(&config);

```

2.1.7 aml_util_freeAlignedBuffer

API	void aml_util_freeAlignedBuffer(unsigned char *addr);
功能	释放使用 aml_util_mallocAlignedBuffer 分配的 inbuf dma 或 outbuf dma 内存空间。
参数	unsigned char *addr: aml_util_mallocAlignedBuffer 返回的指针
返回值	NULL。

示例代码:

```

modelType = IMAGE_CLASSIFY;
outconfig.format = AML_OUTDATA_FLOAT32;
outdata = (nn_output*)aml_module_output_get(qcontext,outconfig);
cls_out = (img_classify_out_t*)post_process_all_module(modelType,outdata);
aml_util_freeAlignedBuffer(inbuf);
aml_util_freeAlignedBuffer(outbuf);
aml_module_destroy(context);

```

2.1.8 aml_util_swapInputBuffer

API	int aml_util_swapInputBuffer(void *context,void *newBuffer,unsigned int inputId);
功能	修改输入 DMA buffer 的起始指针, 实现 input 的快速切换
参数	void *context:句柄 context 指针。
	void *newBuffer:输入的 buffer 信息。
	unsigned int inputId: 输入 ID, 从 0 开始。
返回值	int: 成功返回零, 失败返回非零。

示例代码：

```
unsigned char * outbuf = aml_util_mallocAlignedBuffer(1000*2);
unsigned char * inbuf1 = aml_util_mallocAlignedBuffer(224*224*3);
unsigned char * inbuf2 = aml_util_mallocAlignedBuffer(224*224*3);
config.inOut.outAddr[0] = outbuf;
config.inOut.inAddr[0] = inbuf1;
config.inOut.io_type = AML_IO_VIRTUAL;
config.modelType = TENSORFLOW;
context = aml_module_create(&config);
memcpy(inbuf1,rawdata1,224*224*3);
//执行前向推理与后处理后，切换第二帧图片
memcpy(inbuf2,rawdata2,224*224*3); //该操作可以在第一帧执行过程中准备好，形成类似 ping-pang buffer 的效果，以省去 input 输入的时间。
aml_util_swapInputBuffer(context,(void*)inbuf2,0);
```

2.1.9 aml_util_swapOutputBuffer

API	int aml_util_swapOutputBuffer(void *context,void *newBuffer,unsigned int outputId);
功能	修改输出 DMA buffer 的起始指针，实现 output 的快速切换
参数	void *context:句柄 context 指针。
	void *newBuffer:输出的 buffer 信息。
	unsigned int outputId: 输出 ID，从 0 开始。
返回值	int: 成功返回零，失败返回非零。

示例代码：

```
unsigned char * outbuf = aml_util_mallocAlignedBuffer(1000*2);
config.inOut.outAddr[0] = outbuf;
config.inOut.io_type = AML_IO_VIRTUAL;
context = aml_module_create(&config);
aml_util_swapOutputBuffer(context,(void*)outbuf,0);
aml_module_output_get(context,outconfig);
```

2.1.10 aml_util_flushTensorHandle

API	int aml_util_flushTensorHandle(void* context,aml_flush_type_t type);
功能	刷新 tensor handle
参数	void *context:句柄 context 指针。

	aml_flush_type_t type:tensor 类型，分为 input 和 output 两种 tensor
返回值	NULL

示例代码：

```
memcpy(inbuf,rawdata,224 * 224 * 3);
aml_util_flushTensorHandle(context,AML_INPUT_TENSOR);
```

2.1.11 aml_util_getInputTensorInfo

API	tensor_info* aml_util_getInputTensorInfo(const char* nbldata);
功能	获取 NBG 文件的输入信息。
参数	const char* nbldata: nbg 文件数据信息
返回值	tensor_info*: 返回 nbg 中 tensor 信息的指针

示例代码：

```
inptr = aml_util_getInputTensorInfo(config.pdata);
if(inptr->valid == 1)
{
    printf("the input buffer info is get success\n");
    printf("the input tensor number is %d\n",inptr->num);
    printf("the input[0] data_format:%d\n",inptr->info[0].data_format);
    printf("the input[0] quantization_format:%d\n",inptr->info[0].quantization_format);
    printf("the input[0] sizes_of_dim:%d,%d,%d,%d\n",inptr->info[0].sizes_of_dim[0],
    inptr->info[0].sizes_of_dim[1],inptr->info[0].sizes_of_dim[2],inptr->info[0].sizes_o
f_dim[3]);
}
aml_util_freeTensorInfo(inptr);
```

2.1.12 aml_util_getOutputTensorInfo

API	tensor_info* aml_util_getOutputTensorInfo(const char* nbldata);
功能	获取 NBG 文件的输出信息。
参数	const char* nbldata: nbg 文件数据信息
返回值	tensor_info*: 返回 nbg 中 tensor 信息的指针

示例代码：

```
outptr = aml_util_getOutputTensorInfo(config.pdata);
```



```

if(outptr->valid == 1)
{
    printf("the output buffer info is get success\n");
    printf("the output tensor number is %d\n",outptr->num);
    printf("the output[0] data_format:%d\n",outptr->info[0].data_format);
    printf("the output[0] quantization_format:%d\n",outptr->info[0].quantization_format);
    printf("the output[0] sizes_of_dim:%d,%d,%d,%d\n",outptr->info[0].sizes_of_dim[0],
        outptr->info[0].sizes_of_dim[1],outptr->info[0].sizes_of_dim[2],outptr->info[0].sizes_of_dim[3]);
}
aml_util_freeTensorInfo(outptr);

```

2.1.13 aml_util_freeTensorInfo

API	void aml_util_freeTensorInfo(tensor_info* tinfo);
功能	释放分配的 inptr 或 outptr 的内存空间
参数	tensor_info*: aml_util_getInputTensorInfo 或 aml_util_getOutputTensorInfo 获取的 inptr 或 outptr 数据
返回值	NULL

示例代码：

```

inptr = aml_util_getInputTensorInfo(config.pdata);
aml_util_freeTensorInfo(inptr);

```

2.1.14 aml_util_setProfile

API	int aml_util_setProfile(aml_profile_type_t type,const char *savepath);
功能	设置 Profile，获取性能、带宽、内存信息。
参数	aml_profile_type_t:获取 profile 信息类型。 const char *:profile log 保存路径
返回值	NULL

