




DLIA-SDK 使用说明

深度学习应用于工业质检

DLIA-SDK™ 使用说明(Rev 1.0)

 Copyright (C) 2021-2026 numimag Co., Ltd.

目录

1. DLIA-SDK概述
2. 详细说明
 - 2.1. 配置运行环境
 - 2.2. C# 工程
 - 2.3. C++ 工程
 - 2.4. Java 工程
 - 2.5. LabVIEW 工程
 - 2.6. Python 工程
 - 2.7. DLIA-SDK API应用流程图
 - 2.8. DLIA-SDK API定义
 - 2.9. DLIA-SDK 返回数据说明
 - 2.9.1. 基本格式
 - 2.9.2. 二维码/条码格式
 - 2.10. 软件激活
 - 2.10.1. 使用硬加密狗
 - 2.10.2. 使用软加密狗
 - 2.11. DLIA-SDK示例代码与调试
 - 2.11.1. 例程
 - 2.11.2. 单步调试
 - 2.12. DLIA-SDK HTTP API
 - 2.12.1. 查询http版本
 - 2.12.2. 查询模型信息
 - 2.12.3. 模型初始化
 - 2.12.4. 图片推理测试
 - 2.12.5. Web测试
3. 附录1 错误代码
4. 附录2 JSON数据提取示例代码
5. 附录3 DLIA-SDK使用常见问题
6. 附录4 DLIA-SDK安装步骤
7. 附录5 DLIA-SDK卸载

1. DLIA SDK概述

DLIA™ (Deep Learning for Industrial Applications) SDK是为了帮用户跳出复杂、繁琐技术细节，快速开发自己应用程序、快速落地具体的工业质检或特定目标识别场景应用，且支持主流系统平台、编程语言的产品。

- SDK 列表

SDK	编程语言	系统平台
DLIA-SDK-Win64	C# / C++ / Java / LabVIEW / Python ^[1] / HTTP API	Windows 10/11 x64
DLIA-SDK-Linux64	C++	Linux amd64 ^[2]
DLIA-SDK-Linux64-Docker	HTTP API	Linux amd64 Docker ^[3]

- [1] LabVIEW 16.0+ 版本，Python 3.6+ 版本。
- [2] Linux amd64: Ubuntu 18.04及以上版本。
- [3] Linux amd64 Docker: http协议，详细API定义请联系厂家。

2. DLIA-SDK详细说明

2.1. 配置运行环境

以Windows系统为例，在使用SDK开发之前，首先要确保主机显卡及操作系统满足：

- NVIDIA GTX 1050或以上的显卡配置；
- Windows 10/11 64位操作系统 (推荐)；

根据厂家提供的下载地址，下载解压DLIA-SDK压缩包，双击执行Setup.exe，按照软件界面提示，一步一步操作，直至安装完成，详细安装步骤见[附录4 DLIA SDK安装步骤](#)。

在安装路径(默认为C:\Program Files\DLIA Runtime\x64文件)，包含SDK运行依赖，C++/C#程序示例等，其中SDK调用例子在Samples文件夹下。Samples子文件夹包含对应的应用工程文件、源码，模型以及测试图片，使用SDK开发，根据不同的模型，选择不同的依赖和头文件，模型、依赖dll及头文件三者对应关系如下：

模型	依赖dll	头文件
目标检测/分割/分类模型 (mm01/mm01s/mm01_p/my01/my01s/my01s_p/my01s/mx01)	dlia_runtime_x64.dll	dlia_runtime_export.h
增强型分类(mst)	dlia_runtime_mst_x64.dll	dlia_runtime_mst_export.h
非监督AI模型(ss01)	dlia_runtime_ss01_x64.dll	dlia_runtime_ss01_export.h

2.2. C# 工程

SDK目录下的C#工程使用DotNet开发，仅为演示dll使用，用户可根据实际需求，选用比如 .NET Framework 框架进行开发。

2.2.1 项目结构

```
C#/  
├── DLIA_Runtime_CSharp/           # 图形界面项目  
│   ├── Properties/              # 属性配置  
│   ├── DLiaRuntime.cs          # 核心 SDK 封装类  
│   ├── Form1.cs                # 图形界面表单  
│   ├── Form1.Designer.cs       # 图形界面设计器  
│   ├── Form1.resx              # 图形界面资源  
│   ├── Program.cs              # 程序入口  
│   └── DLIA_Runtime_CSharp.csproj  
├── dll-test.cs                  # 命令行示例代码  
├── dlia_runtime_sdk_cs.csproj    # 命令行项目配置  
├── dlia_runtime_sdk_cs.sln      # 解决方案文件  
├── dlia_runtime_sdk_gui.csproj  # 图形界面项目配置  
├── lib/                          # 依赖库  
│   └── OpenCvSharp/            # OpenCvSharp 库  
├── x64/                          # 运行时库  
│   ├── Debug/                 # Debug 版本  
│   └── Release/                # Release 版本  
├── obj/                          # 编译输出目录  
├── app.manifest                 # 应用程序清单  
└── DLIA_Runtime_CSharp_SDK.md   # SDK 文档
```

2.2.2 系统要求

- .NET Core 3.1 或更高版本
- Windows 64位操作系统
- DLIA Runtime 环境

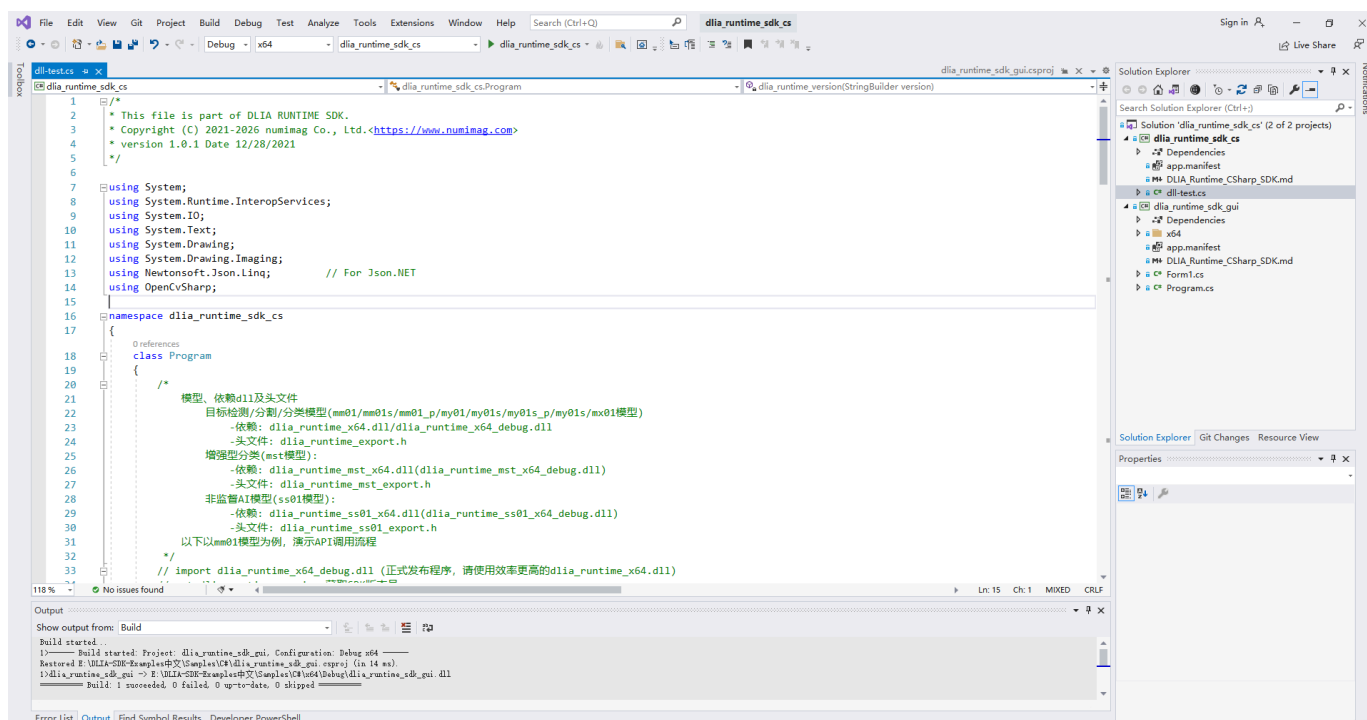
2.2.3 依赖项

SDK 依赖以下库:

依赖项	版本	用途
OpenCvSharp	4.5.3	用于图像处理和预览
Newtonsoft.Json	13.0.1	用于解析 JSON 格式的推理结果

2.2.4 配置方法

- 拷贝DLIA Runtime安装路径里面bin文件夹中(默认安装路径: C:\Program Files\DLIA Runtime\x64\bin)所有dll到工程 exe 目录下, [激软件活](#), 即可编译运行, 具体代码详情请参考SDK代码
- 在项目中添加对 OpenCvSharp 和 Newtonsoft.Json 的引用
- 在代码中添加 P/Invoke 声明, 用于调用 DLIA Runtime DLL



