共达地gddeploy推理SDK用户手册

目录

[1. 概述 2](#_Toc131684713)

[2.SDK使用说明 2](#_Toc131684714)

**[2.1 Runner说明](#_Toc131684715)** [3](#_Toc131684715)

**[2.2 API调用流程：](#_Toc131684716)** [5](#_Toc131684716)

**[2.2.1 Infer API](#_Toc131684717)** [5](#_Toc131684717)

**[2.2.2 Session API](#_Toc131684718)** [6](#_Toc131684718)

**[2.2.3 Processor API](#_Toc131684719)** [7](#_Toc131684719)

**[2.3 API详细说明](#_Toc131684720)** [8](#_Toc131684720)

**[2.4 数据结构定义：](#_Toc131684721)** [8](#_Toc131684721)

**[2.5 结果解析说明：](#_Toc131684722)** [9](#_Toc131684722)

**[2.6 内存管理说明](#_Toc131684723)** [9](#_Toc131684723)

[3.工具使用 11](#_Toc131684724)

**[3.1 获取硬件SN](#_Toc131684725)** [11](#_Toc131684725)

**[3.2 测试应用程序](#_Toc131684726)** [12](#_Toc131684726)

**[4.](#_Toc131684727)****[sample说明](#_Toc131684727)** [12](#_Toc131684727)

**[5.](#_Toc131684728)****[FAQ](#_Toc131684728)** [12](#_Toc131684728)

文档记录

|  |  |  |
| --- | --- | --- |
| 修改版本 | 修改内容 | 修改人 |
| V0.1 | 初始化文档 | 李桂友 |
|  |  |  |

# 概述

gddeploy是共达地面向模型部署推理场景自创的推理部署框架，具有简易、多接口、性能高效、对接多款推理芯片等优点，以满足不同客户需求。以下是框架的总体示意图：

目前已支持硬件平台和算法列表：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 硬件\算法 | 分类 | 检测 | 姿态 | 分割 | OCR | 动作 | 多模态 |
| Nvidia |  |  |  |  |  |  |  |
| 算能 | ✔ | ✔ | ✔ | ✔ |  |  |  |

SDK包包含目录和说明：

thirdparty

opencv

ffmpeg

bin //测试可执行程序

dockerfile //搭建环境使用的docker和安装脚本

docs

lib

include

sample

tools

gtx\_maker//生成SN码工具

README.md

适用硬件平台环境说明

硬件平台：算能bm1684 SOC产品

编译链：aarch64-linux-gnu-g++7.5，cmake 3.20

系统版本:

VERSION: 2.6.0

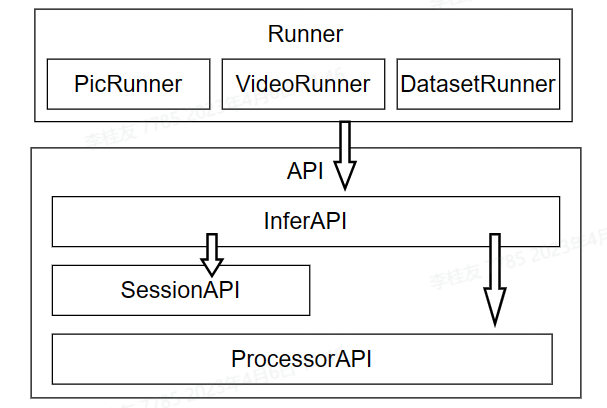
KernelVersion : Linux bm1684 4.9.38-bm1684-v10.3.0-00528-g8be6792 #2 SMP Sun Jan 30 07:12:27 CST 2022 aarch64 GNU/Linux

HWVersion: 0x03

MCUVersion: 0x34

# 2.SDK使用说明

SDK提供Runner和API两个层次的接口，软件包含示意图如下：



Runner为针对某个目的功能的类定义，目前提供图片推理、视频流推理、数据集测试准确率三个功能；

API主要是单帧数据进行模型推理功能，针对不同客户功能要求进行粗细粒度分层；分别是：

1）针对不关心底层和资源管理的客户，使用InferSync/InferAsync等简易接口；

2）针对本身也有其他算法等需要管理硬件资源，使用SessionAPI接口，其中包含InferService/context/session接口设置和控制资源使用情况；

3）针对有开发能力比较强的客户，提供ProcessorAPI接口，包含算法的pre/infer/post三个单元，用户自行决定调用的调度策略；

针对模型计算量大小差异，推荐使用API如下：

1）算力小，一般是端侧推理，建议使用简约接口和pre/infer/post接口

2）算力中等以上：建议使用资源接口

**2.1 Runner说明**

Runner是针对某一特定目的可直接运行程序，比如读取图片和推理，读取视频流和推理，读取数据集和推理获取结果进行准确率测试。API是指单纯的算法推理。Runner是在API基础上做的一些方便用户快速搭建使用的方式。源码也在release的sample/app目录，用户可自行修改；

