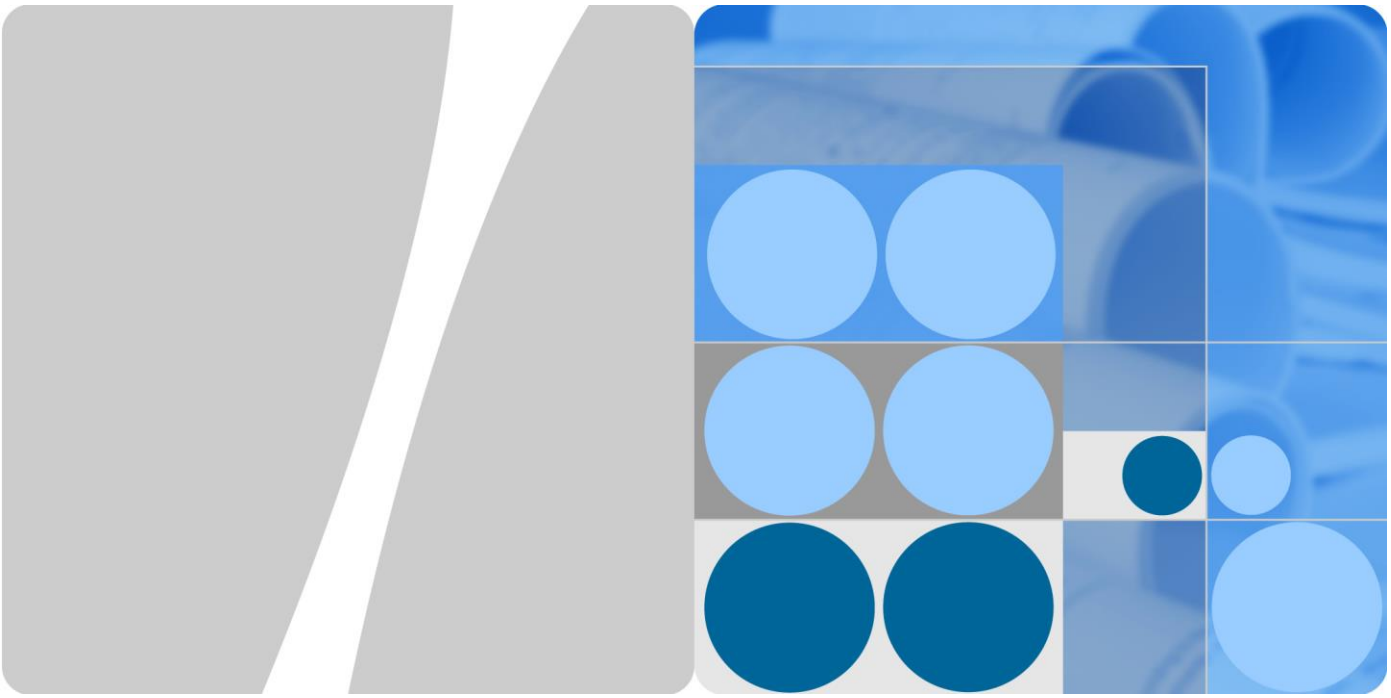


HiAI DDK V320

模型推理集成指导

文档版本 04

发布日期 2020-02-29

华为技术有限公司

华为 HiAI 申请方式

发送申请邮件到邮箱：developer@huawei.com

邮件名称：HUAWEI HiAI+公司名称+产品名称

邮件正文：合作公司+联系人+联系方式+联系邮箱

我们将在收到邮件的 5 个工作日内邮件给您反馈结果，请您注意查收。

官网地址 https://developer.huawei.com/consumer/cn/

前 言

概述

本文提供对 HiAI DDK V320 模型推理集成操作的说明，包括操作步骤，API 和错误码。

本文与以下文档配套使用：

- 华为 HiAI_DDK_V320_快速入门
- 华为 HiAI_DDK_V320_算子规格说明

修改记录

修改记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

日期	修订版本	修改描述
2020-02-29	04	增加模型 Tensor 创建类接口
2019-12-31	03	新增 V320 版本内容
2019-11-22	02	更新模型集成代码
2019-09-04	01	新增 V310 版本内容

目 录

插图目录

表格目录

1 总体说明

由于目前 AI 智能程序一般可执行图片识别分类等任务，且此类程序通常需要借助预先训练好的模型，使用模型进行推理得出结果。模型推理集成指导用户使用华为 DDK 包进行 AI 程序的构建，在构建 APP 的过程中打包集成模型。

1.1 依赖条件

- 准备 Ubuntu 16.04 开发环境（Win10、macOS）
- 准备 Android Studio 开发应用
- 准备训练好的 Caffe 或 TensorFlow 模型
- 准备搭载 Kirin 平台的设备

1.2 支持的算子

请参见《华为 HiAI_DDK_V320_算子规格说明》。

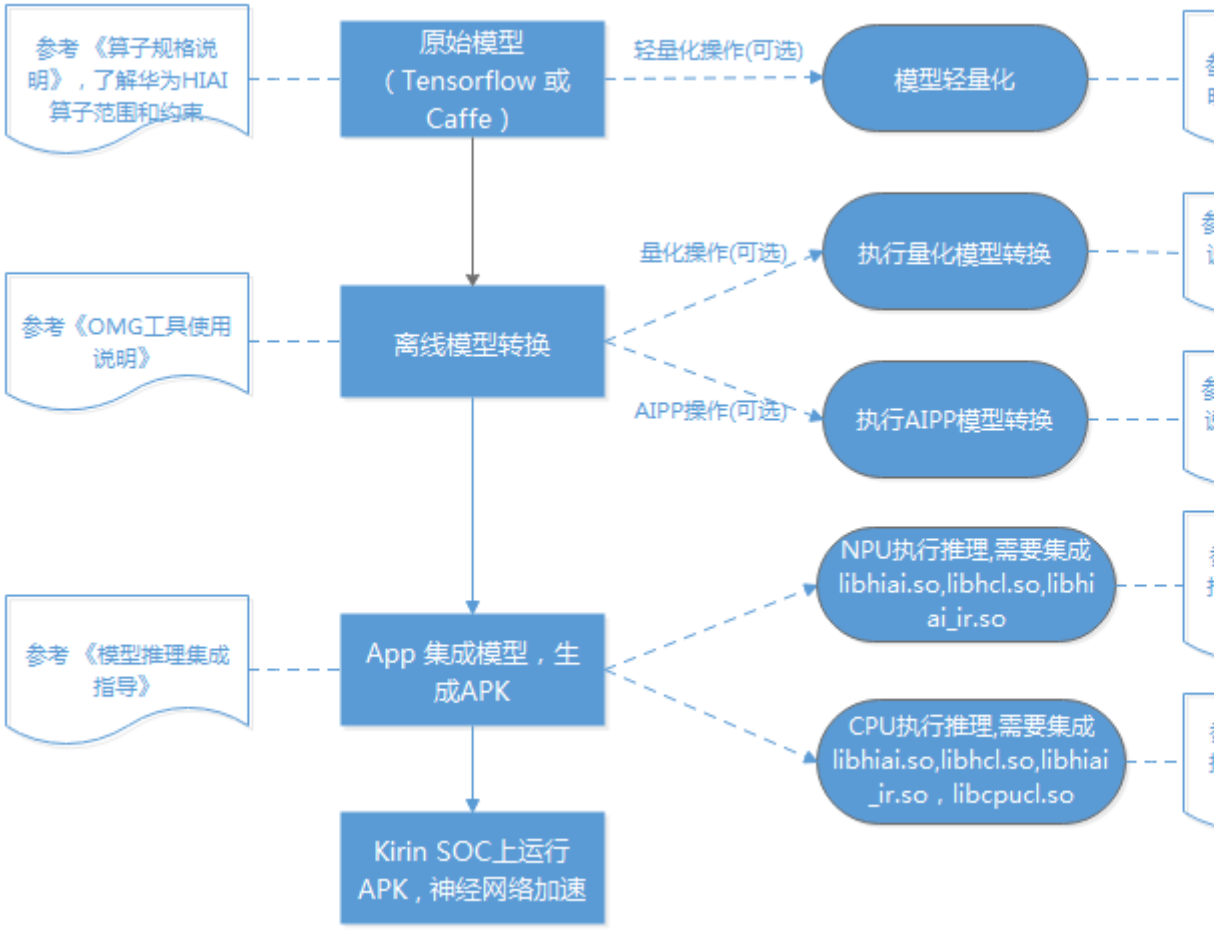
1.3 流程说明

下图介绍了 App 使用 HiAI DDK V320 的集成流程，其中 AIPP（Artificial Intelligence Pre-processing）、量化为可选步骤。

说明

下图中的参考文档请根据实际应用版本进行参考。

图1-1 NPU/CPU 集成操作流程图



原始模型轻量化

针对原始模型（Tensorflow 或 Caffe）进行深度的模型优化，可以帮助用户自动完成模型轻量化，达到减少模型体积以及加快模型推理速度的目的。目前支持无训练模式和重训练模式。轻量化操作可参考《轻量化工具使用说明书》。

离线模型转换

离线模型转换需要将 Caffe 或者 TensorFlow 模型转换为 HiAI 平台支持的模型格式，并可以按需进行 AIPP 操作、量化操作，使用场景及方法如下：

- AIPP 操作

  AIPP 用于在硬件上完成图像预处理，包括改变图像尺寸、色域转换（转换图像格式）、减均值/乘系数（改变图像像素），运用后可避免重新训练匹配推理计算平台需要的数据格式，仅仅通过 AIPP 参数配置或者在软件层面上调用 AIPP 接口即可完成适配，同时由于硬件专用，可以获得较好的推理性能收益，具体操作可参考《华为 HiAI_DDK_V320_OMG 工具使用说明》中 AIPP 模型转换以及配置的操作指导。
- 量化操作

量化是一种可以把 fp32 模型转化为低 bit 模型的操作，以节约网络存储空间、降低传输时延以及提高运算执行效率，量化操作可参考《华为 HiAI_DDK_V320_OMG 工具使用说明》中量化模型转换的操作指导。