- C# 例程代码片段

```
// 使用dlia_runtime_infer_cv进行推理
// inference with API "dlia_runtime_infer_cv"
{
    Console.WriteLine("inference with API \"dlia_runtime_infer_cv\"");

    Mat cvimg = Cv2.ImRead(image, ImreadModes.Color);
    String img_id = "dlia_runtime_infer_cv.jpg";

    rst = dlia_runtime_infer_cv(0, img_id, cvimg.CvPtr, ref results[0], ref res_len); //
    使用Mat图像进行推理
}
```

```

    if (rst != 0)
    {
        Console.WriteLine("dlia runtime inference error: {0}", rst);
    }
    else
    {
        Console.WriteLine("inference results: " + Encoding.UTF8.GetString(results, 0,
(int)res_len));
    }
}

// 使用dlia_runtime_infer_im进行推理
// inference with API "dlia_runtime_infer_im"
{
    Console.WriteLine("inference with API \"dlia_runtime_infer_im\"");

    var img = System.Drawing.Image.FromFile(image);
    String img_id = "dlia_runtime_infer_im.jpg";
    using (var ms = new MemoryStream())
    {
        img.Save(ms, img.RawFormat);
        rst = dlia_runtime_infer_im(0, img_id, ms.ToArray(), (ulong)ms.Length, ref
results[0], ref res_len); //使用已编码encode图像buffer进行推理

        if (rst != 0)
        {
            Console.WriteLine("dlia runtime inference error: {0}", rst);
        }
        else
        {
            Console.WriteLine("inference results: " + Encoding.UTF8.GetString(results, 0,
(int)res_len));
        }
    }
}

// 使用dlia_runtime_infer_im_rawdata进行推理
// inference with API "dlia_runtime_infer_im_rawdata"
{
    Console.WriteLine("inference with API \"dlia_runtime_infer_im_rawdata\"");

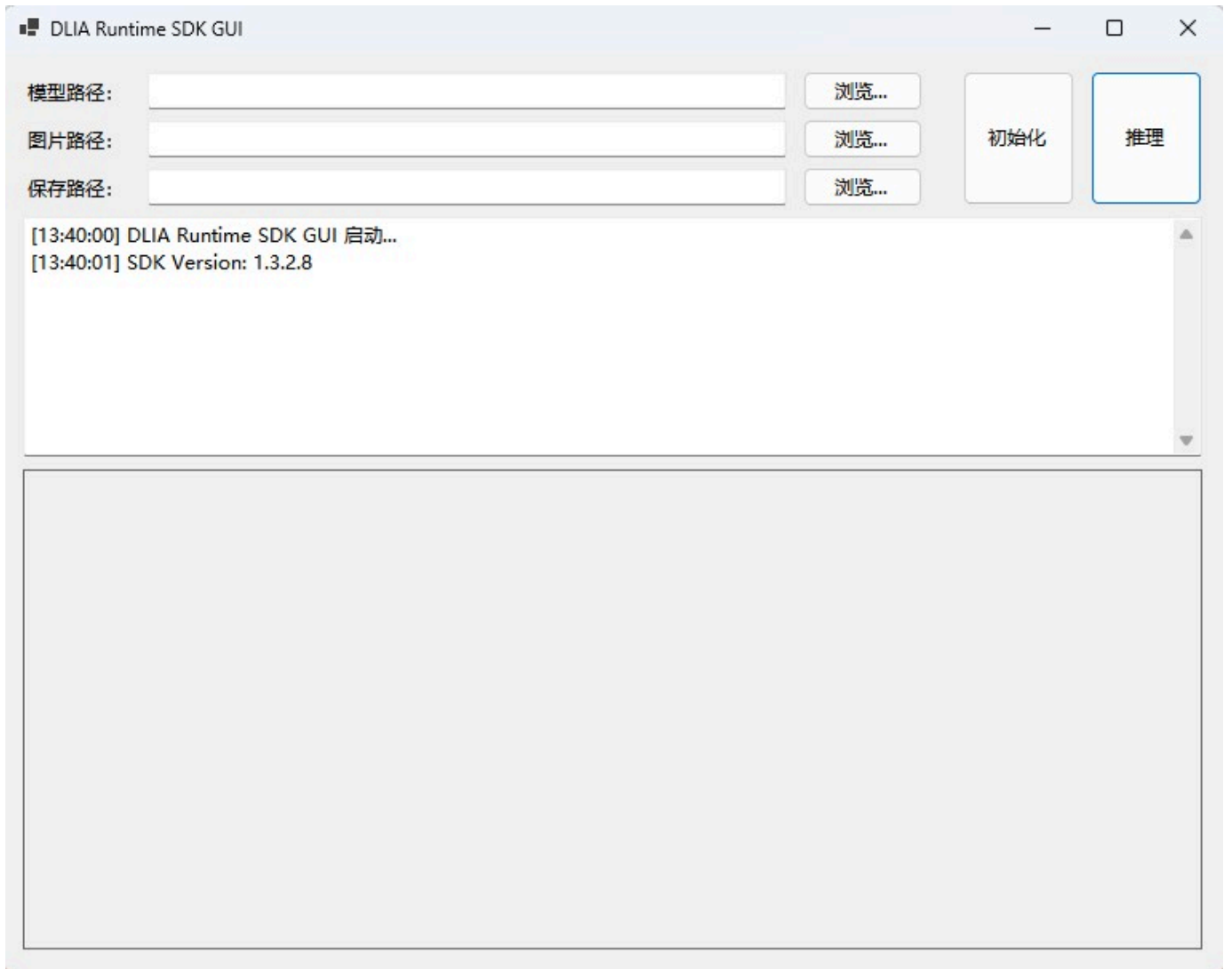
    var img_bitmap = (Bitmap)System.Drawing.Image.FromFile(image);
    String img_id = "dlia_runtime_infer_im_rawdata.jpg";
    bool color_rgb = false;
    int channel = 3;
    rst = dlia_runtime_infer_im_rawdata(0, img_id, BitmapToBytes(img_bitmap),
img_bitmap.Width, img_bitmap.Height, color_rgb, channel, ref results[0], ref res_len);

    if (rst != 0)
    {
        Console.WriteLine("dlia runtime inference error: {0}", rst);
    }
    else
    {
        Console.WriteLine("inference results: " + Encoding.UTF8.GetString(results, 0,
(int)res_len));
    }
}

```

```
}  
}
```

- C# 例程测试界面



2.3. C++ 工程

2.3.1 项目结构

```
C++/
├── include/                # 头文件目录
│   ├── dlia_runtime_export.h # 核心 API 头文件
│   ├── dlia_runtime_mst_export.h # 增强型分类模型头文件
│   └── dlia_runtime_ss01_export.h # 非监督AI模型头文件
├── lib/                    # 库文件目录
├── samples/                # 示例代码目录
│   ├── sample_01/         # 基本功能使用
│   ├── sample_02/         # 多线程使用
│   ├── sample_03/         # 多线程多显卡使用
│   ├── sample_04/         # CPU使用
│   ├── sample_05/         # 原始数据API使用
│   └── sample_06/         # 编码图像API使用
├── 3rdparty/              # 第三方依赖
│   ├── nlohmann/json      # JSON 解析库
│   ├── opencv              # OpenCV 库
│   ├── stb                 # 图像加载库
│   └── utfcpp              # UTF-8 处理库
└── DLIA_Runtime_Cpp_SDK.md # SDK 文档
```

2.3.2 系统要求

- C++11 或更高版本
- Windows 64位操作系统
- DLIA Runtime 环境
- OpenCV (可选, 用于图像处理)

2.3.3 依赖项

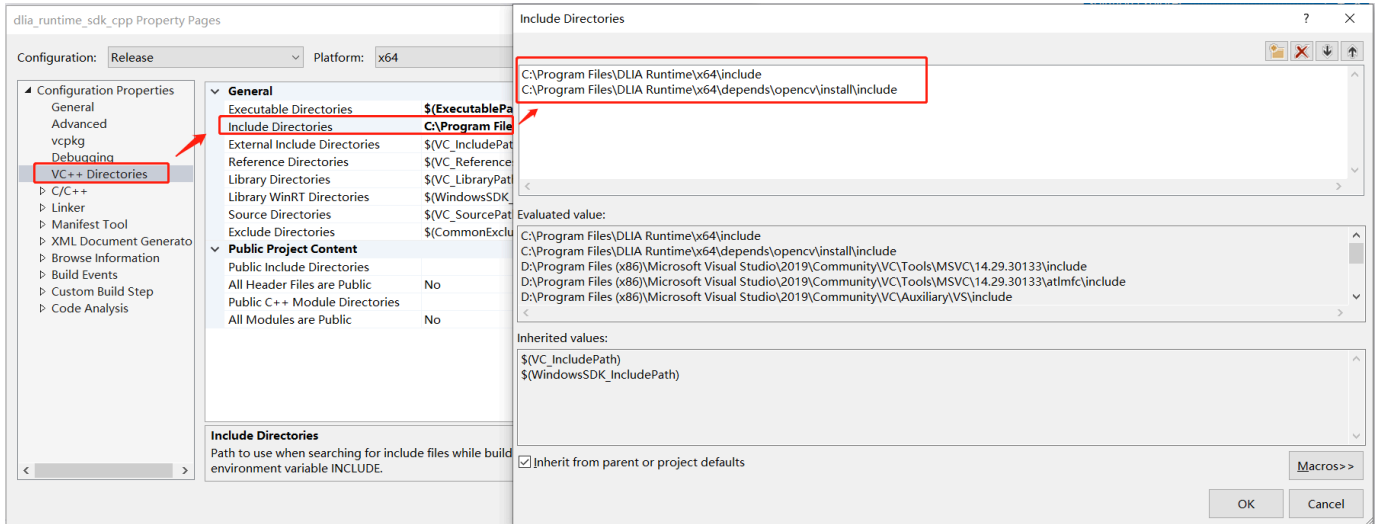
SDK 依赖以下库:

依赖项	用途	是否必需
DLIA Runtime DLL	核心推理库	是
OpenCV	用于图像处理	否
nlohmann/json	用于解析 JSON 格式的推理结果	是
stb_image	用于加载原始图像数据	否