包括如下runner：

PicRunner：推理单图片接口，有同步和异步接口，包含图片解码->推理->结果解析和画图

VideoRunner：包含ffmpeg和opencv版本，可进行一路视频流解码和推理或者多路重复推理

DatasetRunner：输入推理数据集图片，获取结果，视模型类型最后保存json文件或者直接打印准确率

SDK包中bin目录已有编译好可执行程序，可直接在每个可执行程序--help查看用法

以图片推理作为例子解释推理基本流程：

cv::Mat in\_mat = cv::imread(pic\_path); // 读取图片文件解码

gddeploy::BufSurfWrapperPtr surf;

#if WITH\_BM1684

bm\_image img;

cv::bmcv::toBMI(in\_mat, &img, true); // bmnn一般使用bm\_image结构体进行后续操作

convertBmImage2BufSurface(img, surf, false); // bm\_image转为surface

#else

convertMat2BufSurface(in\_mat, surf, true);

#endif

// 创建输入输出对象空间

gddeploy::PackagePtr in = gddeploy::Package::Create(1);

in->data[0]->Set(surf);

gddeploy::PackagePtr out = gddeploy::Package::Create(1);

// 调用InferAPI接口进行推理

infer\_api\_.InferSync(in, out);

// 取出结果中的MetaData，解析结果并打印结果

gddeploy::InferResult result = out->data[0]->GetMetaData<gddeploy::InferResult>();

PrintResult(result);

可以看到调用的基本流程如下：

1）Init，详看2.2部分的Init所需参数；一般是读取模型和一些全局设置；

2）读取多媒体解码得到一帧数据；

3）转换为surface内存格式；

4）创建推理的输入输出package空间；

5）推理帧数据；

6）取出结果，解析结果，并进行结果的打印或者绘画结果到原图等操作；

可以看到模型推理输入的结构体为Package，详细可看2.4结构体定义说明；设计考量主要是为方便用户输入一帧或者多帧数据的灵活性和多batch一般对于推理性能的提高。而推理结果也会在对应的data变量的MetaData可以获取；

前端解码后结构体需要进行转为gddeploy::BufSurfaceWrapper类型，再赋值package中data

OpenCV的Mat转Package，参考type\_convert.cpp文件

FFmpeg的AVframe转Package，参考type\_convert.cpp文件

注意：

如果surface的data\_ptr直接指向原数据帧空间，请务必存活整个周期，否则请新建和拷贝；

**2.2 API调用流程：**

如上所述，API功能是完成帧数据的推理功能，不包含前端解码和后续开发；有三种不同层次粗细粒度，均需要用户自行完成图片/视频解码后转为特定内存结构进行输入。

**2.2.1 Infer API**

对应sample/infer\_api.cpp文件，本质为调用Session API和Processor API实现的进一步封装更简易的接口，可通过Init函数设置使用哪种底层接口。

class InferAPI{

public:

InferAPI();

~InferAPI();

// api\_type: 选择底层的api接口为processor或者session api，区别在于有无预分配空间

void Init(std::string config, std::string model\_path, ENUM\_API\_TYPE api\_type = ENUM\_API\_PROCESSOR\_API);

// 同步接口

int InferSync(const gddeploy::PackagePtr &in, gddeploy::PackagePtr &out); //opencv4可支持解码图片格式

// 异步接口

void SetCallback(InferAsyncCallback cb);

int InferAsync(const gddeploy::PackagePtr &in, InferAsyncCallback cb = nullptr, int timeout = 0);

int WaitTaskDone(const std::string& tag="");

std::string GetModelType();

private:

std::shared\_ptr<InferAPIPrivate> priv\_;

};

注意：Processor API为单算法的前处理、推理、后处理，因此不会有异步接口功能，如果希望使用异步接口需要Init函数选用ENUM\_API\_SESSION\_API参数；

**2.2.2 Session API**

Session API一般针对中高算力，可同时推理多路视频流或者高吞吐量场景。需要在Init阶段进行资源的提前划分，内存预分配等操作，而且推理阶段尽可能是异构流水线并行计算，以达到最高的使用性能。因此建议使用异步接口，使用过程中需要多次调整参数

class SessionAPI{

public:

SessionAPI();

int Init(const std::string config, const std::string model\_path, const std::string properties\_path = "");

int Init(const SessionAPI\_Param &config, const std::string model\_path, const std::string properties\_path = "");

// 同步接口

int InferSync(const gddeploy::PackagePtr &in, gddeploy::PackagePtr &out); //opencv4可支持解码图片格式