APP 集成

APP 集成流程包含模型预处理、加载模型、运行模型、模型后处理。

- NPU 场景下，APP 需要在模型预处理过中集成 libhiai.so、libhcl.so、libhiai_ir.so，编译 APK 后可在 NPU 上执行推理，参考《华为 HiAI_DDK_V320_模型推理集成指导》。
- CPU 场景下，APP 需要在模型预处理过中集成 libhiai.so、libhcl.so、libhiai_ir.so 以及 libcpucl.so，编译 APK 后可在 CPU 上执行推理，参考《华为 HiAI_DDK_V320_模型推理集成指导》。

2 APP 集成

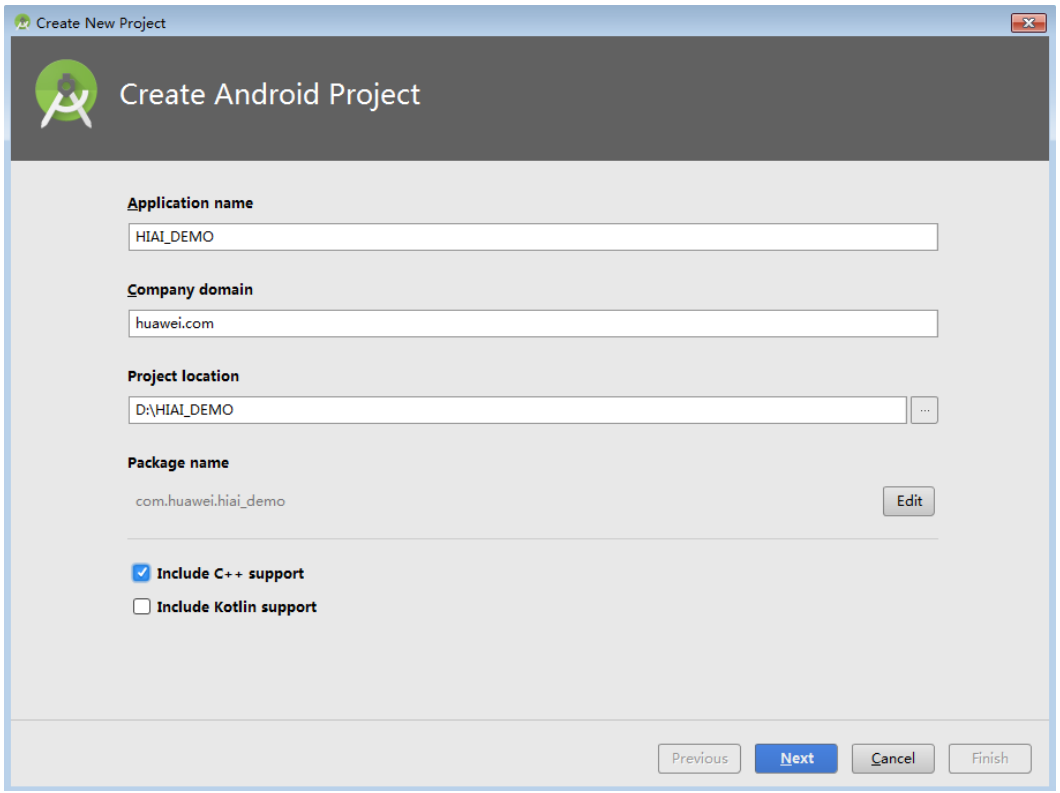
本章以 Caffe Squeezenet 模型集成为例，说明 APP 集成操作过程。

说明

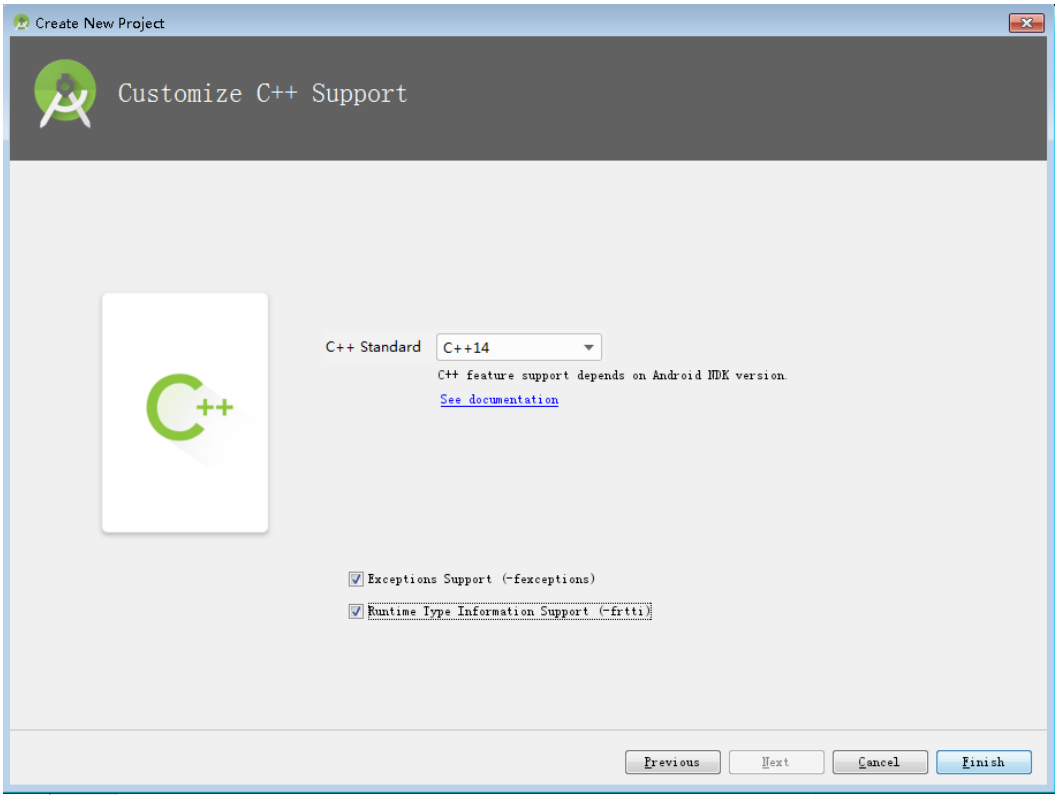
DDK V320 新增支持 CPU 场景，与 NPU 场景的操作差异参考 2.2.3 拷贝 SDK so 到资源库

2.1 创建项目

1. 创建 Android Studio 项目，勾选“Include C++ support”



2. C++ Standard 选择 C++14，勾选 Exceptions Support(-fexceptions)，勾选 Runtime Type Information Support(-frtti)。



2.2 编译 JNI

推理模型编译 JNI 时，APK 打包需要打包 libhiai.so 和 libhiaijni.so

2.2.1 编写 Android.mk 文件

编写 JNI 接口实现代码 classify_jni.cpp、classify_async_jni.cpp、buildmodel.cpp，并将这三个文件拷贝至 DDK 的 jni 目录，编写 Android.mk 文件如下：

```
LOCAL_PATH := $(call my-dir)
DDK_LIB_PATH := $(LOCAL_PATH)/../../../libs/$(TARGET_ARCH_ABI)

include $(CLEAR_VARS)
LOCAL_MODULE    := hiai
LOCAL_SRC_FILES := $(DDK_LIB_PATH)/libhiai.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE := hiaijni
LOCAL_SRC_FILES := \
   classify_jni.cpp \
   classify_async_jni.cpp \
   buildmodel.cpp
```



```
LOCAL_SHARED_LIBRARIES := hiai
LOCAL_LDFLAGS := -L$(DDK_LIB_PATH)
LOCAL_LDLIBS += \
    -llog \
    -landroid

CPPFLAGS=-stdlib=libstdc++ LDLIBS=-lstdc++
LOCAL_CFLAGS += -std=c++14

include $(BUILD_SHARED_LIBRARY)
```

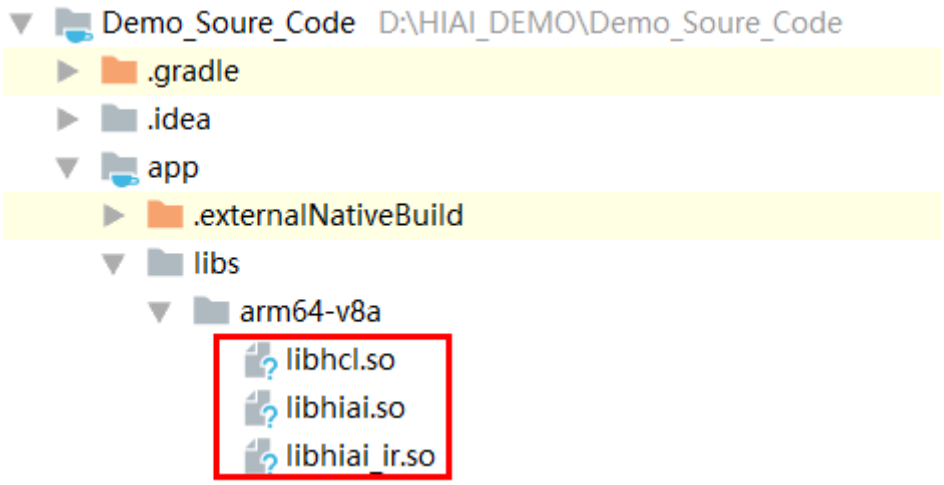
2.2.2 编写 Application.mk 文件

在 jni 目录下，编写 Application.mk 文件如下：

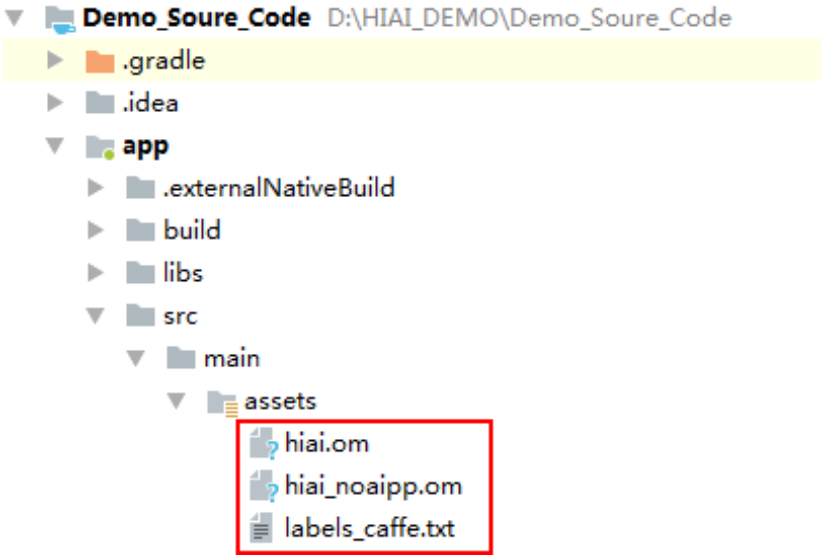
```
APP_ABI := arm64-v8a
APP_STL := c++_shared
```