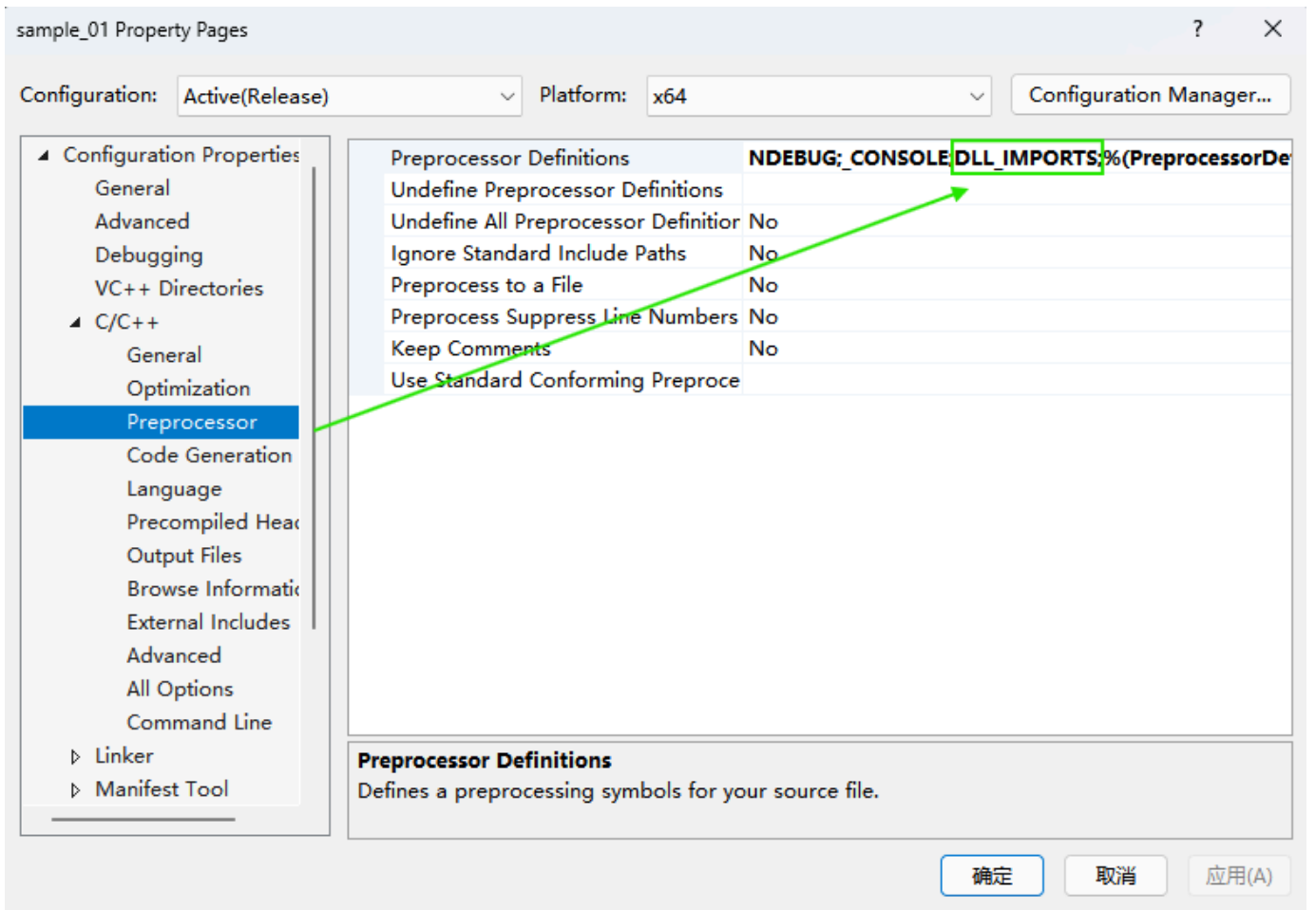
2.3.4 配置方法

建议复制SDK目录下的C++工程, 进行修改开发, 若需要新建C++工程, 则需要手动配置头文件路径、依赖库及路径。

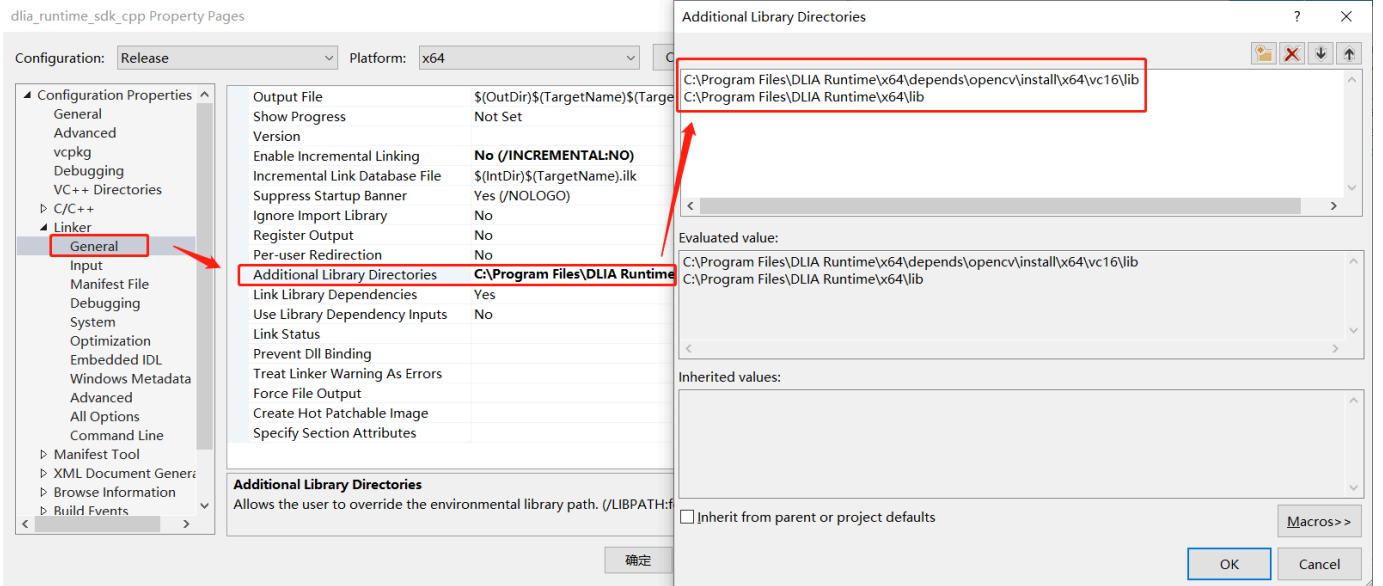
- 设置头文件路径:



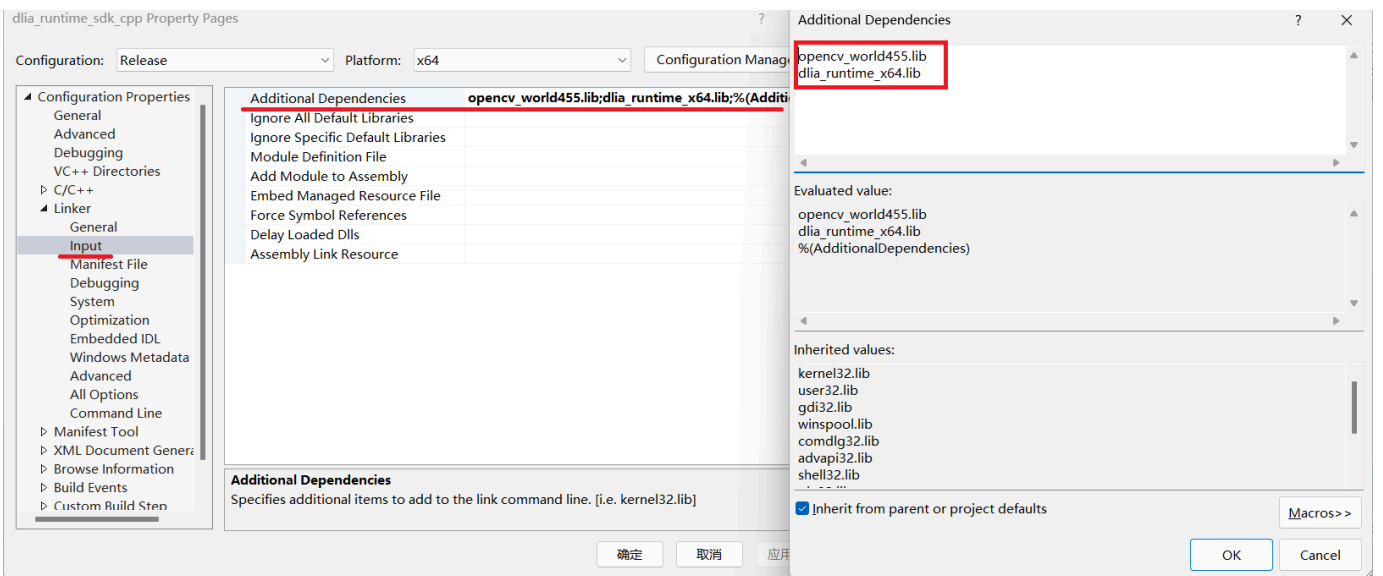
- 添加预定义宏DLL_IMPORTS:



- 设置链接库文件路径:



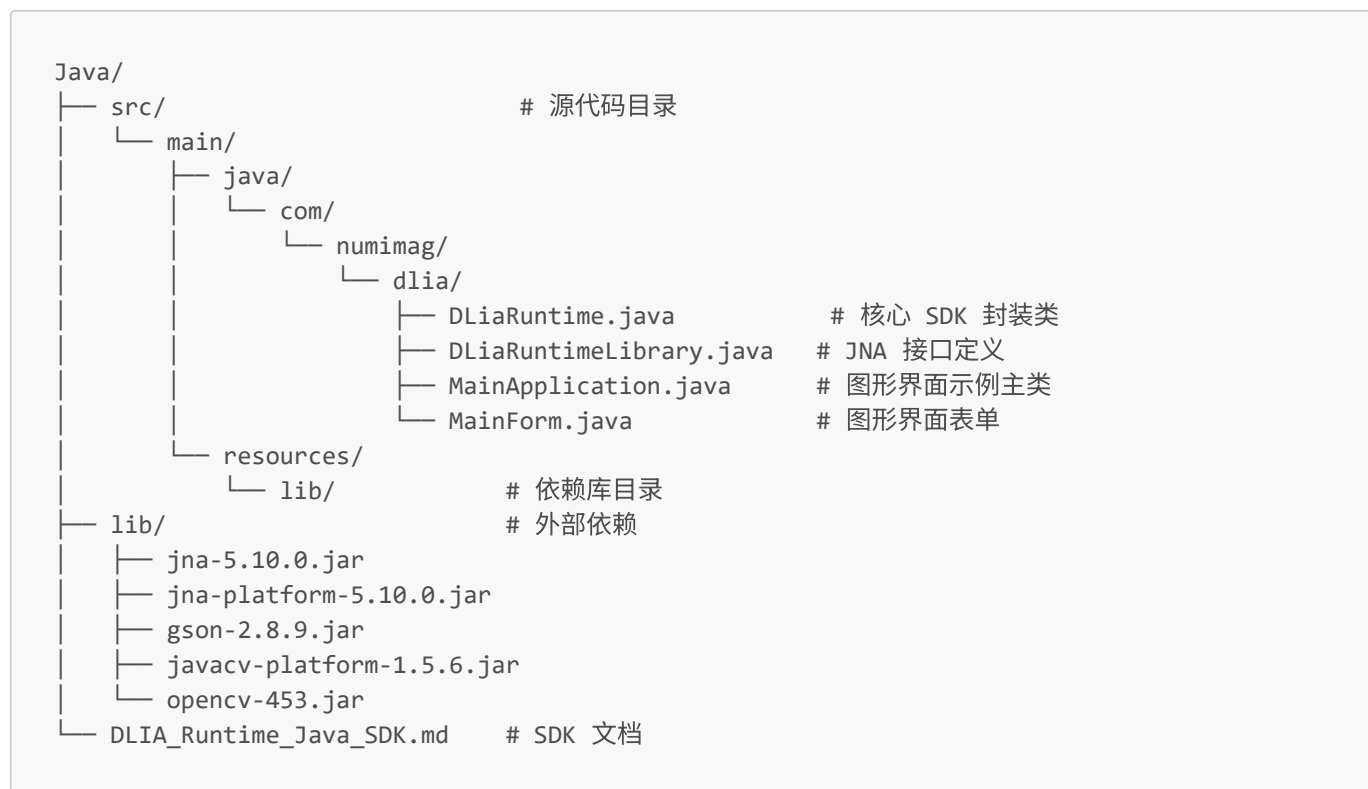
- 设置链接库名称:



拷贝DLIA Runtime安装路径里面bin文件夹中(默认安装路径: C:\Program Files\DLIA Runtime\x64\bin)所有dll到工程exe目录下, [激软件活](#) 即可编译运行, 具体代码详情请参考SDK代码