// 异步接口

void SetCallback(InferAsyncCallback cb);

int InferAsync(const gddeploy::PackagePtr &in, InferAsyncCallback cb = nullptr, int timeout = 0);

int WaitTaskDone(const std::string& tag="");

std::string GetModelType();

std::vector<std::string> GetLabels();

private:

std::shared\_ptr<SessionAPIPrivate> priv\_;

};

其中需要在Init就进行资源参数的设置，定义如下：

typedef struct {

std::string name;

BatchStrategy strategy; // 可选static和dynamic

int batch\_timeout; // dynamic时可用

int engine\_num; // 底层可并行运行engine个数

int priority; // 优先级

bool show\_perf; // default false

} SessionAPI\_Param;

对于中高算力硬件设备，大多具有batch可以明显提高推理性能的特点，比如bm1684的Batch4可以达到Batch1一样的推理时间，Nvidia的TensorRT中Batch2~Batch4有30%~50%的推理性能提高；因此对于高吞吐量场景尽量选用dynamic。

**2.2.3 Processor API**

模型推理基本包含算法前处理、推理、算法后处理。每一部分均为一个Processor单元，串起来构建得到算法的pipeline，相比于SessionAPI的优势的比较轻量级，简单，适用于算力较小或推理实时性强的使用场景。

class ProcessorAPI{

public:

ProcessorAPI();

void Init(std::string config, std::string model\_path);

// 根据模型获取processor，用于最基础层的接口

std::vector<ProcessorPtr> GetProcessor();

std::string GetModelType();

private:

std::shared\_ptr<ProcessorAPIPriv> priv\_;

};

可以看到类定义非常的简介，也即是解析模型得到对应算法的前处理、推理、后处理单元，获取到Processor对象后，逐个调用输入package对象即可推理，可以参考InferAPI源码部分如下：

int InferAPIPrivate::InferSync(const gddeploy::PackagePtr &in, gddeploy::PackagePtr &out)

{

if (api\_type\_ == ENUM\_API\_PROCESSOR\_API){

// 4. 循环执行每个processor的Process函数

for (auto processor : processors\_){

processor->Process(in);

}

out = in;

}

return 0;

}

**2.3 API详细说明**

详看对应头文件说明

**2.4 数据结构定义：**

gddeploy::PackagePtr说明：

异步接口时，如果直接赋值data\_ptr指向空间，请务必存活整个周期，也可让gddeploy::BufSurfaceWrapper托管释放空间

struct Package

{

/// a batch of data, origin data

BatchData data;

/// private member, intermediate storage，可能会作为前处理和推理后数据临时存储

InferDataPtr predict\_io{nullptr};

/// tag of this package (such as stream\_id, client ip, etc.)

std::string tag;

/// perf statistics of one request

std::map<std::string, float> perf;

/// private member

int64\_t priority;

static std::shared\_ptr<Package> Create(uint32\_t data\_num, const std::string &tag = "") noexcept

{

auto ret = std::make\_shared<Package>();

ret->data.reserve(data\_num);

for (uint32\_t idx = 0; idx < data\_num; ++idx)

{

ret->data.emplace\_back(new InferData);

}

ret->tag = tag;

return ret;

}

};

使用技巧：

data保存了输入的帧数据，如果需要多模型串联，可以重复从中裁剪帧数据继续送入第二模型处理

**2.5 结果解析说明：**

具体详细说明请看result\_def.h

typedef struct {

std::vector<int> result\_type;

DetectResult detect\_result;

DetectPoseResult detect\_pose\_result;

ClassifyResult classify\_result;

SegResult seg\_result;

ImageRetrievalResult image\_retrieval\_result;

FaceRetrievalResult face\_retrieval\_result;

OcrDetectResult ocr\_detect\_result;

OcrRecResult ocr\_rec\_result;

void \*user\_data;

}InferResult;

在实际运行过程中需要多个模型串联，为使后需要模型可以用上一模型的结果，设计为把各类算法结果统一起来，因此解析的时候首先读取result\_type，然后解析对应算法结构体；

**2.6 内存管理说明**

内存/显存采用surface结构体管理的方式，定义解析如下：

/\*\*

\* Holds information about a single buffer in a batch.

\*/

typedef struct BufSurfaceParams {

/\*\* Holds the width of the buffer. \*/

uint32\_t width;

/\*\* Holds the height of the buffer. \*/

uint32\_t height;

/\*\* Holds the pitch of the buffer. \*/

uint32\_t pitch;

/\*\* Holds the color format of the buffer. \*/

BufSurfaceColorFormat color\_format;

/\*\* Holds the amount of allocated memory. \*/

uint32\_t data\_size;

/\*\* Holds a pointer to allocated memory. \*/

void \* data\_ptr;

/\*\* Holds a pointer to a CPU mapped buffer.