```

typedef enum {
    AML_PROFILE_NONE           = 0,
    AML_PROFILE_PERFORMANCE    = 1,
    AML_PROFILE_BANDWIDTH      = 2,
    AML_PROFILE_MEMORY         = 3
} aml_profile_type_t;

```

示例代码：

```
aml_util_setProfile(AML_PROFILE_MEMORY, "/media/test/sdk_save.log")
aml_util_setProfile(AML_PROFILE_BANDWIDTH, "/media/test/bandwidth_save.log")
aml_util_setProfile(AML_PROFILE_PERFORMANCE, NULL)
aml_util_setProfile(AML_PROFILE_NONE, NULL)
context = aml_module_create(&config);
```

2.1.15 aml_util_setPowerPolicy

API	int aml_util_setPowerPolicy(aml_policy_type_t type);
功能	设置电源策略
参数	aml_policy_type_t: 电源策略类型，分为 AML_PERFORMANCE_MODE, AML_POWER_SAVE_MODE, AML_MINIMUM_POWER_MODE
返回值	NULL

typedef enum {

AML_PERFORMANCE_MODE = 1,

AML_POWER_SAVE_MODE = 2,

AML_MINIMUM_POWER_MODE = 3

} aml_policy_type_t;

示例代码：

```
aml_util_setPowerPolicy(AML_MINIMUM_POWER_MODE);
context = aml_module_create(&config);
```

2.1.16 aml_util_getHardwareStatus

API	int aml_util_getHardwareStatus(int *customID, int *powerStatus);
功能	获取板端 customID 信息和电源状态信息
参数	int *customID: 芯片信息指针。
	int *powerStatus: 电源状态信息指针。
返回值	NULL

示例代码：

```
int customID, powerStatus;
```

```
aml_util_getHardwareStatus(&customID,&powerStatus);
printf("customID=%x,powerStatus=%x\n",customID,powerStatus);
context = init_network(argc,argv);
```

2.1.17 aml_util_switchInputBuffer

API	int aml_util_switchInputBuffer(void *context,void *newBuffer,unsigned int inputId);
功能	修改输入物理地址对应 DMA buffer 的起始指针，实现 input 的快速切换
参数	void *context:句柄 context 指针。
	void *newBuffer:输入的 buffer 信息。
	unsigned int inputId: 输入 ID，从 0 开始。
返回值	int: 成功返回零，失败返回非零。

示例代码：

```
config.inOut.inAddr[0] = input_phy_addr;
config.inOut.outAddr[0] = output_phy_addr;
config.inOut.io_type = AML_IO_PHYS;
context = aml_module_create(&config);

aml_util_switchInputBuffer(qcontext,(void*)new_in_addr,0);
aml_util_switchOutputBuffer(qcontext,(void*)new_out_addr,0);
aml_module_output_get(context,outconfig);
```

2.1.18 aml_util_switchOutputBuffer

API	int aml_util_switchOutputBuffer(void *context,void *newBuffer,unsigned int outputId);
功能	修改输出物理地址对应 DMA buffer 的起始指针，实现 output 的快速切换
参数	void *context:句柄 context 指针。
	void *newBuffer:输出的 buffer 信息。
	unsigned int outputId: 输出 ID，从 0 开始。
返回值	int: 成功返回零，失败返回非零。

示例代码：

```

config.inOut.inAddr[0] = input_phy_addr;
config.inOut.outAddr[0] = output_phy_addr;
config.inOut.io_type = AML_IO_PHYS;
context = aml_module_create(&config);

aml_util_switchInputBuffer(qcontext,(void*)new_in_addr,0);
aml_util_switchOutputBuffer(qcontext,(void*)new_out_addr,0);
aml_module_output_get(context,outconfig);

```

2.2 Amlogic SDK 数据结构详细说明

2.2.1 aml_config

成员变量	数据类型	含义
typeSize	int	结构体成员变量数
path	const char *	load nbg from file 指针
pdata	const char *	load nbg from memory 指针
length	int	nbg size
modelType	amlInn_model_type	nbg 模型类型
nbgType	amlInn_nbg_type	load nbg 方式
inOut	assign_user_address_t	分配用户地址

```

typedef struct __aml_nn_config
{
    int typeSize;
    const char *path;
    const char *pdata;
    int length;
    amlInn_model_type modelType;
    amlInn_nbg_type nbgType;
    assign_user_address_t inOut;
}aml_config;

```

```

typedef enum _amlInn_nbg_type_ {

```

```

NN_NBG_FILE    = 0, //load nbg from file
NN_NBG_MEMORY  //load nbg from memory
} amlnn_nbg_type;

```

amlnn_model_type

成员变量	模型类型
CAFFE	Caffe 模型
TENSORFLOW	Tensorflow 模型
TENSORFLOWLITE	TensorflowLite 模型
DARKNET	Darknet 模型
ONNX	Onnx 模型
KERAS	Keras 模型
PYTORCH	Pytorch 模型

2.2.2 assign_user_address_t

成员变量	数据类型	含义
io_type	aml_io_format_t	设置 dma 类型
inAddr	unsigned char*	输入地址
outAddr	unsigned char*	输出地址

```

typedef struct __assign_address
{
    unsigned char* inAddr[ADDRESS_MAX_NUM];
    unsigned char* outAddr[ADDRESS_MAX_NUM];
}assign_user_address_t;

typedef enum {
    AML_IO_VIRTUAL    = 0,
    AML_IO_PHYS       = 1,
} aml_io_format_t;

```

2.2.3 aml_output_config_t

成员变量	数据类型	含义
typeSize	int	结构体成员变量数
mdType	aml_module_t	网络模型
perfMode	aml_perf_mode_t	前向推理测试模式
format	aml_output_format_t	输出数据类型

```
typedef struct __aml_nn_module_out_data_t
{
    int typeSize;
    aml_module_t mdType;
    aml_perf_mode_t perfMode;
    aml_output_format_t format;
} aml_output_config_t;
```

2.2.4 aml_perf_mode_t

成员变量	含义
AML_NO_PERF	默认不开启测试模式
AML_PERF_INFERENCE	测试模式下只进行前向推理
AML_PERF_OUTPUT	测试模式下只进行反量化