2.2.3 拷贝 SDK so 到资源库

使用 NPU 执行推理，将 SDK 中 libhiai.so，libhcl.so，libhiai_ir.so 复制到 Android Studio 下的/libs/arm64-v8a 下：



把离线模型以及标签文件(DDK 提供的 Android 源码目录 src/main/assets/下的.om 文件和 labels_caffe.txt 文件)拷贝至所创建工程的/src/main/assets 下。

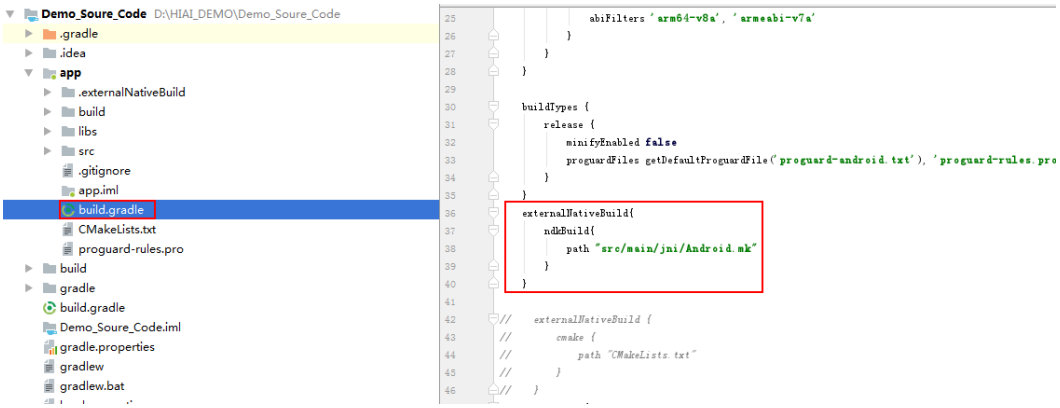


说明

如果集成 CPU 执行推理的能力，则需要拷贝 libcpucl.so 到 Android Studio 的/libs/arm64-v8a 下。

2.2.4 编辑 build.gradle 文件

指定 ndk 编译 C++文件，在/src/build.gradle 文件中，添加 ndk 编译信息如下：



2.3 模型集成

2.3.1 概述

模型推理集成流程包含模型预处理、模型推理。由于 DDK 提供了同步和异步接口，应用开发者根据需求选择使用同步或者异步接口方式。

本节阐述同步和异步模式下单模型的使用，从流程上分别阐述每个步骤在 DDK 层,JNI 层以及应用层的实现和调用，具体代码参考 DDK Demo。接口参见“3 模型推理 APIs”。

Demo 中：

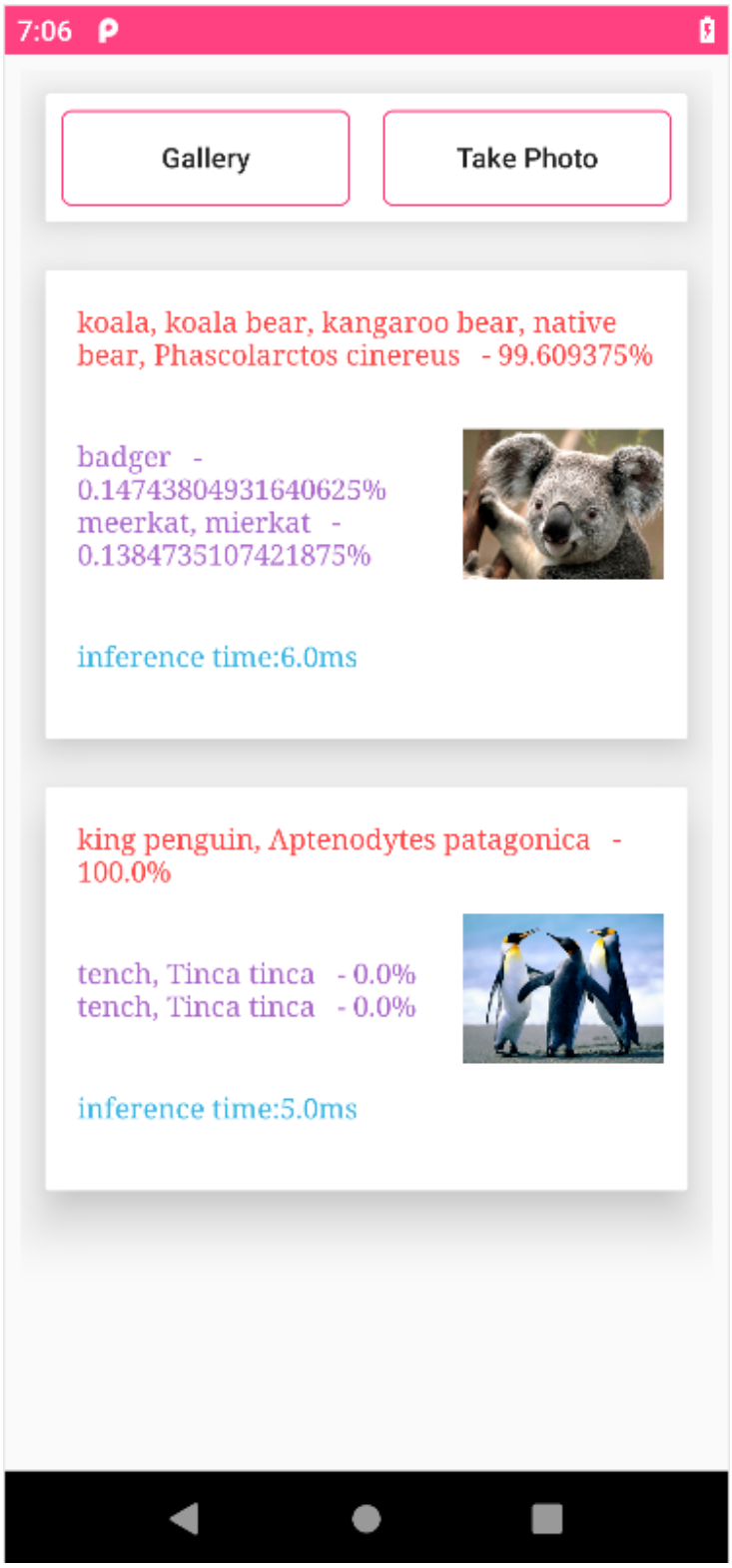
- 同步模式下

  - 应用层代码文件：SyncClassifyActivity.java
  - JNI 层代码文件：classify_jni.cpp
- 异步模式下
  - 应用层代码文件：AsyncClassifyActivity.java
  - JNI 层代码文件：classify_async_jni.cpp

Demo 支持从图库中选择图片或者拍照去做分类。

App 运行效果图如图 2-1 所示。

图2-1 DemoApk 运行结果视图



2.3.2 模型预处理

模型预处理主要功能如下：

- 获取 HiAI DDK 版本，判断是否支持 NPU。
- 判断离线模型是否适合在当前 HiAI 版本上运行。
- OM 模型不兼容当前 HiAI 版本时，通过模型推理中的在线编译接口来尝试版本兼容。

2.3.2.1 应用层模型预处理

通过调用 JNI 层 modelCompatibilityProcessFromFile 函数，模型推理前完成模型兼容性处理，判断当前模型是否能够运行在 NPU 上。应用层代码可参见 app_sample\inference_demo\Demo_Soure_Code\app\src\main\java\com\huawei\hiaidemo\view\MainActivity.java 文件中的 modelCompatibilityProcess 函数。

2.3.2.2 JNI 层模型预处理

JNI 层模型兼容性处理通过获取 HiAI 版本号以及新增的模型兼容性检查接口、在线编译接口来完成判断输入的 OM 模型能否运行在当前终端 NPU 上。主要步骤为：

步骤 1 创建 AiModelMngerClient 对象。

步骤 2 创建 AiModelBuilder 对象。

步骤 3 获取 Membuffer，使用上述 builder 对象的 InputMemBufferCreate 函数，输入参数离线模型路径。

步骤 4 调用 AiModelDescription 对象的 SetModelBuffer 函数进行设置。

步骤 5 调用 AiModelMngerClient 对象的 CheckModelCompatibility 函数进行兼容性检查。

步骤 6 判断是否满足兼容性，如果满足则返回退出；如果不满足，则进行离线模型的在线编译生成兼容的离线模型。

----结束

JNI 层实现代码可参见

app_sample\inference_demo\Demo_Soure_Code\app\src\main\jni\buildmodel.cpp 文件中的 Java_com_huawei_hiaidemo_utils_ModelManager_modelCompatibilityProcessFromFile 函数。

2.3.3 模型推理

2.3.3.1 同步模式

同步模式下的模型推理性能分析可以使用 Profiling 工具，具体的使用说明请参考《华为 HiAI_DDK_V320_系统调试工具使用说明书》。

2.3.3.1.1 应用层实现

首先通过函数 getPixels 从图片中获取模型输入，然后通过调用 JNI 层 runModelSync 函数，进行模型推理，完成推理后调用 postPost 函数进行后处理，输出结果到界面。

实现代码参见 app_sample\inference_demo\Demo_Soure_Code\app\src\main\java\com\huawei\hiaidemo\view\SyncClassifyActivity.java 文件中的 runModel 函数，该函数会调用 ModelManager.java 文件中的 runModelSync 函数。

2.3.3.1.2 JNI 层实现

主要步骤为：

步骤 1 调用函数 env->GetByteArrayElements(buf_, nullptr)和 env->GetArrayLength(buf_)获取应用层输入数据和大小；

步骤 2 调用函数 env->GetObjectClass(modelInfo)获取模型信息；

步骤 3 调用模型管家函数 LoadModelSync 创建模型管家和加载模型；

步骤 4 调用 GetModelIOTensorDim 函数获取模型的输入输出 shape，通过类 AiTensor 中的 init 函数初始化输入张量 input_tensor 和输出张量 output_tensor；