Valid only for CNEDK\_BUF\_MEM\_UNIFIED\* and CNEDK\_BUF\_MEM\_VB\* \*/

void \* mapped\_data\_ptr;

/\*\* Holds planewise information (width, height, pitch, offset, etc.). \*/

BufSurfacePlaneParams plane\_params;

void \* \_reserved[CNEDK\_PADDING\_LENGTH];

} BufSurfaceParams;

/\*\*

\* Holds information about batched buffers.

\*/

typedef struct BufSurface {

/\*\* Holds type of memory for buffers in the batch. \*/

BufSurfaceMemType mem\_type;

/\*\* Holds a Device ID. \*/

uint32\_t device\_id;

/\*\* Holds the batch size. \*/

uint32\_t batch\_size;

/\*\* Holds the number valid and filled buffers. Initialized to zero when

an instance of the structure is created. \*/

uint32\_t num\_filled;

/\*\* Holds an "is contiguous" flag. If set, memory allocated for the batch

is contiguous. Not valid for CNEDK\_BUF\_MEM\_VB on CE3226 \*/

bool is\_contiguous;

/\*\* Holds a pointer to an array of batched buffers. \*/

BufSurfaceParams \*surface\_list;

/\*\* Holds a pointer to the buffer pool context \*/

void \*opaque;

/\*\* Holds the timestamp for video image, valid only for batch\_size == 1 \*/

uint64\_t pts;

void \* \_reserved[CNEDK\_PADDING\_LENGTH];

} BufSurface;

补充说明：

这里的batch\_size决定surface\_list有多少个，而BufSurfaceParams中data\_ptr都指向对于batch的地址，data\_size一般为CHW\*sizeof(pixel\_size)大小，BufSurfacePlaneParams中的Plane是是指一个通道的数据，也就是HW

以batch4为例，如果数据排布如下：

暂时无法在飞书文档外展示此内容

每个batch idx的大小为channel\*height\*width,

BufSurfacePlaneParamsoffset是地址偏移宽度，一般640\*640\*sizeof(float),。可以参考GetColorFormatInfo函数赋值

内存/显存操作有三个头文件，分别的功能如下：

buf\_surface\_utils.h：主要是BufSurfaceWrapper和BufPool类定义，分别是surface智能指针管理和内存池作用

buf\_surface.h：BufSurfaceService类及其接口定义，主要是surface结构体池和分配

buf\_surface\_impl.h：接口类，主要是MemPool内存池和MemAllocator内存分配器定义，MemAllocator是接口类，各个设备的显存接口需要继承和实现

用户侧分配内存/显存做法：

固定已知内存/显存大小，需要预分配，建议采用BufPool方式预分配内存，按需请求获取使用

未知内存/显存大小，临时创建和申请，建议采用CreateSurface创建surface和MemAllocator分配内/显存

# 3.工具使用

**3.1 获取硬件SN**

# 拷贝此工具到目标硬件设备 ，然后执行

./tools/gtx\_maker

**3.2 测试应用程序**

图片推理测试：

# ./bin/sample\_runner\_pic -h

Options:

-h [ --help ] Help screen

--model arg model file path

--pic-path arg pic file path

--save-pic arg save file path

#视频推理测试：

# ./bin/sample\_runner\_video -h

Options:

-h [ --help ] Help screen

--model arg model file path

--video-path arg video file path

--multi-stream arg (=1) multi stream

--is-save arg (=1) is save result pic

--save-pic arg model file path

# 数据集准确率测试：

./bin/sample\_runner\_dataset -h

Options:

-h [ --help ] Help screen

--model arg model file path

--anno-file arg anno file path

--pic-path arg dataset pic path

--result-path arg result file save path

--save-pic arg draw result on pic and save path

1. **sample说明**

（后续开放）

多线程流程：

多模型独立：

多模型依赖：

多模型多线程：

1. **FAQ**

1）有哪些硬件加速技巧，达到最大吞吐量

答：以下操作均有提高性能tricks，请逐个尝试：

* 解码后映射送入推理，不拷贝
* 选用SessionAPI接口，推理时选用异步接口
* Session参数中增大batch，一般建议2~4即可，同时timeout设置100ms
* 回调函数非阻塞，尽量阻塞的操作通过消息队列等方式在另外线程进行；
* 使用硬件解码编码
* engine数量，和推理单元数量一致，过低和过高均影响速度

2）如何设置log等级：

export SPDLOG\_LEVEL=info

目前支持等级：trace/debug/info/warn/err/critical/off

同一模型的不同session，设置参数最好一致，以便最高效使用硬件。