```
typedef enum {
    AML_NO_PERF          = 0,
    AML_PERF_INFERENCE   = 1,
    AML_PERF_OUTPUT      = 2
} aml_perf_mode_t;
```

2.2.5 aml_output_format_t

成员变量	含义
AML_OUTDATA_FLOAT32	输出 float32 类型的数据
AML_OUTDATA_RAW	输出原始数据
AML_OUTDATA_DMA	输出 dma 数据

```
typedef enum {
    AML_OUTDATA_FLOAT32    = 0,
    AML_OUTDATA_RAW        = 1,
    AML_OUTDATA_DMA        = 2
} aml_output_format_t;
```

2.2.6 image_out_t

成员变量	数据类型	含义
height	int	输出图像高度值
width	int	输出图像宽度值
channel	int	输出图像通道数
data	unsigned char *	输出图像 buffer 信息

```
typedef struct __nn_image_out
{
    int height;
    int width;
    int channel;
    unsigned char *data; //this buffer is returned by aml_module_output_get
}image_out_t;
```

2.2.7 tensor_info

成员变量	数据类型	含义
------	------	----

valid	unsigned int	获取 tensor 信息 flag
num	unsigned int	tensor 数目
info	info *	tensor 信息指针

```
typedef struct {
    unsigned int valid;
    unsigned int num;
    info *info;
} tensor_info;
```

2.2.8 info

成员变量	数据类型	含义
dim_count	unsigned int	维度数目
sizes_of_dim	unsigned int[]	维度值
data_format	unsigned int	buffer format 类型
data_type	unsigned int	数据类型
quantization_format	unsigned int	量化类型
fixed_point_pos	int	int8/int16 量化的参数值
TF_scale	float	scale 值
TF_zeropoint	int	zeropoint 值
name	char[]	名称

```
typedef struct {
    unsigned int dim_count;          /*dim count*/
    unsigned int sizes_of_dim[4]; /*dim value,just support 4-d dim*/
    unsigned int data_format;       /*see as nn_buffer_format_e*/
    unsigned int data_type;         /*not use*/
    unsigned int quantization_format; /*see as nn_buffer_quantize_format_e*/
    int fixed_point_pos;            /*for int8/int16 UANTIZE_DYNAMIC_FIXED_POINT*/
}
```



```

float TF_scale;           /*as tf define,scale*/
int TF_zeropoint;        /*as tf define,zeropoint*/
char name[MAX_NAME_LENGTH]; /*not use,will used in future*/
} info

```

2.2.9 detBox

成员变量	数据类型	含义
x	float	输出目标框的 X 轴坐标值
y	float	输出目标框的 Y 轴坐标值
w	float	输出目标框的宽度信息
h	float	输出目标框的高度信息
score	float	输出目标的得分信息
objectClass	float	输出目标的类别信息

2.2.10 point_t

成员变量	数据类型	含义
x	float	输出目标点的 X 轴坐标值
y	float	输出目标点的 Y 轴坐标值

2.2.11 nn_output

成员变量	数据类型	含义
typeSize	int	结构体成员变量数
num	unsigned int	输出 tensor 的个数
out	outBuf_t[]	输出 tensor 结构体数组

```

typedef struct __nnout
{

```

```

int typeSize;
unsigned int num; /*=====output tensor number=====*/
outBuf_t out[OUTPUT_MAX_NUM];
}nn_output;

```

2.2.12 outBuf_t

成员变量	数据类型	含义
size	unsigned int	输出 tensor 的个数
buf	unsigned char *	输出 tensor 的数据地址
param	nn_buffer_params_t *	输出 tensor 的属性
name	char	输出 tensor name

```

typedef struct out_buf
{
    unsigned int size;
    unsigned char *buf;
    nn_buffer_params_t *param;
    char name[MAX_NAME_LENGTH]; //output tensor name
    aml_output_format_t out_format;
}outBuf_t;

```

2.2.13 nn_buffer_params_t

成员变量	数据类型	含义
num_of_dims	unsigned int	Tensor 维度个数
sizes	unsigned int[]	Tensor 各维度值
data_format	nn_buffer_format_e	tensor 的数据类型，有以下几种： NN_BUFFER_FORMAT_FP32 = 0, NN_BUFFER_FORMAT_FP16 NN_BUFFER_FORMAT_UINT8 NN_BUFFER_FORMAT_INT8

		NN_BUFFER_FORMAT_UINT16 NN_BUFFER_FORMAT_INT16
quant_format	nn_buffer_quantize_format_e	量化格式，有以下几种值： NN_BUFFER_QUANTIZE_NONE=0, NN_BUFFER_QUANTIZE_DYNAMIC_FIXED_POINT（动态定点量化） NN_BUFFER_QUANTIZE_TF_ASYMM （非对称量化）
dfp	struct	DYNAMIC_FIXED_POINT 量化类型参数： fl = dfp.fixed_point_pos
affine	struct	TF_ASYMM 量化类型参数： scale=affine.scale zeroPoint=affine.zeroPoint

```

typedef struct nn_buffer_create_params_t
{
    unsigned int    num_of_dims;
    unsigned int    sizes[4];
    nn_buffer_format_e    data_format;
    nn_buffer_quantize_format_e    quant_format;
    union {
        struct {
            unsigned char fixed_point_pos;
        } dfp;
        struct {
            float    scale;
            unsigned int zeroPoint;
        } affine;
    }
    quant_data;
} nn_buffer_params_t;

```

2.2.14 nn_input

成员变量	数据类型	含义
typeSize	int	结构体成员变量数
input_index	int	输入数据对应输入 tensor 的下标，单个输入为 0
size	int	输入数据大小
input	unsigned char*	输入数据地址
input_type	amlnn_input_type	输入数据类型，有以下几种类型： RGB24_RAW_DATA = 0, TENSOR_RAW_DATA, QTENSOR_RAW_DATA, BINARY_RAW_DATA, INPUT_DMA_DATA
info	input_info	输入 buffer 信息