步骤 5 将应用层传入的 dataBuff 通过函数 memcpy 拷贝到 input_tensor；

步骤 6 调用类 AiModelMngerClient 中的 Process 进行模型推理。

步骤 7 将推理结果从 output_tensor 中取出。

----结束

JNI 实现代码参见

app_sample\inference_demo\Demo_Soure_Code\app\src\main\jni\classify_jni.cpp 文件中的 Java_com_huawei_hiaidemo_utils_ModelManager_runModelSync 函数。

2.3.3.2 异步模式

2.3.3.2.1 应用层实现

首先通过函数 getPixels 从图片中获取模型输入，并新建并注册 ModelManagerListener 监听器对象，通过调用 JNI 层 runModelAsync 函数，完成异步模式下模型运行。

实现代码参见 app_sample\inference_demo\Demo_Soure_Code\app\src\main\java\com\huawei\hiaidemo\view\AsyncClassifyActivity.java 文件中的 runModel 函数，该函数会调用 ModelManager.java 文件中的函数 runModelAsync。

2.3.3.2.2 JNI 层实现

异步模式下模型运行过程和同步相同，具体参见“2.3.3.1.2 JNI 层实现”，只是在调用类 AiModelMngerClient 中的函数 Process 时，注意使用的是异步模型管理引擎调用。

异步模式下运行模型后的后处理操作是通过回调函数 OnProcessDone 实现的。如果要使用异步模型管理引擎，需要在新建 AiModelMngerClient 对象是，调用 Init 函数，初

始化 ModelManagerListener 监听器对象。实现代码参见 app_sample\inference_demo\Demo_Soure_Code\app\src\main\jni\classify_async_jni.cpp 文件中的 Java_com_huawei_hiaidemo_utils_ModelManager_runModelAsync 函数。

2.3.3.3 涉及的接口

DDK 模型推理所使用到的主要接口函数原型如下：

```
AIStatus Init(shared_ptr<AiModelManagerClientListener> listener);
AIStatus Load(vector<shared_ptr<AiModelDescription>> &model_desc);
AIStatus Process(AiContext &context, vector<shared_ptr<AiTensor>>
&input_tensor, vector<shared_ptr<AiTensor>> &output_tensor, uint32_t
timeout, int32_t &iStamp);
AIStatus GetModelIOTensorDim(const string& model_name,
vector<TensorDimension>& input_tensor, vector<TensorDimension>&
output_tensor);
```

参数含义请参考：“3 模型推理 APIs”。

3 模型推理 APIs

3.1 模型管家类

须知

1. 模型管家是线程非安全的，不支持多线程共享同一个模型管家，每个线程可以创建各自的模型管家实例。
2. 一个 APP 最多创建 256 个模型管家实例。
3. 一个模型管家实例只能调用一次 Load，并且同步模式最多加载 63 个模型，异步模式最多加载 32 个模型。
4. 异步推理场景下，模型推理接口的任务队列最大长度为 512。

3.1.1 获取 HiAI 版本号

表3-1 GetVersion 接口说明

功能描述	获取手机 ROM 中的 HiAI 版本号
接口原型	char * GetVersion();
参数说明	无
返回值	执行成功，返回相应的 HiAI 版本号。 版本号名称，以<major>.<middle>.<minor>.<point>的形式描述版本。 <major>：产品形态，1XX：手机形态，2XX：边缘计算形态，3XX：Cloud 形态。 <middle>：XXX 三位数表示，同一产品形态的 V 版本，如手机形态下，HiAIV100，HiAIV200，HiAIV300 等。 <minor>：增量 C 版本，新增大特性，XXX 三位数表示。 <point>：B 版本或者补丁版本，XXX 三位数表示，最后一位非 0，表示补丁版本。 例如某手机 ROM 返回的 HiAI 版本号是 100.300.010.010,如果返回

	nullptr 或者 000.000.000.000，表示此版本不支持 NPU 加速；执行失败，返回相应错误值。关于 HiAI Version 说明可参考《华为 HiAI_DDK_V320_缩略语》

3.1.2 创建模型管家

表3-2 Init 接口说明

功能描述	创建模型管家
接口原型	AIStatus Init(shared_ptr<AiModelManagerClientListener> listener);
参数说明	输入参数 listener：监听接口，nullptr 为同步模型管家，非 nullptr 为异步模型管家
返回值	AIStatus::AI_SUCCESS：成功，Others：失败

3.1.3 加载模型

表3-3 load 接口说明

功能描述	加载模型
接口原型	AIStatus Load(vector<shared_ptr<AiModelDescription>> &model_desc);
参数说明	输入参数 model_desc： 模型描述信息数组，可输入多个模型，模型名称不能重复
返回值	AIStatus::AI_SUCCESS：成功，Others：失败

3.1.4 推理模型

表3-4 Process 接口说明

功能描述	模型推理接口
接口原型	AIStatus Process(AiContext &context, vector<shared_ptr<AiTensor>> &input_tensor, vector<shared_ptr<AiTensor>> &output_tensor, uint32_t timeout, int32_t &iStamp);
参数说明	输入参数 context：模型运行上下文，必须带 model_name 字段 输入参数 input_tensor：模型输入节点 Tensor 信息 输入输出参数 output_tensor：模型输出节点 Tensor 信息 输入参数 timeout：推理超时时间(毫秒) 输出参数 iStamp：异步返回标识，基于该标识和模型名称做回调索

	引
返回值	AIStatus::AI_SUCCESS：成功，Others：失败

3.1.5 检查模型兼容性

表3-5 CheckModelCompatibility 接口说明

功能描述	模型兼容性检查接口
接口原型	AIStatus CheckModelCompatibility(AiModelDescription &model_desc, bool &isModelCompatibility);
参数说明	输入参数 model_desc：模型描述 输出参数 isModelCompatibility：兼容性检查标识，True 代表模型兼容性检查通过,False 代表模型兼容性检查失败。
返回值	AIStatus::AI_SUCCESS：成功，Others：失败

3.1.6 获取模型输入输出 Tensor 信息

表3-6 GetModelIOTensorDim 接口说明

功能描述	获取模型输入输出 Tensor 信息
接口原型	AIStatus GetModelIOTensorDim(const string& model_name, vector<TensorDimension>& input_tensor, vector<TensorDimension>& output_tensor);
参数说明	输入参数 model_name：模型名称 输出参数 input_tensor：存储模型输入节点 Tensor 信息 输出参数 output_tensor: 存储模型输出节点 Tensor 信息
返回值	AIStatus::AI_SUCCESS：成功，Others：失败

3.1.7 卸载模型

表3-7 UnLoadModel 接口说明

功能描述	卸载之前加载的模型
接口原型	AIStatus UnLoadModel();
参数说明	无
返回值	AIStatus::AI_SUCCESS：成功，Others：失败

说明

该接口可以释放模型加载相关内存，建议与 Load 配套使用。重复调用 Load，不用 UnLoadModel 会导致内存大量占用。

3.2 模型编译类

3.2.1 OM 离线模型在线编译接口

表3-8 BuildModel 接口说明

功能描述	OM 离线模型在线编译接口
接口原型	AIStatus BuildModel(const vector<MemBuffer *> &input_membuffer, MemBuffer *output_model_buffer, uint32_t &output_model_size);
参数说明	输入参数 input_membuffer：输入的 OM 离线模型 buffer 输出参数 output_model_buffer: 输出模型 buffer 输出参数 output_model_size：输出模型大小（单位：byte）
返回值	AIStatus::AI_SUCCESS：成功，Others：失败

说明

建议将编译后的模型文件（output_model_buffer）本地保存，以便于后续加载该模型文件时提升加载性能。

3.2.2 从文件读取 OM 离线模型 proto 信息

表3-9 ReadBinaryProto 接口说明

功能描述	从文件读取 OM 离线模型 proto 信息
接口原型	MemBuffer* ReadBinaryProto(const string path);
参数说明	输入参数 path：模型文件路径
返回值	MemBuffer 地址；如果为 nullptr 表示读取失败

3.2.3 从内存读取 OM 离线模型 proto 信息

表3-10 ReadBinaryProto 接口说明

功能描述	从内存读取 OM 离线模型 proto 信息
接口原型	MemBuffer* ReadBinaryProto(void* data, uint32_t size);
参数说明	输入参数 data：OM 离线模型内存地址 输入参数 size: OM 离线模型内存存储大小（单位：byte）
返回值	MemBuffer 地址；如果为 nullptr 表示创建失败

3.2.4 从内存创建 OM 模型 MemBuffer

表3-11 InputMemBufferCreate 接口说明

功能描述	从内存创建 OM 模型 MemBuffer
接口原型	MemBuffer* InputMemBufferCreate(void *data, uint32_t size);
参数说明	输入参数 data：模型用户内存地址 输入参数 size: 模型内存存储大小（单位：byte）
返回值	MemBuffer 地址；如果为 nullptr 表示创建失败

3.2.5 从文件创建 OM 模型 MemBuffer

表3-12 InputMemBufferCreate 接口说明

功能描述	从文件创建 OM 模型 MemBuffer
接口原型	MemBuffer* InputMemBufferCreate(const string path);
参数说明	输入参数 path：模型文件路径
返回值	MemBuffer 地址；如果为 nullptr 表示创建失败

3.2.6 为在线编译输出模型创建 MemBuffer

表3-13 OutputMemBufferCreate 接口说明

功能描述	为在线编译输出模型创建 MemBuffer
接口原型	MemBuffer* OutputMemBufferCreate(const int32_t framework, const vector<MemBuffer *> &input_membuffer);

参数说明	输入参数 framework：模型平台类型 输入参数 input_membuffer: 输入的 OM 离线模型 buffer
返回值	MemBuffer 地址；如果为 nullptr 表示创建失败

3.2.7 注销 MemBuffer 内存

表3-14 MemBufferDestroy 接口说明

功能描述	注销 MemBuffer 内存，通过上述 MemBuffer 申请的内存最终都需要由此接口进行释放
接口原型	void MemBufferDestroy(MemBuffer *membuf);
参数说明	输入参数 membuf：创建的 MemBuffer 内存
返回值	无