2.4. Java 工程

2.4.1 项目结构



2.4.2 系统要求

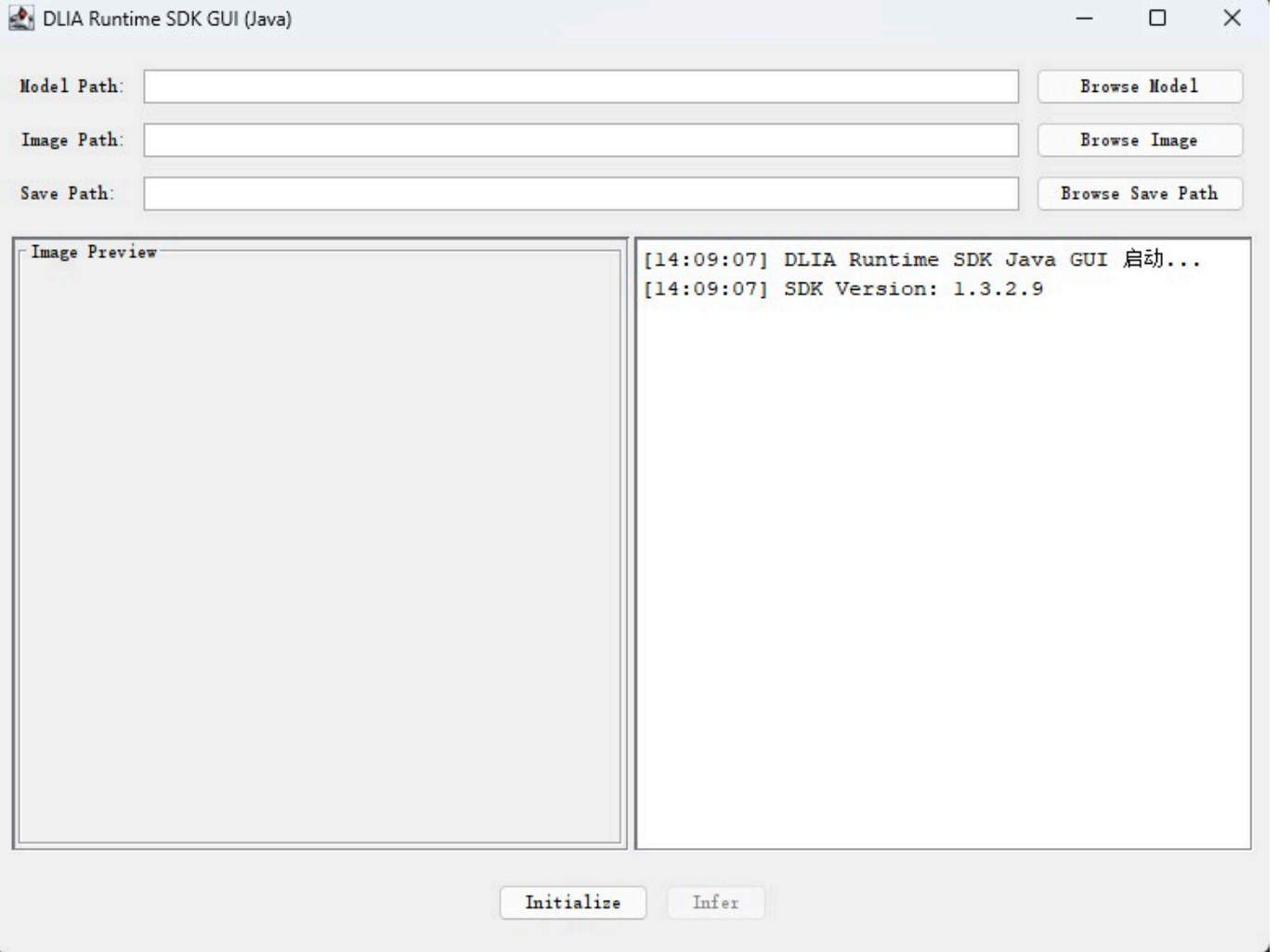
- Java 8 或更高版本
- Windows 64位操作系统
- DLIA Runtime 环境

2.4.3 依赖项

SDK 依赖以下库：

依赖项	版本	用途
JNA	5.10.0	用于调用 DLIA Runtime DLL
GSON	2.8.9	用于解析 JSON 格式的推理结果
JavaCV	1.5.6	用于图像处理和预览
JavaCPP	1.5.6	JavaCV 的依赖
OpenCV	4.5.3	用于图像处理
OpenBLAS	0.3.17	OpenCV 的依赖

2.4.4 例程测试界面



2.5. LabVIEW 工程

2.5.1 系统要求

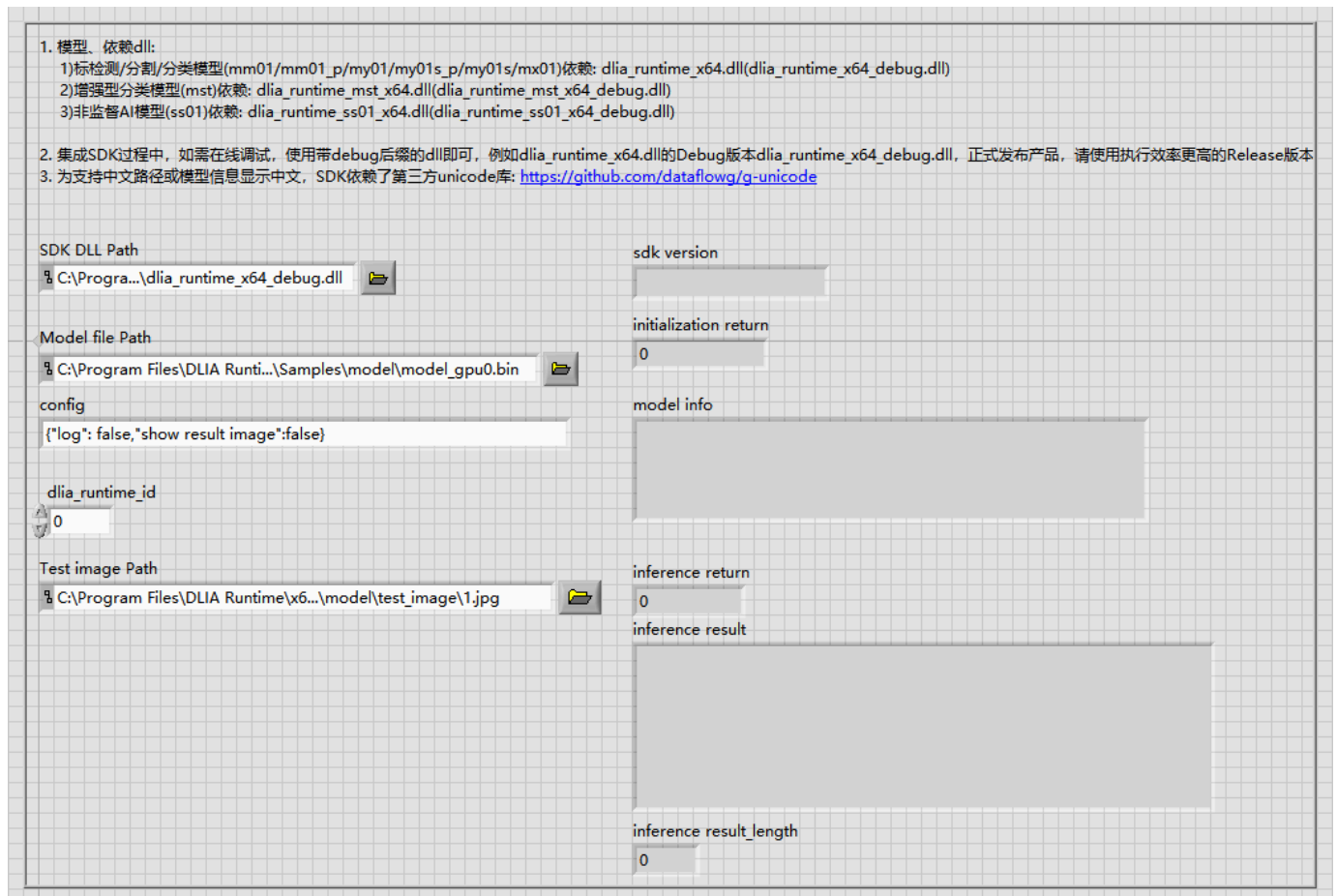
- LabVIEW 16 或更高版本
- Windows 64位操作系统
- DLIA Runtime 环境

2.5.2 依赖项

依赖项	版本	用途
g-unicode	0.3.0	unicode库，支持中文路径或模型信息显示中文 ^[1]

[1] github地址: <https://github.com/dataflowg/g-unicode>

2.5.3 例程测试界面



2.6. Python 工程

2.6.1 目录结构

```
Python/  
├── DLiaRuntime.py          # 核心 SDK 封装类  
├── sample_cli.py          # 命令行示例  
├── sample_gui_pil.py      # 图形界面示例  
├── image_loader.py        # 图像加载工具  
└── DLIA_Runtime_Python_SDK.md # SDK 文档
```

2.6.2 系统要求

- Python 3.6 或更高版本
- Windows 64位操作系统
- DLIA Runtime 环境

2.6.3 依赖项

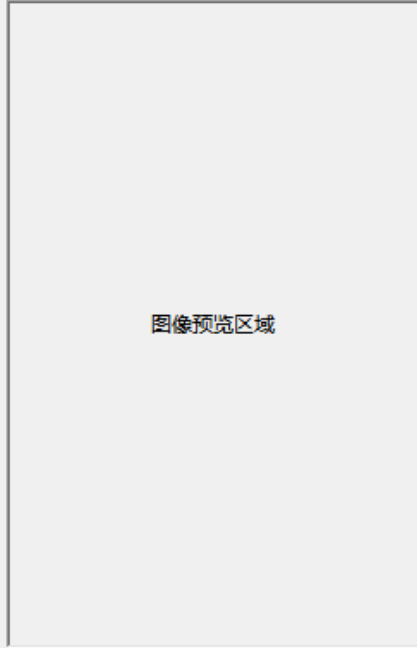
依赖项	用途	是否必需
ctypes	用于调用 DLIA Runtime DLL	是
json	用于解析 JSON 格式的推理结果	是
os	用于文件操作	是
pathlib	用于文件路径操作	是
Pillow (PIL)	用于图像预览和绘制	否
tkinter	用于图形界面	否

2.6.3 例程测试界面

模型路径:

图像路径:

保存路径:

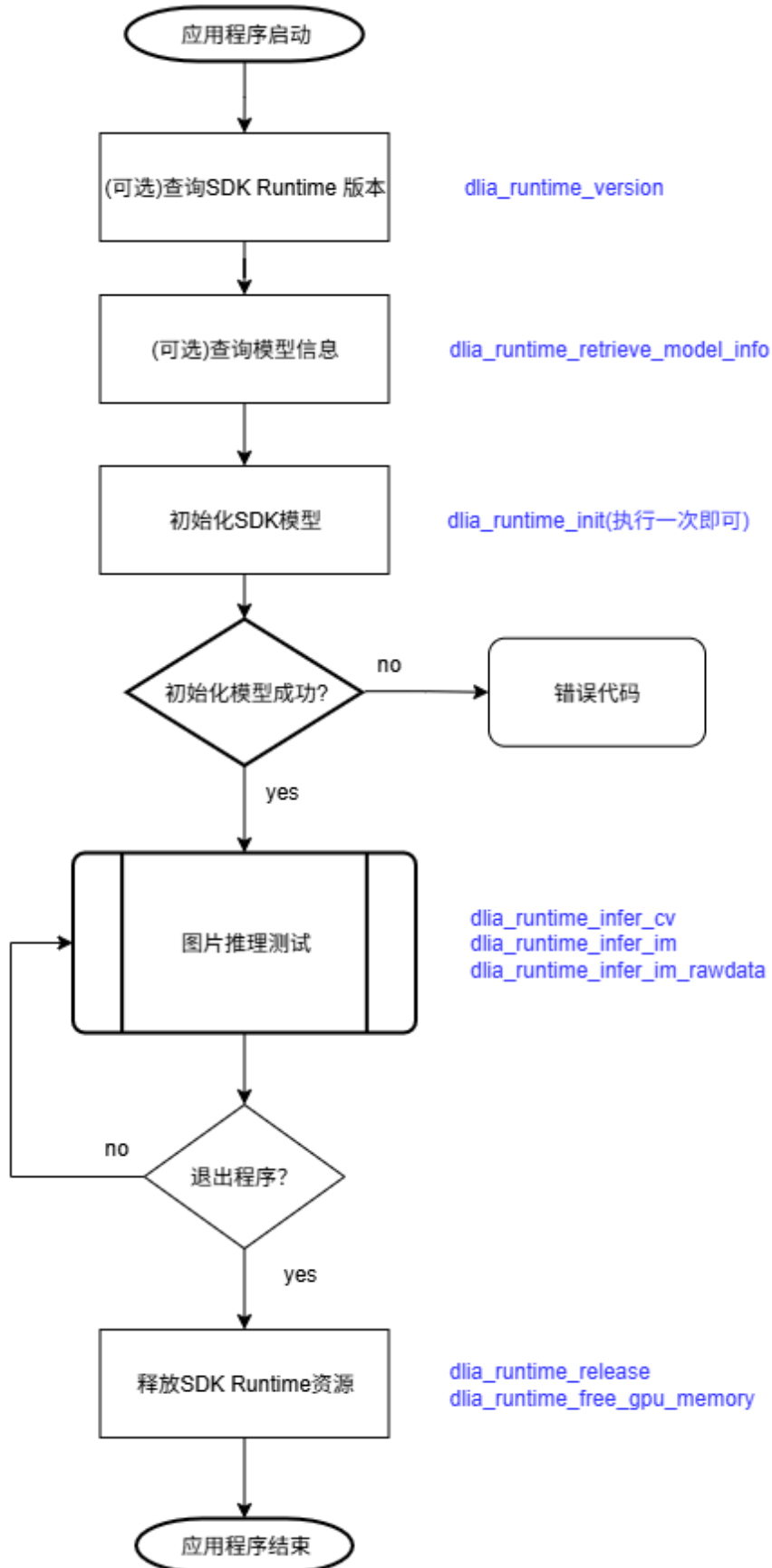


日志输出:

```
[14:56:08] DLIA Runtime SDK GUI 启动...  
[14:56:08] 图像预览支持: JPG/JPEG, BMP, PNG
```

2.7. DLIA-SDK API应用流程图

应用程序调用SDK API如下图，初始化模型和释放SDK资源函数只需执行一次即可，对于多线程应用，确保线程id与API函数dliaruntime_id唯一对应关系，详见SDK使用例子。



2.8. DLIA-SDK API定义

```

/*
    函数 DLIA Runtime 版本获取
    输入: 版本信息, char 类型数组(字符串)
    返回: 无
*/
void dlia_runtime_version(char* ver);

/*
    函数 DLIA Runtime 初始化
    输入:
        dlia_runtime_id : 实例ID, int类型, 多线程应用时, 线程与该实例ID一一对应;
        model_path      : 模型文件的绝对路径;
        config          : 配置参数, 为JSON格式化字符串, 如果使用默认参数, 定义 string
config = "{}" 即可, 如需自定义配置, 请参考下表
    返回: 0 - 执行成功, 其它 - 失败, 详细请参考SDK文档中错误代码说明
*/
int dlia_runtime_init(int dlia_runtime_id, const char* model_path, const char*
config);

```

config参数	类型	是否必须	备注
show result image	bool	否	显示识别结果, 用于调试, 在执行目录下保存推理测试结果图片, 不设置情况下, 默认false
confThreshold	float	否	全局分数(置信度), 针对所有标签, 低于设定值, 会过滤标签, 分数最高为1.0, 最低为0, 默认分数为0.25
device id	int	否	支持多显卡功能, 用于设定使用哪一个显卡设备进行推理, 默认0, 选择默认的显卡, 设为-1, 选择cpu进行推理
label mask	wstring array	否	用于标签过滤, SDK会自动过滤设定的标签, 支持中英文标签, 默认为空
contour size	int	否	分割模型(my01s_p, mm01_p, ss01)输出图像目标轮廓点的数目, 默认INT_MAX, 所能提取轮廓的最大长度值
num threads	int	否	并行区域使用多少个线程(数值范围: 1~CPU最大线程数目), 数值大会提高计算速度, 同时也会提高CPU使用率, 默认1
log	bool	否	调试日志, 设为true, 会在执行文件目录dlia_log文件夹内产生调用log文件, 默认否, 注: 正常量产软件务必设为false, 否则会影响执行效率以及占用磁盘空间
text concatenation	object	否	文本拼接, 自动将识别到字符拼接成字符串输出
-eps	float	/	字符拼接像素距离, 两个字符相距小于此数值, 拼接在一起, 反之不拼接, 默认500像素
-minPts	int	/	文本拼接之后字符串至少包含字符的数目, 默认1

config参数	类型	是否必须	备注
-line spacing	float	/	文本行间距，小于此数值，按同行文本处理，反之按另行处理，默认50像素
overlap	bool	否	是否允许检测目标叠加，若设为true，目标叠加的程度通过iou参数进行设定，默认为否
iou threshold	float	否	当overlap设为true时，用来表征检测目标之间叠加程度，数值处于0~1.0之间，值越高，覆盖程度越大，默认是0.35
heatmap	bool	否	输出热力图(base64格式数据)，适用于ss01模型，设为true，输出热力图，默认为false，注: 若需要显示热力图检测异常结果，需要先将热力图按照自定义权重和原始图片进行叠加，然后再显示
detection contour	bool	否	适用于ss01模型，默认true，输出多边形异常检测区域轮廓点坐标、几何中心坐标、面积等，标签固定为"anomaly"
detection contour area criterion	float	否	检测异常轮廓面积过滤，当"detection contour"设为true，检测到的异常轮廓面积小于阈值，则过滤，默认为0像素
qrcode	bool	否	二维码检测，支持Aztec, DataMatrix, MaxiCode, PDF417, QRCode, MicroQRCode, RMQRCode格式
barcode	bool	否	条码检测，支持Codabar, Code39, Code93, Code128, EAN8, EAN13, ITF, DataBar, DataBarExpanded, DataBarLimited, DXFilmEdge, UPCA, UPCE格式
filter	object	否	过滤器，适用于mm01/my01模型，如果计算长度、面积、分数在设定范围内，则不过滤，返回结果，反之过滤，不返回结果，数据及说明定义如下

过滤配置数据结构说明:

```

"filter" : {
  "label_a": {"length_min": len_min,
              "length_max": len_max,
              "area_min": area_min,
              "area_max": area_max,
              "score_min": score_min},

  "label_b": {"length_min": len_min,
              "length_max": len_max,
              "area_min": area_min,
              "area_max": area_max,
              "score_min": score_min},

  "label_c": {"length_min": len_min,
              "length_max": len_max,
              "area_min": area_min,
              "area_max": area_max,
              "score_min": score_min}
}

```

1. label_a, label_b, label_c 为标签名称;
2. length_min, length_max 为长度最小和最大值键，对应键值len_min, len_max 为float类型数值;
3. area_min, area_max 为面积最小和面积最大值键，对应键值area_min, area_max 为float类型数

值;

4. score_min 为最低分数值, 低于此分数值改标签会被过滤, 注意, 由于全局分数confThreshold优先级高, 当且仅当score_min高于全局confThreshold分数值时, 才会起作用。

```
/*
函数 DLIA Runtime 测试(推理, 仅支持3通道, 且颜色空间为BGR的Mat)
输入:
    dlia_runtime_id : 实例ID, int类型
    image_name      : 图像名称, 字符串数据类型
    image           : OpenCV Mat格式的图像(3通道, 颜色空间为BGR)
输出:
    result          : 返回识别结果buffer, char类型数组(JSON格式化字符串)
    res_len         : 识别结果的数据长度
返回: 0 - 执行成功, 其它 - 失败, 详细请参考SDK文档中错误代码说明
*/
int dlia_runtime_infer_cv(int dlia_runtime_id, const char* image_name, cv::Mat &
image, char* result, uint64_t * res_len);

/*
函数 DLIA Runtime 测试(推理)
输入:
    dlia_runtime_id : 实例ID, int类型
    image_name      : 图像名称, 字符串数据类型
    image           : 图像(已经编码encode), 支持jpg/jpeg/png/bmp/gif/tiff等格式
    image_size      : 图像大小, 单位字节
输出:
    result          : 返回识别结果buffer, char类型数组(JSON格式化字符串)
    res_len         : 识别结果的数据长度
返回: 0 - 执行成功, 其它 - 失败, 详细请参考SDK文档中错误代码说明
*/
int dlia_runtime_infer_im(int dlia_runtime_id, const char* image_name, const char*
image, uint64_t image_size, char* result, uint64_t* res_len);

/*
函数 DLIA Runtime 测试(推理,仅支持Depth为1, 即每个像素pixel精度为8bit unsigned char的图像)
输入:
    dlia_runtime_id : 实例ID, int类型
    image_name      : 图像名称, 字符串数据类型
    image           : 图像原始数据buffer
    width           : 图像宽, 单位像素
    height          : 图像高, 单位像素
    rgb             : 颜色空间, bool类型, true为RGB/RGBA,, false为BGR/BGRA
    channel         : 图像通道, int类型, 例如grayscale图像为1, RGB图像为3, RGBA图像为4等
输出:
    result          : 返回识别结果buffer, char类型数组(JSON格式化字符串)
    res_len         : 识别结果的数据长度
返回: 0 - 执行成功, 其它 - 失败, 详细请参考SDK文档中错误代码说明
*/
int dlia_runtime_infer_im_rawdata(int dlia_runtime_id, const char* image_name, const
char* image, int width, int height, bool rgb, int channel, char* result, uint64_t*
res_len);

/*
函数 DLIA Runtime 释放资源
输入:
    dlia_runtime_id : 实例ID, int类型;
```

```
    返回: 0 - 执行成功, 其它 - 失败, 详细请参考错误代码
*/
int dlia_runtime_release(int dlia_runtime_id);

/*
    函数 DLIA Runtime 提取模型信息
    输入:
        _model_path      : 模型文件的绝对路径;
        _model_info     : 模型信息, char类型数组(JSON格式化字符串);
    返回: 0 - 执行成功, 其它 - 失败, 详细请参考SDK文档中错误代码说明
*/
int dlia_runtime_retrieve_model_info(const char* _model_path, char* _model_info);
```