```
typedef struct {
    int valid;
    float mean[INPUT_CNANNEL];
    float scale;
    aml_input_format_t input_format;
    int int16_type; //when int16 quantize model mean value is not 128,128,128,set
1
    int preprocess_debug; // save npu input data
}input_info;
typedef enum {
    AML_INPUT_DEFAULT    = 0,    //channle format: caffe 2 1 0 ,others 0 1 2
    AML_INPUT_MODEL_1    = 1,    //channle format: 0 1 2
    AML_INPUT_MODEL_2    = 2,    //channle format: 2 1 0
} aml_input_format_t;
```

3. SDK API 使用方法介绍

目前 SDK 包使用特定网络 ID: CUSTOM_NETWORK 来代表新的网络, 这样输出的数据就会保留原始网络的输出, 不会针对该类型做后处理操作。在多网络应用时, 客户根据网络 aml_module_create 返回的 context 值来区分管理多个网络实例。

各功能使用方式示例 demo 请见 GitHub: https://github.com/Amlogic-NN/AML_NN_SDK/tree/master/Linux/Demo

3.1 SDK 自定义模型调用流程

1、设置 input/output dma (选)

```
unsigned char * inbuf = aml_util_mallocAlignedBuffer(224*224*3);
unsigned char * outbuf = aml_util_mallocAlignedBuffer(1000*2);
memcpy(inbuf,rawdata,224*224*3);
config.inOut.outAddr[0] = outbuf;
config.inOut.inAddr[0] = inbuf;
```

2、获取输入输出 tensor 信息 (选)

```
inptr = aml_util_getInputTensorInfo(config.pdata);
outptr = aml_util_getOutputTensorInfo(config.pdata);
aml_util_freeTensorInfo(inptr);
aml_util_freeTensorInfo(outptr);
```

3、初始化模型: void* context = aml_module_create(&config);

4、设置 Input buffer swap (选)

```
unsigned char * inbuf2 = aml_util_mallocAlignedBuffer(224*224*3);
memcpy(inbuf2,rawdata,224*224*3);
aml_util_swapInputBuffer(context,(void*)inbuf2,0);
```

5、flush cache (选)

```
aml_util_flushTensorHandle(context,AML_INPUT_TENSOR);//配合 input dma 使用
```

6、准备输入数据, 并设置输入数据:

```
inData.input_index = 0;
inData.size = 224*224*3;
inData.input = rawdata;
inData.input_type = INPUT_DMA_DATA;
int ret = aml_module_input_set(context,&inData);
```

7、进行前向推理及获取推理结果:

```
outdata = (nn_output *)aml_module_output_get_simple(context);
```

8、多次推理：多次重复执行 6、7 两步。

9、卸载模型：int ret = aml_module_destroy(context);

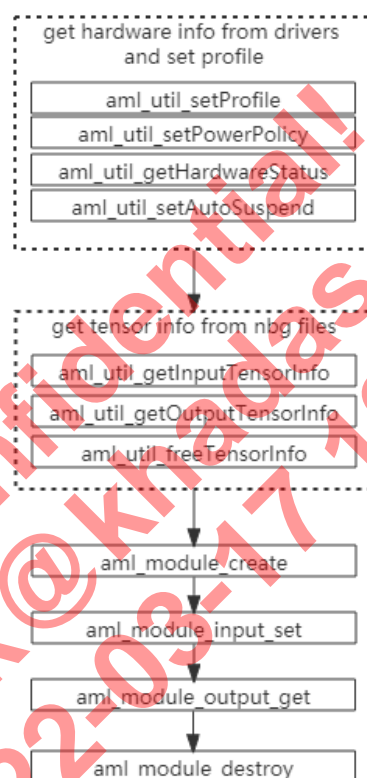
10、释放 input/output 内存空间（选）

```
aml_util_freeAlignedBuffer(outbuf);
```

```
aml_util_freeAlignedBuffer(inbuf);
```

3.2 SDK 基本调用流程说明

图 3.1 SDK API 调用流程图



3.2.1 模型初始化

```
aml_config config;
```

//有两种模型 nbg load 方式

方式一：load nbg from memory

```
fp = fopen(argv[1], "rb");
fseek(fp, 0, SEEK_END);
size = (int)ftell(fp);
rewind(fp);
config.pdata = (char *)calloc(1, size);
```

```
fread((void*)config.pdata,1,size,fp);

config.nbgType = NN_NBG_MEMORY;

config.length = size;

fclose(fp);
```

方式二：load nbg from file

```
config.path= (const char *)argv[1];

config.nbgType = NN_NBG_FILE;
```

```
config.modelType = TENSORFLOW;           //根据实际使用框架确定

void *context = aml_module_create(&config);
```

注：

1、SDK 提供已集成模型 Nbg 文件以供用户使用，Nbg 需要用户 push 进平台的指定路径下。https://github.com/Amlogic-NN/AML_NN_SDK

目前 nbg 文件区分平台，可使用以下方式确定需要的 nbg 文件。

执行命令：cat /proc/cpuinfo

查看倒数第二行数 Serial 对应的数列码前四个字符，根据如下对应关系来确定 nbg 文件。

PID	序列号
0X7D	Serial : 290a70004ba55adb38423231474c4d4
0X88	Serial : 290b70004ba55adb38423231474c4d4
0X99	Serial : 2b0a0f00df472d383156314d534c4d4
0XA1	Serial : 300a010245bbf1e131305631434c4d4

例如：序列号前四位为 290b，则需要将 **xxx_88.nb** push 进平台的指定路径下。

2、modelType: 模型框架类型。modelType 指定对应的框架类型，不同的框架类型对应不同的前处理方式。

其中，caffe、pytorch、darknet、onnx 默认为 nchw 格式，tensorflow、tensorflowlite、keras 默认为 nhwc 格式，且格式转换过程已集成在 libnnsdk.so 中，用户不需特意修改格式。其中 caffe 的 reorder channel 默认设置为'2,1,0'，其余框架均默认设置为'0,1,2'。如果 pytorch 模型的 channel 格式为'2,1,0'，则需要将框架类型设置为 caffe。

Caffe:	BBBGGGRRR
Pytorch、darknet、onnx:	RRRGGBBB
tensorflow、tensorflowlite、keras:	RGBRGRGB

如果用户的模型框架类型与 reorder channel 的组合方式不在上述的范围内，则可通过选择 info.valid 模式设置 channel 格式。