3.2.8 导出在线编译模型文件

表3-15 MemBufferExportFile 接口说明

功能描述	将 buffer 里的模型导出到文件
接口原型	AIStatus MemBufferExportFile(MemBuffer *membuf, const uint32_t build_size, const string build_path);
参数说明	输入参数 membuf：存储离线模型信息内存指针 输入参数 build_size：离线模型大小（单位：byte） 输入参数 build_path：离线模型导出文件存储路径
返回值	AIStatus::AI_SUCCESS：成功，Others：失败

3.3 模型描述类

3.3.1 获取模型名称

表3-16 GetName 接口说明

功能描述	获取模型名称
接口原型	string GetName() const;

参数说明	无
返回值	模型名称字符串，如果为 NULL，则获取失败。

3.3.2 获取模型内存

表3-17 GetModelBuffer 接口说明

功能描述	获取模型内存
接口原型	void* GetModelBuffer() const;
参数说明	无
返回值	void *

3.3.3 设置模型内存

表3-18 SetModelBuffer 接口说明

功能描述	设置模型内存
接口原型	AIStatus SetModelBuffer(const void* data, uint32_t size);
参数说明	输入参数 Data:模型 buffer 地址 输入参数 Size:模型大小（单位：byte）
返回值	AIStatus::AI_SUCCESS：成功，Others：失败

3.3.4 获取加载模型频率

表3-19 GetFrequency 接口说明

功能描述	获取加载模型的频率
接口原型	int32_t GetFrequency() const;
参数说明	无
返回值	AiModelDescription_Frequency 中的枚举值，请参考“3.10.1 AiModelDescription_Frequency”

3.3.5 获取模型框架类型

表3-20 GetFramework 接口说明

功能描述	获取模型框架类型
接口原型	int32_t GetFramework() const;
参数说明	无
返回值	AiModelDescription_Framework 中的枚举值，请参考“3.10.3 AiModelDescription_Framework”

3.3.6 获取模型类型

表3-21 GetModelType 接口说明

功能描述	获取模型类型
接口原型	int32_t GetModelType() const;
参数说明	无
返回值	AiModelDescription_ModelType 中的枚举值，请参考“3.10.4 AiModelDescription_ModelType”

3.3.7 获取设备类型

表3-22 GetDeviceType 接口说明

功能描述	获取设备类型
接口原型	int32_t GetDeviceType() const;
参数说明	无
返回值	AiModelDescription_DeviceType 中的枚举值，请参考“3.10.2 AiModelDescription_DeviceType”

3.3.8 获取模型 size 大小

表3-23 GetModelNetSize 接口说明

功能描述	获取模型大小（单位：byte）
接口原型	uint32_t GetModelNetSize() const;

参数说明	无
返回值	模型大小，类型是 uint32_t

3.4 模型 Tensor 创建类

3.4.1 初始化 Tensor

表3-24 根据 Tensor 的尺寸进行初始化

功能描述	根据 Tensor 的尺寸进行初始化
接口原型	AIStatus Init(const TensorDimension* dim);
参数说明	输入参数 dim：输入 Tensor 的尺寸结构信息
返回值	AIStatus::AI_SUCCESS：成功，Others：失败,如下： AIStatus::AI_INVALID_PARA，dim 为空 AIStatus::AI_FAILED， Tensor size 为 0 或者申请内存失败

说明

表 3-25 和表 3-26 为 HIAI DDK V310 新增接口，其中表 3-25 接口仅 AIPP 模型支持。

表3-25 根据 NHW 和图片的格式初始化 Tensor

功能描述	根据 NHW 和图片的格式初始化 Tensor
接口原型	AIStatus Init(uint32_t number, uint32_t height, uint32_t width, AiTensorImage_Format format);
参数说明	输入参数 number：输入的 Tensor 的数量 输入参数 height：输入的 Tensor 的高度 输入参数 width：输入的 Tensor 的宽度 输入参数 format：输入图片的格式 AiTensorImage_Format 类型（取值范围： AiTensorImage_YUV420SP_U8, AiTensorImage_XRGB8888_U8, AiTensorImage_YUV400_U8, AiTensorImage_ARGB8888_U8, AiTensorImage_YUYV_U8, AiTensorImage_YUV422SP_U8, AiTensorImage_AYUV444_U8）

返回值	AIStatus::AI_SUCCESS：成功；Others：失败，如下： AIStatus::AI_INVALID_PARA，N,H,W 中任意一个为 0 或者 format 不在所支持的范围内 AIStatus::AI_FAILED， Tensor size 为 0 或者申请内存失败

表3-26 根据 Tensor 的尺寸和 DataType 初始化 Tensor

功能描述	根据 Tensor 的尺寸和 DataType 初始化 Tensor
接口原型	AIStatus Init(const TensorDimension* dim, HIAI_DataType DataType);
参数说明	输入参数 dim: 输入 Tensor 的尺寸结构信息 输入参数 DataType: 枚举类型，包含 HIAI_DATATYPE_UINT8, HIAI_DATATYPE_FLOAT32, HIAI_DATATYPE_FLOAT16, HIAI_DATATYPE_INT32，HIAI_DATATYPE_INT8，HIAI_DATATYPE_INT16，HIAI_DATATYPE_BOOL，HIAI_DATATYPE_INT64，HIAI_DATATYPE_UINT32，HIAI_DATATYPE_DOUBLE
返回值	AIStatus::AI_SUCCESS：成功，Others：失败,如下： AIStatus::AI_INVALID_PARA，dim 为空或者 DataType 不在可支持范围 AIStatus::AI_FAILED， Tensor size 为 0 或者申请内存失败

表3-27 根据 NativeHandle, TensorDimension 和 DataType 初始化 Tensor

功能描述	根据 NativeHandle, TensorDimension 和 DataType 初始化 Tensor
接口原型	AIStatus Init(const NativeHandle& handle, const TensorDimension* dim, HIAI_DataType pdataType);
参数说明	输入参数 handle: 输入 NativeHandle 的结构信息； NativeHandle 只支持 ION 内存 输入参数 dim: 输入 Tensor 的尺寸结构信息 输入参数 DataType: 枚举类型，包含 HIAI_DATATYPE_UINT8, HIAI_DATATYPE_FLOAT32, HIAI_DATATYPE_FLOAT16, HIAI_DATATYPE_INT32，HIAI_DATATYPE_INT8，HIAI_DATATYPE_INT16，HIAI_DATATYPE_BOOL，HIAI_DATATYPE_INT64，HIAI_DATATYPE_UINT32，HIAI_DATATYPE_DOUBLE
返回值	AIStatus::AI_SUCCESS：成功，Others：失败,如下： AIStatus::AI_INVALID_PARA，dim 为空或者 DataType 不在可支持范围 AIStatus::AI_FAILED， Tensor size 为 0 或者申请内存失败

3.4.2 获取 Tensor 地址

表3-28 GetBuffer 接口说明

功能描述	获取 Tensor buffer 地址接口
接口原型	void *GetBuffer() const;
参数说明	无
返回值	void *

3.4.3 获取 Tensor 大小

表3-29 GetSize 接口说明

功能描述	获取 Tensor buffer 内存大小（单位：byte）
接口原型	uint32_t GetSize() const;
参数说明	无
返回值	Tensor 大小，类型为 uint32_t

3.4.4 设置 Tensor 的尺寸结构信息

表3-30 SetTensorDimension 接口说明

功能描述	设置 Tensor 的尺寸结构信息
接口原型	AIStatus SetTensorDimension(const TensorDimension* dim);
参数说明	输入参数 dim：输入 Tensor 的尺寸结构信息
返回值	AIStatus::AI_SUCCESS：成功，Others：失败,如下： AIStatus::AI_INVALID_PARA，dim 为空 AIStatus::AI_FAILED， Tensor size 为 0 或者申请内存失败

3.4.5 获取 Tensor 的尺寸结构信息

表3-31 GetTensorDimension 接口说明

功能描述	获取 Tensor 的尺寸结构信息

接口原型	TensorDimension GetTensorDimension() const;
参数说明	无
返回值	Tensor 的尺寸结构信息

3.4.6 获取 TensorBuffer 地址

表3-32 GetTensorBuffer 接口说明

功能描述	获取 TensorBuffer 地址接口
接口原型	void *GetTensorBuffer() const;
参数说明	无
返回值	void*

3.5 AIPP 对外接口类

3.5.1 通用接口

3.5.1.1 获取模型 AIPP 配置信息

表3-33 AiModelMngerClient::GetModelAippPara 接口说明

功能描述	获取模型 AIPP 配置信息
接口原型	AIStatus GetModelAippPara(const std::string& modelName, std::vector<std::shared_ptr<AippPara>>& aippPara);
参数说明	输入参数 modelName：模型名称 输出参数 AippPara：模型中的 AIPP 配置参数，保存在 AippPara 对象中
返回值	AI_SUCCESS 表示获取成功，返回其他值表示获取失败

3.5.1.2 获取模型中特定下标的 AIPP 配置信息

表3-34 AiModelMngerClient::GetModelAippPara 接口说明

功能描述	获取模型中某个输入的 AIPP 配置信息
接口原型	AIStatus GetModelAippPara(const std::string& modelName, uint32_t

	index, std::vector<std::shared_ptr<AippPara>>& aippPara);
参数说明	输入参数 modelName：模型名称 输入参数 index：输入下标，取值需>=0 输出参数 AippPara：下标为 index 的模型输入对应的 AIPP 配置参数，保存在 AippPara 对象中
返回值	AI_SUCCESS 表示获取成功，返回其他值表示获取失败

3.5.1.3 获取 Tensor buffer 地址

表3-35 AippTensor::GetBuffer 接口说明

功能描述	获取 Tensor buffer 地址
接口原型	void* GetBuffer() const
参数说明	无
返回值	void * tensor buffer 地址

3.5.1.4 获取 Tensor buffer 内存大小

表3-36 AippTensor::GetSize 接口说明

功能描述	获取 Tensor buffer 地址
接口原型	uint32_t GetSize() const;
参数说明	无
返回值	Tensor buffer 内存大小

3.5.1.5 获取 Tensor 指针

表3-37 AippTensor::GetAiTensor 接口说明

功能描述	获取 Tensor 指针
接口原型	std::shared_ptr<AiTensor> GetAiTensor() const;
参数说明	无
返回值	Tensor 指针