2.9. DLIA-SDK 返回数据说明

2.9.1 基本格式

参数名	数据类型	备注
error	string	返回"0"-成功, 其它失败, 详细代码参考 附录
data	object	数据
-image	string	测试图片名称, 保留显示"null"。
-results	object	结果数据, json格式, 以标签名称为键, 键值为字符串数组, 字符串数组内每条字符串代表一个缺陷, 里面包含: 缺陷区域形状多边形polygon或矩形rectangle, 缺陷位置坐标、面积、置信度等信息。
processing times	string	处理时间, 单位毫秒

- 返回JSON数据举例-目标检测/实例分割

```
{
  "error" : "0",
  "data" : {
    "image": "abc.jpg",
    "results": [
      {"label_A" : ["polygon, [[1570,157],[1566,196],[1570,214],[1578,225],
[1572,235],[1569,260],[1571,299],[1605,300],[1621,300],[1651,299],[1643,284],
[1619,286],[1612,266],[1602,256],[1613,252],[1621,234],[1624,210],[1616,198],
[1609,187],[1598,178]] | 6263 | [1517,384] | 0.973255"}},
      {"label_B" : ["rectangle, [[231, 21], [564, 185]] | 0.95", "rectangle,
[[65, 211], [164, 485]] | 0.99"]}
    ]
  },
  "processing times": 123
}
```

输入图片是"abc.jpg", 预测标签有"label_A"和"label_B", 其中"label_A"只有一个多边形缺陷, "label_B"则包含2个矩形缺陷, 总处理时间为 123 ms。

字符串数据解析举例:

```
"rectangle, [[231, 21], [564, 185]] | 0.95"
```

矩形区域, [231, 21]左上角坐标, [564, 185]右下角坐标值, 置信度0.95;

```
"polygon, [[1570,157],[1566,196],[1570,214],[1578,225],[1572,235],[1569,260],[1571,299],
[1605,300],[1621,300],[1651,299],[1643,284],[1619,286],[1612,266],[1602,256],[1613,252],
[1621,234],[1624,210],[1616,198],[1609,187],[1598,178]] | 6263 | [1517,384] | 0.97325"
```

多边形区域, [1570,157],[1566,196], ... ,[1598,178]轮廓点坐标值, 6263 为面积(像素单位), [1517,384] 轮廓中心点坐标值, 0.97325为置信度。

- 返回JSON数据举例-图像分类

```
{
  "error" : "0",
  "data" : {
    "image": "123",
    "results": [
      {"label_A" : "0.85"}
    ]
  },
  "processing times" : "80"
}
```

输入图片是"123", 预测分类是"label_A", 置信度0.85, 处理时间 80 ms。

- 返回JSON数据举例-没有检测到任何标签/类别

```
{
  "error" : "0",
  "data" : {
    "image": "1212.png",
    "results": null
  },
  "processing times" : "100"
}
```

输入图片是"1212.png", 没有识别到任何标签或类别, 处理时间100 ms。

2.9.2 二维码/条码格式

模型初始化中使能qrcode和barcode, 如果检测到二维码或条码, 在返回JSON数据里面包括键["__USER_CODE_" / "MatrixCodes"]和["__USER_CODE_" / "LinearCodes"], 其键值为二维码/条码识别结果, 包括"条码/二维码格式"、"位置坐标"、"旋转角度", "文本内容"。

```
{
  "__USER_CODE__" : [
    {
      "MatrixCodes": [
        {
          "Format" : "QRcode",
          "Position" : {"point0": [2912, 2582], "point1": [3095, 2579], "point2": [3090, 27591], "point3": [2908, 2763]},
          "Rotation" : -1,
          "Text" : "10200020094921"
        }
      ]
    },
    {
      "LinearCodes": [
        {
          "Format" : "ITF",
          "Position" : {"point0": [2904, 2172], "point1": [2904, 2472], "point2": [2847, 24741], "point3": [247, 2173]},
        }
      ]
    }
  ]
}
```

```
    "Rotation" : 90,  
    "Text" : "1693528412007"  
  }  
]  
}
```



2.10. 软件激活

在安装软件时，会自动安装virbox用户工具软件(若没有安装或者安装异常，用户也可去virbox官网 <https://lm.virbox.com/tools.html> 下载安装)，软件激活可以使用硬加密狗或者软加密狗，两种任选其一，其中软加密狗又有在线和离线激活两种方式。

2.10.1 使用硬加密狗

将硬加密狗插入计算机USB口即可。

2.10.2 使用软加密狗

点击左下角计算机Windows开始图标，输入"virbox"或者右下角状态栏中单击virbox软件图标打开virbox用户工具软件。

- 在线激活

在计算机连接外网情况下，激活软件操作流程如下图，输入厂家提供的授权码即可激活。



注意在激活过程中，计算机需要保持外网连接状态。

- 离线激活

在计算机无法连接外网情况下，需要离线激活软件，具体操作步骤为：先使用virbox导出离线绑定的c2d文件，然后发给厂家，再使用厂家授权之后d2c文件，在virbox软件导入即可完成软件激活。

区域: 中国

云账号 +

软锁

本地软锁

云/软锁

硬件锁

我的软件

服务设置

诊断修复

高级

版本: v2.5.0.60698(2.5.0.60698) 常见问题

许可信息

账号/主机:	本地软锁
主机IP:	192.168.1.104
许可:	0 套
正常:	0 套
已过期:	0 套

刷新

生成离线绑定c2d文件

导入离线绑定d2c文件

授权码在线激活 [授权码网页兑换](#)

2.11. DLIA-SDK示例代码与调试

2.11.1 例程

DLIA SDK包含C#/C++/Java/LabVIEW/Python等项目例程，涉及"单线程单显卡"，"多线程单显卡"、"多线程多显卡"以及"使用CPU推理"，详见SDK Samples文件夹内容。

2.11.2 单步调试

集成SDK过程中，如需在线单步调试，使用依赖dll的Debug版本即可，正式发布产品，请使用执行效率更高的Release版本。

Release版本dll	Debug版本dll	适用模型
dli-runtime_x64.dll	dli-runtime_x64_debug.dll	mm01/mm01_p/my01/my01s_p/my01s/mx01
dli-runtime_mst_x64.dll	dli-runtime_mst_x64_debug.dll	mst
dli-runtime_ss01_x64.dll	dli-runtime_ss01_x64_debug.dll	ss01

2.12. DLIA-SDK HTTP API

DLIA SDK为扩展应用场景，提供了一个HTTP接口程序(dlia_httpsrv.exe，如需要Linux版本，请联系厂家)，位于SDK安装目录下http api文件夹内，API详细说明如下：

```
/*
接口格式：http://{ip}:{port}/{api}，其中ip默认为本机ip("127.0.0.1")，端口号默认"5009"，如需要
修改，按以下格式启动程序：
dlia_httpsrv.exe [options] ...
options:
  -h, --host      host ip (string [=127.0.0.1])
  -p, --port      port number(1-65535) (int [=5009])
  -n, --nthread   number of threads(1-30) (int [=8])
  -?, --help      print this message
*/
```

api	接口	描述
version	http://{ip}:{port}/version	查询http版本
modif	http://{ip}:{port}/modif	查询模型信息，获取模型内标签、模型类型等等
init	http://{ip}:{port}/init	模型初始化
infer	http://{ip}:{port}/infer	推理测试

2.12.1 查询http版本

- 请求: /version
- 返回:

```
{
  "DLIA HTTPS Server": "1.1.0",
  "DLIA SDK": "1.3.0.8"
}
```

参数名	数据类型	备注
DLIA HTTPS Server	string	http server 版本
DLIA SDK	string	SDK 版本

2.12.2 查询模型信息

- 请求: /modif
- 参数:

```
{
  "model" : "C:\\Program Files\\DLIA Runtime\\x64\\Samples\\model\\模型model_cpu.bin"
}
```

"model": 模型绝对路径地址。

返回:

```
{"tutorial-mm01": {"annotation labels": ["plating", "discoloration", "contamination"], "chunked": true, "classification": "0", "device id": -1, "end time": "2025-05-24 17:36:05", "model": "mm01", "model format": "torchscript", "resize": [1024, 1024], "size": 176589655, "start time": "2025-05-24 17:03:56", "status": "Finished"}}
```

"tutorial-mm01": 项目名称。

"annotation labels": 标签列表。

"device id": GPU或CPU模型, -1为CPU, 0~n对应GPU0~GPU_n显卡模型。

"model": 模型类型。

"start time": 训练起始时间。

"end time": 训练结束时间。

2.12.3 模型初始化

- 请求: /init
- 参数:

```
{
  "0": {
    "model": "C:\\Program Files\\DLIA Runtime\\x64\\Samples\\model\\model_gpu0.bin",
    "config": {
      "show result image": false,
      "device id": 0
    }
  }
}
```

"0": dlia runtime id, 键值为对应的配置参数。

"model": 模型绝对路径地址

"config": SDK模型初始化模型配置参数, 配置项目及含义见[定义](#)。

返回:

```
{
  "error": "0",
  "results": {
    "0": 0
  }
}
```

"error": "0"为成功，非"0"字符串对应不同的错误文本。
{ "0": "0" }: runtime id 0模型初始化成功，任何非0值都是初始化失败，可查看[错误代码](#)。

2.12.4 图片推理测试

- 请求: /infer
- 参数:

```
{
  "runtime id" : 0,
  "image" : "/9j/4AAQSkZJRgABAQA....省略...",
  "image name": "ocr500.jpg",
  "inference results image" : true
}
```

参数名	数据类型	备注
runtime id	int	对应模型初始化时runtime id
image	string	图片base64数据
image name	string	图片名称
inference results image	bool	是否返回检测结果图像，若设为true，则返回数据中包括"inference results image"键，其键值为base64检测结果图片数据

返回:

```
{
  "data": {
    "image": "ocr500.jpg",
    "results": [{"plating discoloration": [{"rectangle", [[1262.0,1678.0], [1503.0,1790.0]] | 0.9999", "rectangle", [[1515.0,266.0], [1751.0,382.0]] | 0.9998"}]}], "error": "0", "processing times": 4032,
    "inference results image": "/9j/4AAQSkZJRgABAQAAAQAB.....省略...",
    "inference results image height": 2048,
    "inference results image width": 3072
  }
}
```

参数名	数据类型	备注
data	object	参考 返回数据说明
inference results image	string	检测结果图像base64
inference results image height	int	检测结果图像高
inference results image width	int	检测结果的图像宽

- 注：只有在/infer中"inference results image"键值设为true时，推理结果才返回"inference results image"、"inference results image height"、"inference results image width"键和对应的键值。

2.12.5 Web测试

http server提供了一个web测试接口，地址: `http://{ip}:{port}`，默认: `http://127.0.0.1:5009`，配置初始化参数、选择模型初始化之后，上传图片即可显示推理结果。

第一步：模型初始化

配置模型初始化参数，并选择模型文件后点击提交:

模型初始化参数	<pre>{ "log": false, "show result": true, "image": false, "confThreshold": 0.25, "device": "id": 0, "barcode": false, "qrcode": false }</pre>	
模型文件	<input type="button" value="选择文件"/> 未选择任何文件	<input type="button" value="提交"/>

3. 附录1-错误代码

返回码	描述	解决办法
0	操作成功	无
16777216 (0x01000000)	配置文件错误	检查配置文件格式及路径
16777217 (0x01000001)	模型文件错误	检查模型文件及路径
16777218 (0x01000002)	模型格式不支持	确认模型格式或联系经销商
16777219 (0x01000003)	模型类型不支持	确认模型类型或联系经销商
16777220 (0x01000004)	模型提取失败	检查模型文件完整性
16777221 (0x01000005)	实例化失败	检查输入参数
16777222 (0x01000006)	实例未初始化	先初始化实例再使用
16777223 (0x01000007)	参数无效	检查输入参数格式和值
16777224 (0x01000008)	模型没有授权	联系经销商获取授权
16777225 (0x01000009)	图像格式不支持	支持格式: jpg/jpeg/png/bmp/tiff
16777226 (0x0100000A)	模型写入失败	检查磁盘权限和空间
33554430 (0x01FFFFFFE)	程序异常	重启应用, 问题持续请联系支持
33554431 (0x01FFFFFFF)	推理繁忙	等待当前推理完成后重试

4. 附录2-JSON数据提取示例代码

C++ JSON数据处理推荐使用: [nlohmann json](#), 详见例程。

- 示例代码

```
void inference_retrieve(const char* results) {
    if (results == nullptr) return;

    json infer_rst = json::parse(results, nullptr, false);

    if (!infer_rst.is_discarded()) {
        cout << "error code: " << infer_rst["error"] << endl;
        cout << "processing time: " << infer_rst["processing times"] << " ms" << endl;
        cout << "image id: " << infer_rst["data"]["image"] << endl;

        if (infer_rst["data"]["results"].is_null() == false) {
            //rs : [{"carton": ["rectangle, [[128,1536], [3546,2931]] | 0.875766" }]
            json rs = infer_rst["data"]["results"];
            std::cout << "rs: " << rs.dump() << endl;
            int s = (int)rs.size();
            std::cout << "s: " << s << endl;
            string score = "";
            for (int i = 0; i < s; i++) { //遍历
                json crops_json = rs[i].items().begin().value();
                std::cout << rs[i].items().begin().key() << " Detected." << endl; //检测
                std::cout << crops_json.dump() << endl;
                if (crops_json.is_array()) {
                    for (int j = 0; j < crops_json.size(); j++) { //遍历
                        if(crops_json[j].is_string()){
                            string ss = crops_json[j];

                            if (ss.find("rectangle") != string::npos) { //是矩形
                                //获取分数
                                {
                                    int found = (int)ss.find("|");
                                    if (found != string::npos) {
                                        score = ss.substr((int64_t)found + 1, ss.length()
                                        - found);

                                        //去掉头尾空格
                                        if (!score.empty()) {
                                            score.erase(0, score.find_first_not_of(" "));
                                            score.erase(score.find_last_not_of(" ") + 1);
                                        }
                                        cout << "scores: " << score << endl;
                                    }
                                }
                            }
                        }
                    }
                }
                //获取坐标
                vector<string> coord;
                item_split_coordinate(ss, coord);
            }
        }
    }
}
```

```

cv::Point2i tl(stoi(coord[0]), stoi(coord[1])); //top left
cv::Point2i br(stoi(coord[2]), stoi(coord[3])); //bottom
right

//打印左上角和右下角的图像坐标
cout << "coordinate: " << std::dec << tl.x << ", " << tl.y
<< ", " << br.x << ", " << br.y << endl;

}
else if (ss.find("polygon") != string::npos) { //是多边形目标
/*
["polygon, [[745,162],[721,180],[696,205],[668,231],
[643,252],[622,270],[600,293],[582,319],[564,344],[543,373],[526,416],
[526,702],[545,747],[575,781],[594,810],[611,831],
[633,852],[661,880],[693,901],[722,924],[771,949],[1102,937],[1137,917],
[1157,899],[1176,882],[1199,864],[1221,840],
[1239,820],[1256,796],[1273,775],[1295,754],[1314,719],[1300,383],[1281,348],
[1263,329],[1239,293],[1220,273],[1202,255],
[1185,237],[1159,219]] | 535030 | [919,558] | 0.999934"]
*/
//获取分数
string score = "";
{
    int found = (int)ss.find_last_of("|");
    if (found != string::npos) {
        score = ss.substr((int64_t)found + 1, ss.length()
- found);

        //去掉头尾空格
        if (!score.empty()) {
            score.erase(0, score.find_first_not_of(" "));
            score.erase(score.find_last_not_of(" ") + 1);
        }
        cout << "scores: " << score << endl;
    }
}

//获取坐标
vector<cv::Point> coord;
cv::Point center;
item_split_polygon_coordinate(ss, coord, center);

cout << "coordinate: ";
for (auto it : coord) {
    cout << std::dec << it.x << " " << it.y << " ";
}
cout << endl;

cout << "center: " << std::dec << center.x << ", " <<
center.y << endl;

}
else { //不支持
//
}
}
}
else { //对于分类, 此处为单一数值—分数

```

```
        score = to_string(crops_json);
        cout << "scores: " << score << endl;
    }
}
else {
    cout << "No Label Detected." << endl;
}
}
```

5. 附录3 DLIA-SDK使用常见问题

- 控制台中文标签显示乱码

C++代码可以选择执行系统指令

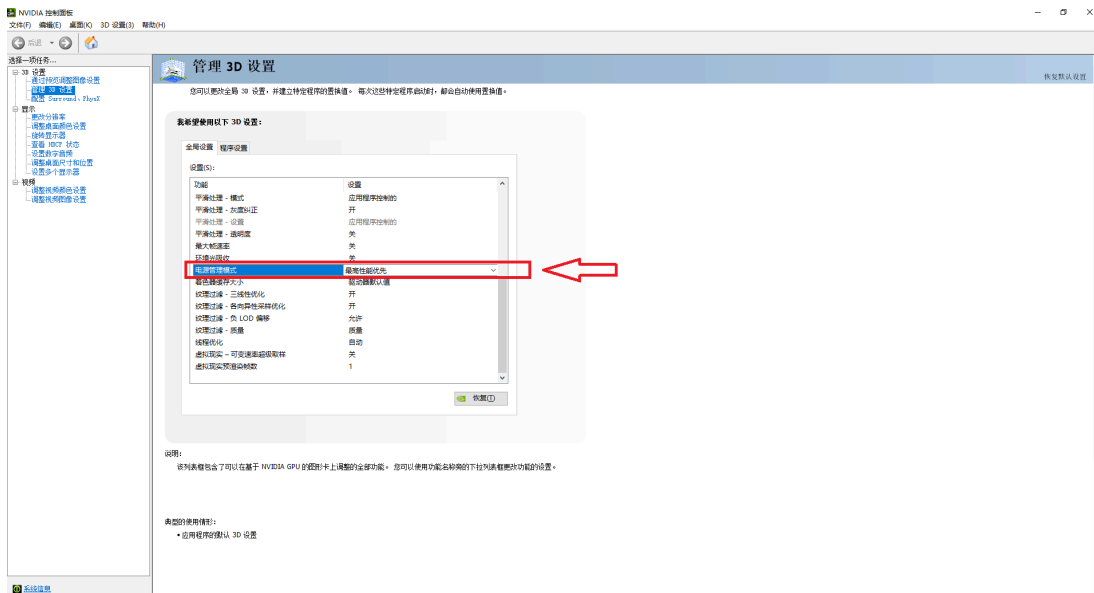
```
system("chcp 65001");
```

或者修改计算机语言配置：



- 显卡高性能模式

默认情况下, NVIDIA显卡在一段时间没有运行之后, 会进入低功耗模式, 计算性能下降, 而在某些工业应用可能需要显卡一直处于高计算性能状态, 此时可在NVIDIA控制面板中进行工作模式配置, 切换到高性能模式。



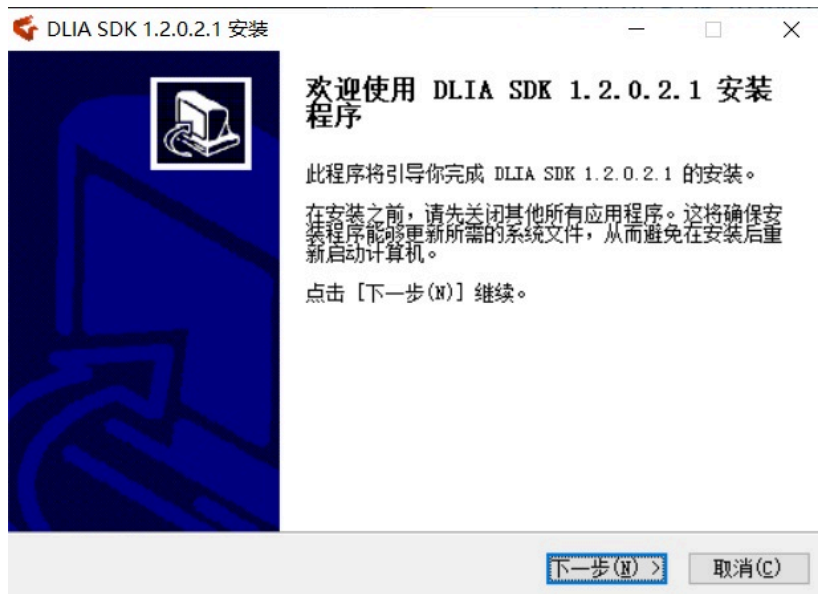
- 杀毒软件识别成病毒并误杀

DLIA SDK软件使用了第三方工具进行IP保护和防止破解, 杀毒软件可能会被误认为是病毒软件, 解决办法:

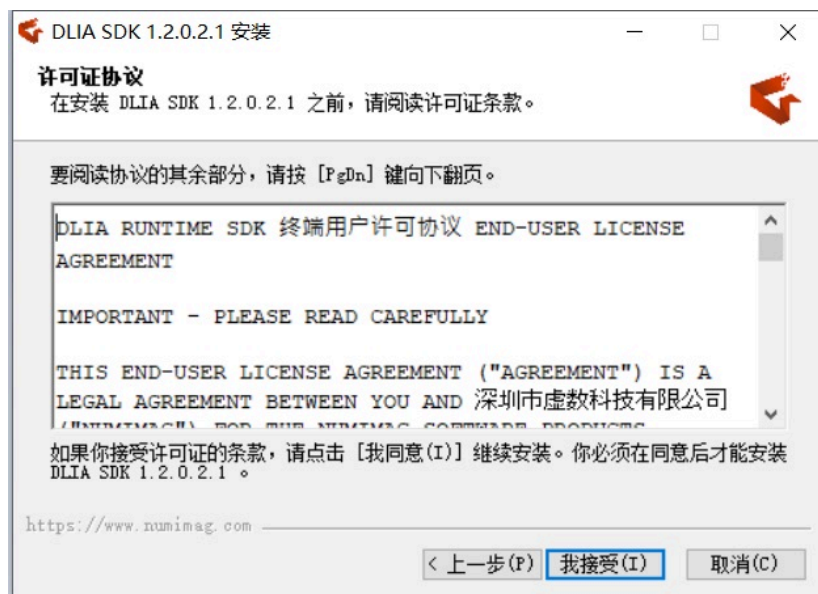
- a. 如果已经删除, 需要到杀毒软件里面进行恢复。
- b. 添加安装目录到杀毒软件白名单里面, 参考链接:<https://h.virbox.com/vbp/docs/faqs/主流安全软件免杀指南/>。