```
inData.info.valid = 2;

inData.info.input_format = AML_INPUT_MODEL_1;    // reorder channel '0,1,2'

inData.info.input_format = AML_INPUT_MODEL_2;    // reorder channel '2,1,0'
```

3、file 方式输入参数为 nbg 文件的路径。其中，config.path 是指向 nbg 具体信息的指针，当指定 config.nbgType 为 NN_NBG_FILE 时，其赋值即为 nbg 文件的路径。

4、当 nbg 内容已保存在内存中时，可以选择从 memory 加载。其中，config.pdata 是指向 nbg 具体信息的指针，当指定 config.nbgType 为 NN_NBG_MEMORY 时，其赋值代表 nbg 的 memory 首地址。与 NN_NBG_FILE 不同，该方式需额外设置 config.size 为 nbg 的 memory 的实际长度。

5、使用 memory 加载 nbg 的方式需要进行内存释放，free((void*)config.pdata)。

3.2.2 设置输入

```
nn_input inData;

unsigned char *rawdata = get_jpeg_rawData(jpeg_path,416,416); //设置输入尺寸

inData.input_index = 0; //输入数据索引值，支持数据多输入

inData.size = 416*416*3; //设置输入尺寸，W*H*C

inData.input = rawdata; //设置输入 buffer

inData.input_type = RGB24_RAW_DATA; //设置输入数据格式
```

注：

- 1、get_jpeg_rawData：读取 jpeg_path 图像，当前函数只支持与指定尺寸一致的图像输入。用户可自定义相关接口读取输入图像信息。
- 2、inData.size：根据实际模型设置输入尺寸，W*H*C。
- 3、input_index：输入数据索引值。单个输入设置 0 即可，多个输入的情况，需多次调用该接口，索引值依次加 1。
- 4、inData.input_type：输入数据类型，有以下几种类型：

输入数据类型	含义	处理操作
RGB24_RAW_DATA	输入 RGB 格式数据，(0,255)	进行 reorder、mean、scale 等操作
TENSOR_RAW_DATA	原始 tensor 数据 float 输入，处理方式为 float32	<code>memcpy(&fval.rawdata,sizeof(float));</code> <code>convertFloatToDtype(fval.tensordata);</code>
QTENSOR_RAW_DATA	原始 tensor 数据 float 输入，处理方式为 Uint8	<code>memcpy(&fval.rawdata,sizeof(float));</code> <code>Tensordata = (uint8)fval;</code>
BINARY_RAW_DATA	原始数据输入，中间不做任何处理与转换	<code>memcpy(tensordata_all.rawdata_all,all_size);</code>

此外支持 INPUT_DMA_DATA 类型，如使用 input dma 功能，则 inData.input_type 需要设置为该类型。

Input_type 类型说明：

- RGB24_RAW_DATA：输入 RGB 格式数据。数据输入后，系统会做 reorder、mean、scale 等相关操作。这些参数的值来源于 acuity 生成 nbg 时设置的信息。

- TENSOR_RAW_DATA：原始 tensor 数据 float 输入。系统拿到该数据 buffer 后，会做以下类似的操作：

```
float fval;
```



```
memcpy(&fval,rawdata,sizeof(float));
convertFloatToDtype(fval,tensordata);
```

以上完成了一个 float 数据的处理。其中 rawdata 为输入原始数据，tensordata 为最终输入到 model 的数据。

●QTENSOR_RAW_DATA: 与 TENSOR_RAW_DATA 类似，输入数据为 float 类型，处理方式 (uint8)：

```
float fval;
memcpy(&fval,rawdata,sizeof(float));
tensordata = (uint8)fval;
```

以上完成了一个 float 数据的处理。其中 rawdata 为输入原始数据，tensordata 为最终输入到 model 的数据。

●BINARY_RAW_DATA:原始数据输入，中间不做任何处理与转换。流程近似为：

```
memcpy(tensordata_all,rawdata_all,all_size);
```

将输入数据原封不动的全部 copy 到 model 的 input buffer 中。

●INPUT_DMA_DATA: 使用 DMA 功能获取的输入。

```
data = _get_rgb_data(tensor, pmeta, input, format_type);
memcpy(input_dma,data,sz*stride);
```

3.2.3 获取模型结果

使用 nn_output 的输出结构体来接收 aml_module_output_get 接口返回的结果。

```
nn_output *outdata = NULL;
outconfig.typeSize = sizeof(aml_output_config_t);
modelType = CUSTOM_NETWORK; //设置输出模型类型
outconfig.format = AML_OUTDATA_FLOAT32; //设置输出数据类型
outdata = (nn_output*)aml_module_output_get(context,outconfig);
if(outdata->out[0].param->data_format == NN_BUFFER_FORMAT_FP32)
{
    process_top5((float*)outdata->out[0].buf,outdata->out[0].size/sizeof(float),NULL);
}
```

outdata 中包含了完整的输出信息，包括输出的 dim，输出的格式，输出的内容等等，输出内容默认为 float32 格式，后面可以进行相关网络的后处理实现，如 mobilenet：

```
process_top5((float*)pout->out[0].buf,pout->out[0].size/sizeof(float));
```

注：

1、outconfig.format 可根据实际需要选择 AML_OUTDATA_FLOAT32、AML_OUTDATA_RAW、AML_OUTDATA_DMA。

2、其中 AML_OUTDATA_FLOAT32 表示输出的数据类型为 f32，类型转换已在 sdk 内部完成；AML_OUTDATA_RAW 表示输出原始类型的数据；AML_OUTDATA_DMA 表示输出 dma output buffer 数据类型，需要配合 DMA 使用。

3、为了便于用户使用自定义模型，提供一个参数简单的接口进行前向推理及获取推理结果。

```
void* aml_module_output_get_simple(void* context);
```

此接口只需传入 context，默认输出数据类型为 float32。其使用方式为：