3.5.1.6 获取所有 AippPara 对象指针

表3-38 AippTensor::GetAippParas 接口说明

功能描述	获取 AippPara 对象指针
接口原型	std::vector<std::shared_ptr<AippPara>> GetAippParas() const;
参数说明	无
返回值	保存所有 AippPara 指针的 vector

3.5.1.7 获取特定 AippPara 对象指针

表3-39 AippTensor::GetAippParas 接口说明

功能描述	获取特定 AippPara 对象指针
接口原型	std::shared_ptr<AippPara> GetAippParas(uint32_t index) const;
参数说明	index：输入下标，取值需>=0
返回值	AippPara 对象指针

3.5.1.8 用于 AippPara 对象的初始化

表3-40 AippPara::Init 接口说明

功能描述	用于 AippPara 对象的初始化
接口原型	AIStatus Init(uint32_t batchCount);
参数说明	输入参数 batchCount：模型输入的 batch 大小，对应 NCHW 维度的 N 维，默认为 1
返回值	AI_SUCCESS 表示获取成功，返回其他值表示获取失败

3.5.1.9 获取 batch 大小

表3-41 AippPara::GetBatchCount 接口说明

功能描述	获取 batch 大小
接口原型	uint32_t GetBatchCount();
参数说明	无
返回值	AippPara 对象中的 batch 大小

3.5.1.10 设置 inputIndex 参数

表3-42 AippPara::SetInputIndex 接口说明

功能描述	设置 AIPP inputIndex 参数
接口原型	AIStatus SetInputIndex(uint32_t inputIndex);
参数说明	inputIndex：用于标识此 AIPP 参数作用于模型的第几个输入，取值需>=0
返回值	AI_SUCCESS 表示成功，返回其他值表示失败

3.5.1.11 获取 inputIndex 参数

表3-43 AippPara::GetInputIndex 接口说明

功能描述	获取 AIPP 参数中的 inputIndex 参数，inputIndex 参数用于标识 AIPP 配置参数作用于第几个模型输入
接口原型	int32_t GetInputIndex();
参数说明	无
返回值	inputIndex 参数

3.5.1.12 设置 inputAippIndex 参数

表3-44 AippPara::SetInputAippIndex 接口说明

功能描述	设置 AIPP inputAippIndex 参数
接口原型	AIStatus SetInputAippIndex(uint32_t inputAippIndex);
参数说明	inputAippIndex：用于标识 AIPP 配置参数在输入 Data 有多个输出分支时作用于第几个分支，取值需>=0
返回值	AI_SUCCESS 表示成功，返回其他值表示失败

3.5.1.13 获取 inputAippIndex 参数

表3-45 AippPara::GetInputAippIndex 接口说明

功能描述	获取 AIPP 参数中的 inputAippIndex 参数，inputAippIndex 参数用于标识 AIPP 配置参数在输入 Data 有多个输出分支时作用于第几个分支

接口原型	int32_t GetInputAippIndex();
参数说明	无
返回值	inputAippIndex 参数

3.5.1.14 获取输入图片尺寸

表3-46 AippPara::GetInputShape 接口说明

功能描述	获取 AIPP 参数中的图片尺寸参数
接口原型	AippInputShape GetInputShape();
参数说明	无
返回值	图片尺寸，AippInputShape 结构体

3.5.1.15 获取输入图片格式

表3-47 AippPara::GetInputFormat 接口说明

功能描述	获取 AIPP 参数中的输入图片格式参数
接口原型	AiTensorImage_Format GetInputFormat();
参数说明	无
返回值	输入图片类型

3.5.1.16 获取色域转换参数

表3-48 AippPara::GetCscPara 接口说明

功能描述	获取 AIPP 参数中的 Color Space Conversion 色域转换参数
接口原型	AippCscPara GetCscPara();
参数说明	无

返回值	色域转换参数
	参考：
	`struct AippCscPara {`
	`bool switch_ = false;`
	`int32_t matrixR0C0 = 0;`
	`int32_t matrixR0C1 = 0;`
	`int32_t matrixR0C2 = 0;`
	`int32_t matrixR1C0 = 0;`

	int32_t matrixR1C1 = 0;
	int32_t matrixR1C2 = 0;
	int32_t matrixR2C0 = 0;
	int32_t matrixR2C1 = 0;
	int32_t matrixR2C2 = 0;
	int32_t outputBias0 = 0;
	int32_t outputBias1 = 0;
	int32_t outputBias2 = 0;
	int32_t inputBias0 = 0;
	int32_t inputBias1 = 0;
	int32_t inputBias2 = 0;
	};

3.5.1.17 获取通道交换参数

表3-49 AippPara::GetChannelSwapPara 接口说明

功能描述	获取 AIPP 参数中的 Channel Swap 通道交换参数
接口原型	AippChannelSwapPara GetChannelSwapPara();
参数说明	无
返回值	Channel Swap 通道交换参数

3.5.1.18 获取图片裁剪参数

表3-50 AippPara::GetCropPara 接口说明

功能描述	获取 AIPP 参数中的 Crop 参数
接口原型	AippCropPara GetCropPara(uint32_t batchIndex);
参数说明	batchIndex：batch 下标，标识获取第几个 batch 中的 Crop 参数，取值需>=0
返回值	Crop 参数，如果 batchIndex 超过了 AippPara 的 batchCount，返回模型 AippCropPara 参数值

3.5.1.19 获取图片缩放参数

表3-51 AippPara::GetResizePara 接口说明

功能描述	获取 AIPP 参数中的 Resize 参数
接口原型	AippResizePara GetResizePara(uint32_t batchIndex);
参数说明	batchIndex：batch 下标，标识获取第几个 batch 中的 Resize 参数，取

	值需>=0

返回值	Resize 参数
	参考：
	`struct AippResizePara {`
	`bool switch_ = false;`
	`uint32_t resizeOutputSizeW = 0;`
	`uint32_t resizeOutputSizeH = 0;`
	`};`

3.5.1.20 获取图片补边参数

表3-52 AippPara::GetPaddingPara 接口说明

功能描述	获取 AIPP 参数中的 Padding 参数
接口原型	AippPaddingPara GetPaddingPara(uint32_t batchIndex);
参数说明	batchIndex：batch 下标，标识获取第几个 batch 中的 Padding 参数，取值需>=0
返回值	Padding 参数

3.5.1.21 获取数据类型转换参数

表3-53 AippPara::GetDtcPara 接口说明

功能描述	获取 AIPP 参数中的 Data Type Conversion 参数
接口原型	AippDtcPara GetDtcPara(uint32_t batchIndex);
参数说明	batchIndex：batch 下标，标识获取第几个 batch 中的 DTC 参数，取值需>=0
返回值	DTC 参数

3.5.1.22 通过 buffer_handle_t 创建 AippTensor

表3-54 根据 buffer_handle_t，TensorDimension，imageFormat 创建 AippTensor

功能描述	根据 buffer_handle_t，TensorDimension，imageFormat，创建 AippTensor
接口原型	std::shared_ptr<AippTensor> HIAI_CreateAiPPTensorFromHandle(buffer_handle_t& handle, const TensorDimension* dim, AiTensorImage_Format imageFormat = AiTensorImage_INVALID);

参数说明	输入参数 handle：buffer_handle_t 结构体信息； 输入参数 dim: 输入 Tensor 的尺寸结构信息 输入参数 imageFormat: handle 中存储的 ImageFormat 信息 （支持如下格式 AiTensorImage_XRGB8888_U8 AiTensorImage_RGB888_U8 AiTensorImage_YUV422SP_U8 AiTensorImage_YUV420SP_U8 AiTensorImage_YUYV_U8 AiTensorImage_YUV400_U8）
返回值	AippTenso shared_ptr 指针

3.5.2 动态 AIPP 接口

3.5.2.1 设置输入图片尺寸

表3-55 AippPara::SetInputShape 接口说明

功能描述	设置 AIPP 输入图片尺寸 inputShape 参数

接口原型	AIStatus SetInputShape(AippInputShape inputShape);
	参考：
	`struct AippInputShape {`
	`uint32_t srcImageSizeW = 0;`
	`uint32_t srcImageSizeH = 0;`
	`};`

参数说明	inputShape：输入图片尺寸
返回值	AI_SUCCESS 表示成功，返回其他值表示失败

3.5.2.2 设置输入图片格式

表3-56 AippPara::SetInputFormat 接口说明

功能描述	设置 AIPP 的 inputFormat 参数，设置输入图片的类型
接口原型	AIStatus SetInputFormat(AiTensorImage_Format inputFormat);
参数说明	inputFormat：输入图片类型，枚举类型（取值范围： AiTensorImage_YUV420SP_U8, AiTensorImage_XRGB8888_U8, AiTensorImage_YUV400_U8, AiTensorImage_ARGB8888_U8,

	AiTensorImage_YUYV_U8, AiTensorImage_YUV422SP_U8, AiTensorImage_AYUV444_U8）
返回值	AI_SUCCESS 表示成功，返回其他值表示失败

3.5.2.3 设置色域转换参数

表3-57 AippPara::SetCscPara 接口说明

功能描述	设置 AIPP Color Space Conversion 色域转换参数
接口原型	AIStatus SetCscPara(AiTensorImage_Format targetFormat, ImageType imageType=JPEG);
参数说明	targetFormat：转换后的图片类型，系统根据转换后的图片类型自动填充 CSC 参数（取值范围： AiTensorImage_RGB888_U8,AiTensorImage_BGR888_U8,AiTensorImage_YUV444SP_U8,AiTensorImage_YVU444SP_U8,AiTensorImage_YUV400_U8） imageType：图片格式，默认 JPEG（取值范围：JPEG, BT_601_NARROW, BT_601_FULL, BT_709_NARROW）
返回值	AI_SUCCESS 表示成功，返回其他值表示失败

3.5.2.4 设置通道交换参数

表3-58 AippPara::SetChannelSwapPara 接口说明

功能描述	设置 AIPP ChannelSwap 通道交换参数
接口原型	AIStatus SetChannelSwapPara(AippChannelSwapPara channelSwapPara);

参数说明	channelSwapPara：ChannelSwap 通道交换参数
	参考：
	`struct AippChannelSwapPara {`
	`bool rbuvSwapSwitch = false;`
	`bool axSwapSwitch = false;`
	`};`