6. 附录4 DLIA-SDK安装步骤

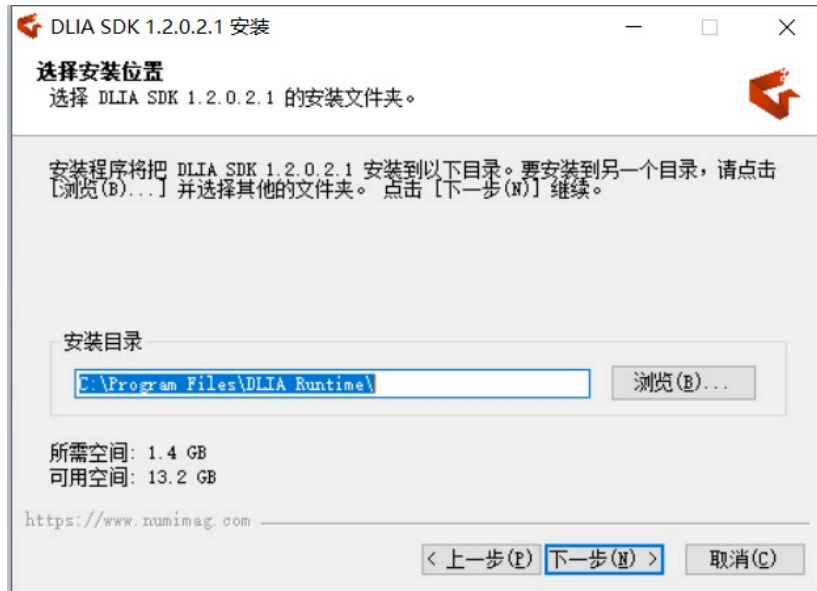
- 执行DLIA SDK安装程序Setup.exe，进入欢迎安装页面，点击下一步。



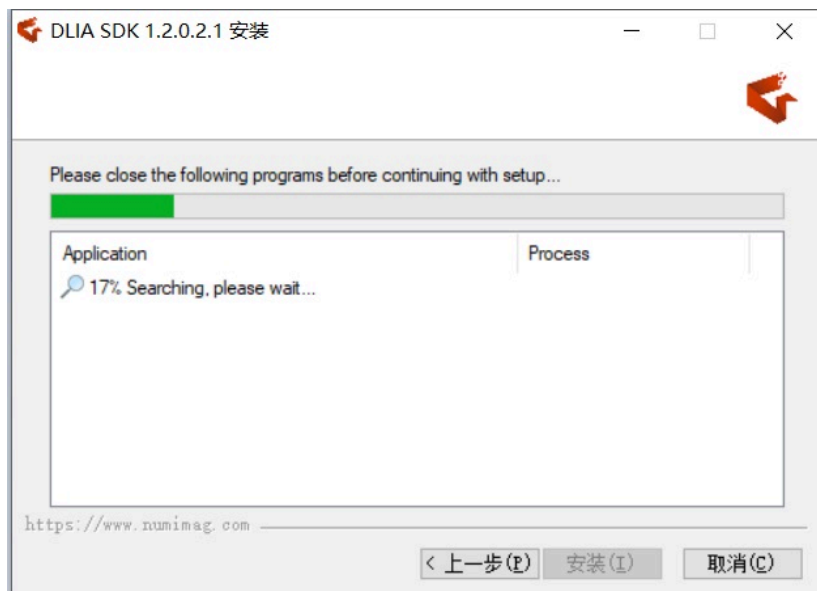
- 界面上点击按钮"我接受"，接受终端用户协议。



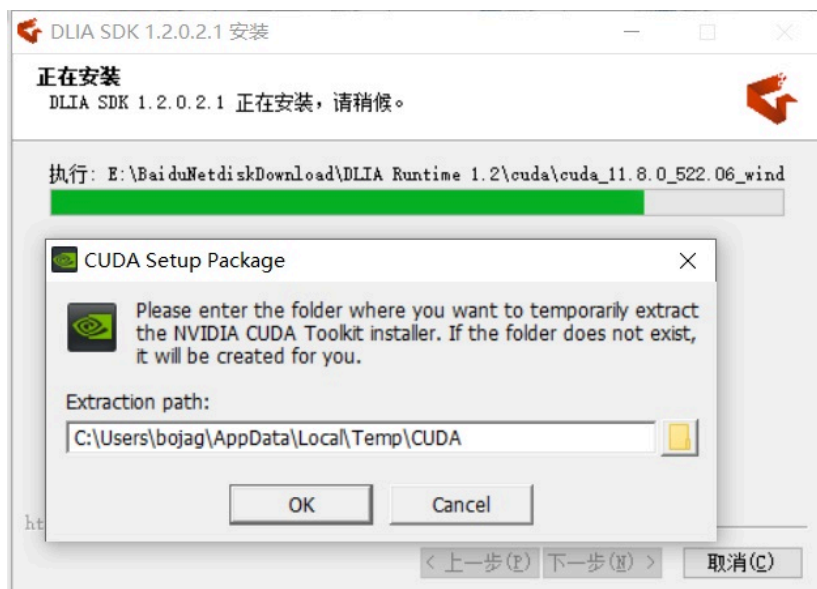
- 选择安装路径，点击下一步。



- 检查安装路径下是否有文件被占用，若被占用，需要先关闭占用程序



- 解压并安装CUDA步骤，除非非常确定已经安装此版本CUDA，可以选择跳过(点击按键Cancel)，不然就执行安装(点击按键OK)



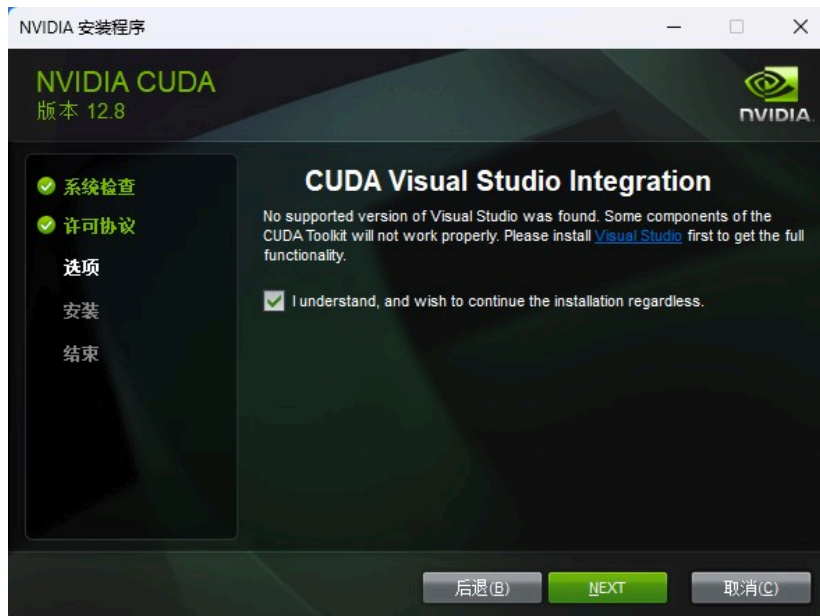
- 同意NVIDIA的终端用户许可协议，界面上点击按钮"同意并继续"。



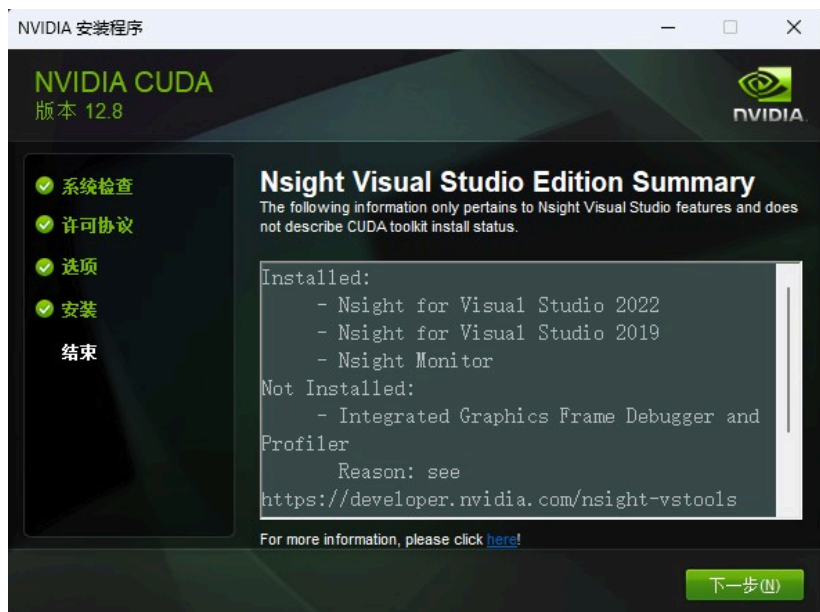
- 选择默认精简方式安装，点击下一步



- CUDA没有找到适合visual studio版本，勾选"了解"，点击下一步



- 一段时间之后，界面会提示CUDA已经或没有安装的组件列表，如果没有出错，点击下一步，如果提示出错，可根据出错信息，排查错误之后，再次安装。



- 安装结束最后一步，取消选择创建快捷方式和启动Experience，点击关闭。



- 返回到主安装程序，提示安装结束，界面上点击按钮"完成"。



7. 附录5 DLIA-SDK卸载

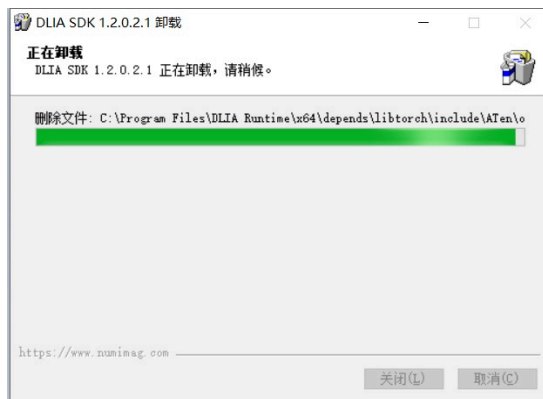
- 进入Windows应用程序管理，选择DLIA SDK，点击"卸载"按钮。



- 在卸载确认框中，选择"是"，执行卸载操作。



- DLIA SDK 开始卸载。



- 一段时间之后，提示框中点击"确定"，完成卸载。