```
nn_output * outdata = (nn_output *)aml_module_output_get_simple(context);
```

用户可根据需要灵活选择推理接口。

4、若用户需要获取模型前向推理时间，可通过 outconfig.perfMode 开启性能测试模式分别获取前向推理和反量化的耗时。outconfig.perfMode 默认不设置，测试模式默认关闭。

例如：

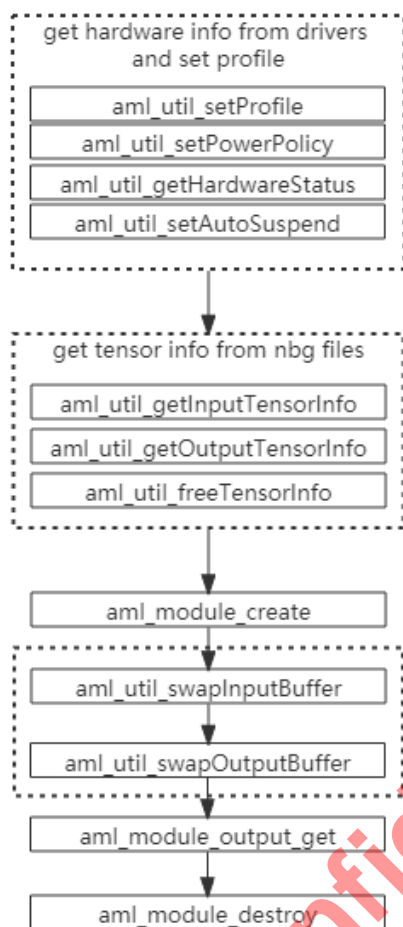
```
outconfig.typeSize = sizeof(aml_output_config_t);
modelType = CUSTOM_NETWORK;
outconfig.format = AML_OUTDATA_FLOAT32;
tmsStart = get_perf_count();
outconfig.perfMode = AML_PERF_INFERENCE;
for (i=0;i<100;i++)
    outdata = (nn_output*)aml_module_output_get(qcontext,outconfig);
tmsEnd = get_perf_count();
msVal = (tmsEnd - tmsStart)/1000000;
usVal = (tmsEnd - tmsStart)/1000;
printf("AML_PERF_INFERENCE: %ldms or %ldus\n", msVal, usVal);

tmsStart = get_perf_count();
outconfig.perfMode = AML_PERF_OUTPUT;
outdata = (nn_output*)aml_module_output_get(qcontext,outconfig);
tmsEnd = get_perf_count();
msVal = (tmsEnd - tmsStart)/1000000;
usVal = (tmsEnd - tmsStart)/1000;
printf("AML_PERF_OUTPUT: %ldms or %ldus\n", msVal, usVal);
```

3.3 SDK 拓展功能介绍

3.3.1 SDK - DMA 功能介绍

图 3.2 SDK API DMA 功能调用流程图



1、input dma 设置

1.1 Virtual Address

```

inbuf = aml_util_mallocAlignedBuffer(224*224*3);    //malloc buffer
config.inOut.inAddr[0] = inbuf;                    //inAddr 信息设置
context = aml_module_create(&config);

```

方式一：

```

memcpy(inbuf,rawdata,224*224*3);                    //rawdata 为读取的输入数据

```

方式二：

```

inData.input_index = 0;
inData.size = 224*224*3;
inData.input = rawdata;
inData.input_type = INPUT_DMA_DATA;                  //设置输入数据类型
aml_module_input_set(context,&inData);

```

注：

1、`aml_util_mallocAlignedBuffer` 分配内存需对应输入图像的尺寸。

2、mallocAlignedBuffer 和 config.inOut.inAddr 设置**必须**在 aml_module_create 之前完成，否则 DMA 功无法正常使用。

3、方式一在设置 input 指针后，可以直接将 input 数据 copy 到 inputbuf 中，不用再做 aml_module_input_set 操作。此方式**必须**保证输入的数据满足预处理要求。

4、如果不清楚需要做的预处理操作可使用方式二，设置 inData.input_type = INPUT_DMA_DATA，系统会做对应的预处理后直接 copy 到 inputbuf，没有 npu 与 cpu 的内存交互。

1.2 Physical Address

```
config.inOut.inAddr[0] = (unsigned char*)0x10000000;//set physical address
```

```
config.inOut.io_type = AML_IO_PHYS;
```

```
context = aml_module_create(&config);
```

2、input dma swap 设置

```
inbuf1 = aml_util_mallocAlignedBuffer(224*224*3);
```

```
inbuf2 = aml_util_mallocAlignedBuffer(224*224*3);
```

```
config.inOut.inAddr[0] = inbuf1; //设置 inAddr
```

```
context = aml_module_create(&config);
```

```
memcpy(inbuf1,rawdata1,224*224*3);
```

```
memcpy(inbuf2,rawdata2,224*224*3);
```

```
aml_util_swapInputBuffer(context,(void*)inbuf2,0);
```

注：

1、aml_util_swapInputBuffer 第三个参数为 inputId，从 0 开始。

2、该操作可以在第一帧执行过程中准备好，形成类似 ping-pang buffer 的效果，以省去 input 输入的时间。

3、output dma 设置

```
outbuf = aml_util_mallocAlignedBuffer(1000*2);
```

```
config.inOut.outAddr[0] = outbuf;
```

```
context = aml_module_create(&config);
```

注：

1、aml_util_mallocAlignedBuffer 需要设置输出 buffer size，如如分类模型的分类结果是 1000 个种类，每个种类是 float16 类型，故 outbuf 需要的长度为 1000*2。

2、aml_util_mallocAlignedBuffer 和 config.inOut.outAddr **必须**在 aml_module_create 之前完成，否则 dma 功能无法正常使用。

3、若使用 input/output dma 功能则需要进行 flush cache，否则可能出现运行结果错误的情况。在输入较小时可选择不进行该操作，输入尺寸超过 224*224*3 时必须进行 flush 操作。

flush cache

```
memcpy(inbuf,rawdata,224*224*3)
```

```
aml_util_flushTensorHandle(context,AML_INPUT_TENSOR)
```

3.3.2 SDK - GetTensorInfo 功能介绍