返回值	AI_SUCCESS 表示成功，返回其他值表示失败

3.5.2.5 设置所有 batch 的图片裁剪参数

表3-59 AippPara::SetCropPara 接口说明

功能描述	设置 AIPP Crop 图片裁剪参数，为所有的 batch 设置相同的 Crop 参数
接口原型	AIStatus SetCropPara(AippCropPara cropPara);

参数说明	cropPara：Crop 图片裁剪参数
	参考：
	`struct AippCropPara {`
	`bool switch_ = false;`
	`uint32_t cropStartPosW = 0;`
	`uint32_t cropStartPosH = 0;`
	`uint32_t cropSizeW = 0;`
	`uint32_t cropSizeH = 0;`
	`};`

返回值	AI_SUCCESS 表示成功，返回其他值表示失败

3.5.2.6 设置特定下标 batch 的图片裁剪参数

表3-60 AippPara::SetCropPara 接口说明

功能描述	设置 AIPP Crop 图片裁剪参数，为特定下标的 batch 设置 Crop 参数
接口原型	AIStatus SetCropPara(uint32_t batchIndex, AippCropPara cropPara);
参数说明	batchIndex：batch 下标，标识为第几个 batch 设置 Crop 参数，取值需>=0 cropPara：Crop 图片裁剪参数
返回值	AI_SUCCESS 表示成功，返回其他值表示失败

3.5.2.7 设置所有 batch 的图片缩放参数

表3-61 AippPara::SetResizePara 接口说明

功能描述	设置 AIPP Resize 图片缩放参数，为所有的 batch 设置相同的 Resize 参数
接口原型	AIStatus SetResizePara(AippResizePara resizePara);

参数说明	resizePara：Resize 图片缩放参数
	参考：
	`struct AippResizePara {`
	`bool switch_ = false;`
	`uint32_t resizeOutputSizeW = 0;`

	uint32_t resizeOutputSizeH = 0; };
返回值	AI_SUCCESS 表示成功，返回其他值表示失败

3.5.2.8 设置特定下标 batch 的图片缩放参数

表3-62 AippPara::SetResizePara 接口说明

功能描述	设置 AIPP Resize 图片缩放参数，为特定下标的 batch 设置 Resize 参数
接口原型	AIStatus SetResizePara(uint32_t batchIndex, AippResizePara resizePara);
参数说明	batchIndex：batch 下标，标识为第几个 batch 设置 Resize 参数，取值需>=0 resizePara：Resize 图片缩放参数
返回值	AI_SUCCESS 表示成功，返回其他值表示失败

3.5.2.9 设置所有 batch 的图片补边参数

表3-63 AippPara::SetPaddingPara 接口说明

功能描述	设置 AIPP Padding 图片补边参数，为所有的 batch 设置相同的 Padding 参数
接口原型	AIStatus SetPaddingPara(AippPaddingPara paddingPara);

参数说明	paddingPara：Padding 图片补边参数
	参考：
	struct AippPaddingPara {
	bool switch_ = false;
	uint32_t paddingSizeTop = 0;
	uint32_t paddingSizeBottom = 0;
	uint32_t paddingSizeLeft = 0;
	uint32_t paddingSizeRight = 0;
	};

返回值	AI_SUCCESS 表示成功，返回其他值表示失败

3.5.2.10 设置特定下标 batch 的图片补边参数

表3-64 AippPara::SetPaddingPara 接口说明

功能描述	设置 AIPP Padding 图片补边参数，为特定下标的 batch 设置 Padding

	参数
接口原型	AIStatus SetPaddingPara(uint32_t batchIndex, AippPaddingPara paddingPara);
参数说明	batchIndex：batch 下标，标识为第几个 batch 设置 Padding 参数，取值需>=0 paddingPara：Padding 图片补边参数
返回值	AI_SUCCESS 表示成功，返回其他值表示失败

3.5.2.11 设置所有 batch 的数据类型转换参数

表3-65 AippPara::SetDtcPara 接口说明

功能描述	设置 AIPP DTC 数据类型转换参数，为所有的 batch 设置相同的 DTC 参数
接口原型	AIStatus SetDtcPara(AippDtcPara dtcPara);

参数说明	dtcPara：DTC 数据类型转换参数
	参考：
	`struct AippDtcPara {`
	`int16_t pixelMeanChn0 = 0;`
	`int16_t pixelMeanChn1 = 0;`
	`int16_t pixelMeanChn2 = 0;`
	`int16_t pixelMeanChn3 = 0;`
	`float pixelMinChn0 = 0;`
	`float pixelMinChn1 = 0;`
	`float pixelMinChn2 = 0;`
	`float pixelMinChn3 = 0;`
	`float pixelVarReciChn0 = 1.0;`
	`float pixelVarReciChn1 = 1.0;`
	`float pixelVarReciChn2 = 1.0;`
	`float pixelVarReciChn3 = 1.0;`
	`};`

返回值	AI_SUCCESS 表示成功，返回其他值表示失败

3.5.2.12 设置特定下标 batch 的数据类型转换参数

表3-66 AippPara::SetDtcPara 接口说明

功能描述	设置 AIPP DTC 数据类型转换参数，为特定下标的 batch 设置 DTC 参数
接口原型	AIStatus SetDtcPara(uint32_t batchIndex, AippDtcPara dtcPara);
参数说明	batchIndex：batch 下标，标识为第几个 batch 设置 DTC 参数，取值

	需>=0 dtcPara：DTC 数据类型转换参数
返回值	AI_SUCCESS 表示成功，返回其他值表示失败

3.5.3 使用示例

假定当前有一个模型，训练时采用的训练集为 BRG888 的图片，使能了动态 AIPP 之后，可以接收 YUYV 类型的图片作为模型推理的输入；当用于模型推理的图片尺寸与训练集图片的尺寸不一致时，还可以使用 AIPP 的裁剪、缩放和 Padding 功能，改变输入图片尺寸。以下用例基于 HiAI API，使能了 AIPP 的裁剪功能，将一张 YUYV，尺寸为 480x480 的图片预处理为 224x224 的输入。

```
int ReadFile(const char* fileName, char*& dataBuff, int& fileLength);
int main(int argc, char* const argv[])
{
   if (argc != 4) {
      printf("wrong args,usage:./main modelname modelpath
input_data_file\n");
      return -1;
   }
   printf("Create model manager client! model file path: %s\n", argv[2]);
   string modelFilePath = argv[2];
   shared_ptr<AiModelMngerClient> modelManagerClient =
make_shared<AiModelMngerClient>();
   shared_ptr<AiModelBuilder> modelBuilder =
make_shared<AiModelBuilder>(modelManagerClient);
   auto ret = modelManagerClient->Init(nullptr);
   if (ret != 0) {
      printf("Init modelBuilder Failed!\n");
      return ret;
   }
   MemBuffer* buffer = modelBuilder->InputMemBufferCreate(modelFilePath);
   if (buffer == NULL) {
      printf("CrAiModelBuildereate memory buffer failed, please check
model file path.\n");
      return -1;
   }
   vector<shared_ptr<AiModelDescription>> modelDescs;
   printf("Create model description. version %s\n", modelManagerClient-
>GetVersion());
   shared_ptr<AiModelDescription> modelDesc =
make_shared<AiModelDescription>(argv[0], 3, 0, 0, 2);
   modelDesc->SetModelBuffer(buffer->GetMemBufferData(), buffer-
```



```
>GetMemBufferSize());
    modelDescs.push_back(modelDesc);

    ret = modelManagerClient->Load(modelDescs);
    if (ret != 0) {
        printf("Load model failed!\n");
        return ret;
    }
    printf("Load model success!\n");

    printf("Get model %s IO Tensor.\n", modelDesc->GetName().c_str());
    vector<TensorDimension> inputDimension;
    vector<TensorDimension> outputDimension;
    ret = modelManagerClient->GetModelIOTensorDim(modelDesc->GetName(),
inputDimension, outputDimension);
    if (ret != 0 && inputDimension.size() != 1 &&
outputDimension.size() != 1) {
        printf("Get Model IO Tensor Dimension failed.\n");
    }
    printf("INPUT NCHW : %d %d %d %d.\n" , inputDimension[0].GetNumber(),
inputDimension[0].GetChannel(), inputDimension[0].GetHeight(),
inputDimension[0].GetWidth());
    printf("OUTPUT NCHW : %d %d %d %d.\n" , outputDimension[0].GetNumber(),
outputDimension[0].GetChannel(), outputDimension[0].GetHeight(),
outputDimension[0].GetWidth());

    // 设置动态AIPP参数
    vector<shared_ptr<AippPara>> aippParas;
    shared_ptr<AippPara> aippPara = make_shared<AippPara>();
    aippPara->Init(1);
    ret = aippPara->SetInputFormat(AiTensorImage_YUYV_U8);
    ret = aippPara->SetInputShape({480, 480});
    ret = aippPara->SetCropPara({true, 100, 100, 224, 224});
    ret = aippPara->SetCscPara(AiTensorImage_BGR888_U8, JPEG);
    aippParas.push_back(aippPara);

    // 创建带AIPP的模型输入
    vector<shared_ptr<AiTensor>> inputTensor;
    for (auto in_dim : inputDimension) {
        shared_ptr<AiTensor> input = make_shared<AiTensor>();
        input->Init(1, 480, 480, AiTensorImage_YUYV_U8);
        shared_ptr<AippTensor> aippTensor = make_shared<AippTensor>(input,
aippParas);
```



```
        inputTensor.push_back(aippTensor);
    }
    char* dataBuff;
    int fileLength;
    ret = ReadFile(argv[3], dataBuff, fileLength);
    if (ret != 0 || dataBuff == NULL || fileLength != inputTensor[0]-
>GetSize()) {
        printf("Read input file failed!");
        delete [] dataBuff;
        return ret;
    }
    printf("Source copy file data len: %d, Dest Tensor buffer size %d\n",
fileLength, inputTensor[0]->GetSize());
    memcpy(inputTensor[0]->GetBuffer(), dataBuff, inputTensor[0]-
>GetSize());

    vector<shared_ptr<AiTensor>> outputTensor;
    for (auto out_dim : outputDimension) {
        shared_ptr<AiTensor> output = make_shared<AiTensor>();
        output->Init(&out_dim);
        outputTensor.push_back(output);
    }
    AiContext context;
    string key = "model_name";
    string value = argv[0];
    context.AddPara(key, value);
    int32_t iStamp;
    ret = modelManagerClient->Process(context, inputTensor, outputTensor,
3000, iStamp);
    if (ret != 0) {
        printf("Run aipp model failed!");
        delete [] dataBuff;
        return ret;
    }
    printf("Run model ret: %d stamp %d\n", ret, iStamp);

    uint32_t outputsize = outputDimension[0].GetNumber() *
outputDimension[0].GetChannel() * outputDimension[0].GetHeight() *
outputDimension[0].GetWidth();
    for (size_t i = 0; i < outputTensor.size(); ++i) {
        printf("Store process output to file: %d\n", outputTensor[i]-
>GetSize());
        string filenamep = string("output_sync_") + to_string(i);
        std::ofstream outputFile(filenamep, std::ios::out);
```



```
        outputFile.write((char*)outputTensor[i]->GetBuffer(),
outputTensor[i]->GetSize());
        outputFile.close();
    }
    delete [] dataBuff;
    return 0;
}
int ReadFile(const char* fileName, char*& dataBuff, int& fileLength)
{
    string fileNameStr = fileName;
    std::ifstream file(fileNameStr);
    if (!file.is_open()) {
        printf("Open file failed! path:%s\n", fileNameStr.c_str());
        return -1;
    }
    file.seekg (0, file.end);
    fileLength = file.tellg();
    printf("read file %s length: %d\n", fileNameStr.c_str(), fileLength);
    file.seekg (0, file.beg);
    dataBuff = new (std::nothrow) char[fileLength];
    file.read(dataBuff, fileLength);
    return 0;
}
```