```

inptr = aml_util_getInputTensorInfo(config.pdata);
outptr = aml_util_getOutputTensorInfo(config.pdata);
if(inptr->valid == 1)
{
    printf("the input buffer info is get success\n");
    printf("the input tensor number is %d\n",inptr->num);
    printf("the input[0] data_format:%d\n",inptr->info[0].data_format);
    printf("the input[0] quantization_format:%d\n",inptr->info[0].quantization_format);
    printf("the input[0] sizes_of_dim:%d,%d,%d,%d\n",inptr->info[0].sizes_of_dim[0],
inptr->info[0].sizes_of_dim[1],inptr->info[0].sizes_of_dim[2],inptr->info[0].sizes_of_dim[3]);
}

if(outptr->valid == 1){
    for (i=0;i<outptr->num;i++) {
        printf("the outptr tensor name is %s\n",outptr->info[i].name);
        printf("the outptr buffer info is get success\n");
        printf("the outptr tensor number is %d\n",outptr->num);
        printf("the outptr[%d] data_format:%d\n",i,outptr->info[i].data_format);
        printf("the outptr[%d] quantization_format:%d\n",i,outptr->info[i].quantization_format);
        printf("the outptr[%d] sizes_of_dim:%d,%d,%d,%d\n",i,
outptr->info[i].sizes_of_dim[0],outptr->info[i].sizes_of_dim[1],outptr->info[i].sizes_of_dim[2],outptr->info[i].sizes_of_dim[3]);
    }
}

aml_util_freeTensorInfo(inptr);          //释放 inptr 或 outptr
aml_util_freeTensorInfo(outptr);

```

注:

- 1、config.pdata: nbg 文件的数据信息。
- 2、获取 tensor 信息的功能需要配合使用内存读取 nbg 方式才可正常使用。
- 3、获取的信息包括: dim_count , sizes_of_dim[4] , data_format, data_type, quantization_format, fixed_point_pos , TF_scale , TF_zerpoint, name

3.3.3 SDK - mean 值不一致设置方式介绍

mean 值设置

```
inData.info.mean[0] = 123;    //the input mean will set as this;
inData.info.mean[1] = 116;
inData.info.mean[2] = 103;
```

注:

1、当前 uint8 类型的 mean 默认三通道一致，若用户的模型三通道 mean 值不一致，则可在 aml_module_input_set 之前设置 mean 值。

2、当前 int8/int16 类型的 mean 值默认设置为 128，若用户模型 mean 不为 128，则可在 aml_module_input_set 之前设置 mean 值。

示例代码:

```
context = aml_module_create(&config);
inData.typeSize = sizeof(inData);
inData.input_index = 0;
inData.size = 320*320*3;
inData.input_type = RGB24_RAW_DATA;
inData.info.valid = 1;
inData.info.mean[0] = 123;
inData.info.mean[1] = 116;
inData.info.mean[2] = 103;
aml_module_input_set(context,&inData);
```

3.3.4 SDK - usb camera 功能介绍

usb camera 通过调用 camera_thread_func 和 net_thread_func 两个线程实现实时显示在 framebuffer 上。

```
context = init_network(argc,argv);
pthread_mutex_init(&mutex_data,NULL);
init_fb();
if (0 != pthread_create(&tid[0],NULL,camera_thread_func,NULL))
{
    fprintf(stderr, "Couldn't create thread func\n");
    return -1;
}
thread_args = (void*)argv[3];
if (0 != pthread_create(&tid[1],NULL,net_thread_func,thread_args))
```

```

{
    fprintf(stderr, "Couldn't create thread func\n");
    return -1;
}
while(1)
{
    for (i=0;i<2;i++)
        pthread_join(tid[i], NULL);
}

```

其中，net_thread_func 中 camera 的调用方式可参考以下代码，详细代码请见 release 包中的 face_detect 模型实现。

```

dup_rgbbuf = (unsigned char*)malloc(display_width * display_high * 3);
input_data = (unsigned char*)malloc(input_width * input_high * 3);
while(1)
{
    if(rgbbuf != NULL)
    {
        pthread_mutex_lock(&mutex_data);
        memcpy(dup_rgbbuf,rgbbuf,display_width*display_high*3);
        pthread_mutex_unlock(&mutex_data);
    }
    resize_input_data(dup_rgbbuf,input_data);
    ret = run_network(netType,context,input_data,AML_IN_CAMERA,dup_rgbbuf);
}

```

注：

1、当前分辨率默认设置为 640*480，支持 1920*1080、1280*720、640*480 三种分辨率。

3.3.5 SDK - profile 控制方式介绍

```
aml_util_setProfile(AML_PROFILE_MEMORY,"save_log_path");
```

其中 setProfile 可以设置四种模式：

- AML_PROFILE_NONE 表示不输出 Profile 信息；
- AML_PROFILE_PERFORMANCE 表示输出模型运行时间，即性能指标；
- AML_PROFILE_BANDWIDTH 表示输出模型运行消耗的带宽信息；
- AML_PROFILE_MEMORY 表示输出模型运行占用的内存信息。

aml_util_setProfile 操作需要在 init_network 之前完成。通过设置不同的模式可输出对应的信息并保存至指定路径，其中 AML_PROFILE_PERFORMANCE 当前不可保存 log。

注:

- 1、save_log_path 必须设置为板端有权限的路径
- 2、若同时设置不同模式，需要在 setProfile 之后再进行一次刷新操作，即 `aml_util_setProfile(AML_PROFILE_NONE,NULL);`，然后再设置另一种模式。
- 3、如需获取前向推理时间，通过设置环境变量设置循环次数；
例如： `export NN_LOOP_TIME=100`

3.3.6 SDK - PowerPolicy 控制方式介绍

```
aml_util_setPowerPolicy(AML_MINIMUM_POWER_MODE);
```

其中，setPowerPolicy 可以设置三种模式：

- AML_PERFORMANCE_MODE 表示全算力运行模型；
- AML_POWER_SAVE_MODE 表示半算力运行模型；
- AML_MINIMUM_POWER_MODE 表示四分之一算力运行模型。

setPowerPolicy 操作需要在 init_network 之前完成。

注:

- 1、三种模型的运行时间大致呈线性的关系，即全算力的二倍、四倍的关系。
- 2、若同时设置不同模式，需要在 setProfile 之后再进行一次刷新操作，即 `aml_util_setProfile(AML_PROFILE_NONE,NULL);`，然后再设置另一种模式。

3.3.7 SDK - HardwareStatus 信息获取方式介绍

```
int aml_util_getHardwareStatus(int* customID,int *powerStatus)
```

其中 customID 是芯片 PID 信息，分为 7d,88,99,a1,b9,be 六种平台。

powerStatus 为电源当前的状态，为以下四种状态 POWER_IDLE，POWER_ON，POWER_OFF，POWER_RESET。

通过此接口可以获取芯片信息和电源状态方便用户 debug。

3.3.8 ffmpeg 解析视频接口介绍

当前 SDK 支持输入视频数据，利用 ffmpeg 解码视频用于网络的输入。将视频解析并经过 resize 得到的图像作为 rawdata 输入数据送入模型。

```
sprintf(cmd,"ffmpeg -i %s -qscale:v 2 -r 24 ",jpath);  
ptr=strcat(cmd,"tmp/image%5d.bmp");  
system(ptr);  
sprintf(img_name,"tmp/image%05d.bmp",index);  
while (1)
```



```

{
    sprintf(img_name,"tmp/image%05d.bmp",index);
    if((access(img_name,F_OK)) < 0)
    {
        break;
    }
    img=imread(img_name,199);
    resize(img,dst,dst.size());

    ret = run_network(netType,context,dst.data,AML_IN_VIDEO,(unsigned
char*)img_name);
    index++;
}

system("ffmpeg -f image2 -i tmp/image%5d.bmp -b:v 5626k videoout.mp4");

```

3.3.9 SDK - 保存 NPU 输入信息

当前 `inData.input_type` 支持 `RGB24_RAW_DATA`、`TENSOR_RAW_DATA`、`QTENSOR_RAW_DATA`、`BINARY_RAW_DATA`、`INPUT_DMA_DATA` 多种格式的输入数据，为了便于确认前处理流程正常且传递给 NPU 的输入数据是预期的。可设置数据前处理 debug flag，保存输入信息。

```
inData.info.preprocess_debug=1;
```

运行模型后会保存文件名为 “`input_data_[input_num]_[input size].txt`” 的文本文件，可与 PC 仿真结果进行对比。

3.4 参考代码

```

int main(int argc,char **argv)
{
    void *context;
    const char *jpeg_path = NULL;
    unsigned char *rawdata;
    aml_config config;
    nn_input inData;
    nn_output *outdata = NULL;
    int ret;
    int i;
    nn_query *query;
    jpeg_path = (const char *)argv[2];
    config.modelType = TENSORFLOW;
    /***** load nbg from memory *****/
    fp = fopen(argv[1],"rb");
    fseek(fp,0,SEEK_END);
    size = (int)ftell(fp);
    rewind(fp);

```

```

config.pdata = (char *)calloc(1,size);
fread((void*)config.pdata,1,size,fp);
config.nbgType = NN_NBG_MEMORY;
config.size = size;
fclose(fp);
/***** load nbg from file *****/
config.path= (const char *)argv[1];
config.nbgType = NN_NBG_FILE;

context = aml_module_create(&config);
rawdata = get_jpeg_rawData(jpeg_path,224,224);
inData.input_index = 0; //this value is index of input,begin from 0
inData.size = 224*224*3;
inData.input = rawdata;
inData.input_type = BINARY_RAW_DATA;
ret = aml_module_input_set(context,&inData);
if(rawdata != NULL)
{
    free(rawdata);
    rawdata = NULL;
}
outdata = (nn_output *)aml_module_output_get_simple(context);
if(outdata->out[0].param->data_format == NN_BUFFER_FORMAT_FP32)
{
    process_top5((float*)outdata->out[0].buf,outdata->out[0].size/sizeof(float),NULL)
;}
ret = aml_module_destroy(context);
return ret;
}

```

outdata 中包含了完整的输出信息，包括输出的 dim，输出的格式，输出的内容等等，输出内容默认为 float32 格式，后面可以进行相关网络的后处理实现，如 mobilenet:

```

process_top5((float*)pout->out[0].buf,pout->out[0].size/sizeof(float));
void process_top5(float *buf,unsigned int num,img_classify_out_t* clsout)
{
    int j = 0;
    unsigned int MaxClass[5]={0};
    float fMaxProb[5]={0.0};

    float *pfMaxProb = fMaxProb;
    unsigned int *pMaxClass = MaxClass,i = 0;

    for (j = 0; j < 5; j++)
    {
        for (i=0; i<num; i++)
        {
            if ((i == *(pMaxClass+0)) || (i == *(pMaxClass+1)) || (i == *(pMaxClass+2)) ||
                (i == *(pMaxClass+3)) || (i == *(pMaxClass+4)))
            {
                continue;
            }
            if (buf[i] > *(pfMaxProb+j))
            {
                *(pfMaxProb+j) = buf[i];
                *(pMaxClass+j) = i;
            }
        }
    }
}

```

```

    }
}
for (i=0; i<5; i++)
{
    if (clsout == NULL)
    {
        printf("%3d: %8.6f\n", MaxClass[i], fMaxProb[i]);
    }
    else
    {
        clsout->score[i] = fMaxProb[i];
        clsout->topClass[i] = MaxClass[i];
    }
}
}
}

```

3.5 模型性能 debug 接口

1. aml_util_setProfile(aml_profile_type_t type, const char *savepath);

获取模型运行指标

1.1. Savepath

指定板端的保存路径，DDK 将相关信息保存到指定的路径下

1.2 aml_profile_type_t type

profile 调试方式

A. AML_PROFILE_PERFORMANCE

获取模型 run_graph 运行时间

B. AML_PROFILE_BANDWIDTH

获取模型运行时的带宽信息并写入到文件中

C. AML_PROFILE_MEMORY

获取模型运行时的内存占用信息并写入到文件中

D. AML_PERLAYER_RUNTIME

获取模型每层的运行时间并写入到文件中

E. AML_PERLAYER_BANDWIDTH

获取模型每层的带宽信息并写入到文件中

F. AML_PERLAYER_OUTPUT

获取模型每层的运行结果并写入到文件中

注：D/E/F 功能后续版本加以完善，当前暂不支持

2. aml_util_setPowerPolicy(aml_policy_type_t type);

通过控制 NPU 频率设置电源策略

typedef enum {

```

    AML_PERFORMANCE_MODE           = 1,    // nanoqFreq
    AML_POWER_SAVE_MODE             = 2,    // nanoqFreq/2
    AML_MINIMUM_POWER_MODE          = 3     // nanoqFreq/4
} aml_policy_type_t;

```

3. aml_util_getHardwareStatus(int* customID, int *powerStatus, int* version);

获取硬件信息

- A. 获取平台 ID
- B. 获取当前电源状态
- C. 获取 DDK 版本信息

4. `aml_util_setAutoSuspend(int timeout);`

设置休眠时间，当 NPU 在指定的时间内没有任务时系统自动进入下电状态

Confidential!
nick@khadas.com
2022-03-17 19:26:37