3.6 用户自定义上下文类

3.6.1 获取 para 内容

表3-67 GetPara 接口说明

功能描述	通过 Key 获取 para 内容
接口原型	string GetPara(const string &key) const;
参数说明	输入参数 key：关键字类型
返回值	关键字内容

3.6.2 增加自定义的 para

表3-68 AddPara 接口说明

功能描述	增加自定义的 Para，其中“model_name”是必带的参数。

接口原型	void AddPara(const string &key, const string &value);
参数说明	输入参数 key：关键字类型 输出参数 value：参数值
返回值	无

3.6.3 设置自定义的 para

说明

本接口为 HIAI DDK V310 新增接口。

表3-69 SetPara 接口说明

功能描述	设置自定义的 Para。
接口原型	void SetPara(const string &key, const string &value);
参数说明	输入参数 key：关键字类型 输入参数 value：参数值
返回值	无

3.6.4 删除自定义的 para

说明

本接口为 HIAI DDK V310 新增接口。

表3-70 DelPara 接口说明

功能描述	删除自定义的 Para。
接口原型	void DelPara(const string &key);
参数说明	输入参数 key：关键字类型
返回值	无

3.6.5 清除自定义的 para

说明

本接口为 HIAI DDK V310 新增接口。

表3-71 ClearPara 接口说明

功能描述	清除自定义的 Para。
接口原型	void ClearPara();
参数说明	无
返回值	无

3.6.6 获取已添加的所有参数 key

表3-72 GetAllKeys 接口说明

功能描述	获取已添加的所有的参数 key
接口原型	AIStatus GetAllKeys(vector<string> &keys);
参数说明	输出参数 keys：所有关键字类型
返回值	AIStatus::AI_SUCCESS：成功，Others：失败

3.7 异步回调注册类

3.7.1 OnProcessDone 回调虚函数

表3-73 OnProcessDone 接口说明

功能描述	模型执行的结果异步回调虚函数，需要 APK 实现，在模型初始化时注册进来
接口原型	virtual void OnProcessDone(const AiContext &context, int32_t result, const vector<shared_ptr<AiTensor>> &out_tensor, int32_t iStamp)
参数说明	输入参数 context：用户自定义可扩展上下文 输入参数 result：推理结果 输入输出参数 out_tensor：推理后的输出 Tensor 输入参数 iStamp：异步处理标识
返回值	无

3.7.2 OnServiceDied 回调虚函数

表3-74 OnServiceDied 接口说明

功能描述	OnServiceDied 回调虚函数，需要 APK 实现。在 C++的实现中，当 server 出现异常并回调 ServiceDied 时会通知所有的模型管家注册的异步回调该接口。
接口原型	virtual void OnServiceDied()
参数说明	无
返回值	无

3.8 Tensor 尺寸的描述结构类

3.8.1 设置输入 Tensor 数量

表3-75 SetNumber 接口说明

功能描述	设置输入 Tensor 数量
接口原型	void SetNumber(const uint32_t number);
参数说明	输入参数 number：模型输入 Tensor 的数量
返回值	无

3.8.2 设置输入 Tensor 通道数

表3-76 SetChannel 接口说明

功能描述	设置输入 Tensor 通道数
接口原型	void SetChannel(const uint32_t channel);
参数说明	输入参数 channel：模型输入 Tensor 的通道数
返回值	无

3.8.3 设置输入 Tensor 高度

表3-77 SetHeight 接口说明

功能描述	设置输入 Tensor 高度
接口原型	void SetHeight(const uint32_t height)
参数说明	输入参数 height：模型输入 Tensor 的高度
返回值	无

3.8.4 设置输入 Tensor 宽度

表3-78 SetWidth 接口说明

功能描述	设置输入 Tensor 宽度
接口原型	void SetWidth(const uint32_t width);
参数说明	输入参数 width：模型输入 Tensor 的宽度
返回值	无

3.8.5 获取输入 Tensor 数目

表3-79 GetNumber 接口说明

功能描述	获取输入 Tensor 数目
接口原型	uint32_t GetNumber() const;
参数说明	无
返回值	输入 Tensor 数目

3.8.6 获取输入 Tensor 通道数

表3-80 GetChannel 接口说明

功能描述	获取输入 Tensor 通道数
接口原型	uint32_t GetChannel() const;
参数说明	无
返回值	输入 Tensor 通道数

3.8.7 获取输入 Tensor 高度

表3-81 GetHeight 接口说明

功能描述	获取输入 Tensor 高度
接口原型	uint32_t GetHeight() const;
参数说明	无
返回值	输入 Tensor 高度

3.8.8 获取输入 Tensor 宽度

表3-82 GetWidth 接口说明

功能描述	获取输入 Tensor 宽度
接口原型	uint32_t GetWidth() const;
参数说明	无
返回值	输入 Tensor 宽度

3.8.9 判断与输入 Tensor 的尺寸结构信息是否相同

表3-83 IsEqual 接口说明

功能描述	判断与输入 Tensor 的尺寸结构信息是否相同
接口原型	bool IsEqual(const TensorDimension &dim);
参数说明	需判断的 Tensor 尺寸结构信息
返回值	bool : 与输入 Tensor 尺寸结构信息是否相同

3.9 MemBuffer 通用的 HiAI 内存 buffer 类

3.9.1 获取通用 MEMBuffer 的内存地址

表3-84 GetMemBufferData 接口说明

功能描述	获取通用 MEMBuffer 的内存地址
接口原型	void* GetMemBufferData();
参数说明	无
返回值	void *

3.9.2 获取通用 MEMBuffer 的内存大小

表3-85 GetMemBufferSize 接口说明

功能描述	获取通用 MEMBuffer 的内存大小（单位：byte）
接口原型	uint32_t GetMemBufferSize();
参数说明	无
返回值	内存大小

3.10 接口枚举类型说明

3.10.1 AiModelDescription_Frequency

返回值	含义	取值
AiModelDescription_Frequency_LOW	低功耗	1
AiModelDescription_Frequency_MEDIUM	均衡	2
AiModelDescription_Frequency_HIGH	高性能	3
AiModelDescription_Frequency_EXTREME	极致性能	4

说明

极致性能会造成功耗偏高，建议谨慎使用。

3.10.2 AiModelDescription_DeviceType

返回值	含义	取值
AiModelDescription_DeviceType_NPU	设备类型为 NPU	0
AiModelDescription_DeviceType_IPU	设备类型为 IPU	1
AiModelDescription_DeviceType_MLU	设备类型为 MLU	2
AiModelDescription_DeviceType_CPU	设备类型为 CPU	3

3.10.3 AiModelDescription_Framework

返回值	含义	取值
HIAI_FRAMEWORK_NONE	非第三方框架	0
HIAI_FRAMEWORK_TENSORFLOW	HIAI 框架为 TensorFlow	1
HIAI_FRAMEWORK_KALDI	HIAI 框架为 KALDI	2
HIAI_FRAMEWORK_CAFFE	HIAI 框架为 Caffe	3
HIAI_FRAMEWORK_TENSORFLOW_8BIT	HIAI 框架为 TENSORFLOW_8BIT	4
HIAI_FRAMEWORK_CAFFE_8BIT	HIAI 框架为 CAFFE_8BIT	5

3.10.4 AiModelDescription_ModelType

返回值	含义	取值
HIAI_MODELTYPE_ONLINE	在线编译	0
HIAI_MODELTYPE_OFFLINE	离线模型	1

3.10.5 AiTensorImage_Format

返回值	含义	取值
AiTensorImage_YUV420SP_U8	AiTensorImage 为 YUV420SP_U8	0
AiTensorImage_XRGB8888_U8	AiTensorImage 为 XRGB8888_U8	1
AiTensorImage_YUV400_U8	AiTensorImage 为 YUV400_U8	2
AiTensorImage_ARGB8888_U8	AiTensorImage 为 AiTensorImage_ARGB8888_U8	3
AiTensorImage_YUYV_U8	AiTensorImage 为 AiTensorImage_YUYV_U8	4
AiTensorImage_YUV422SP_U8	AiTensorImage 为 AiTensorImage_YUV422SP_U8	5
AiTensorImage_AYUV444_U8	AiTensorImage 为 AiTensorImage_AYUV444_U8	6
AiTensorImage_RGB888_U8	AiTensorImage 为 AiTensorImage_RGB888_U8	7
AiTensorImage_BGR888_U8	AiTensorImage 为 AiTensorImage_BGR888_U8	8
AiTensorImage_YUV444SP_U8	AiTensorImage 为 AiTensorImage_YUV444SP_U8	9
AiTensorImage_YVU444SP_U8	AiTensorImage 为 AiTensorImage_YVU444SP_U8	10
AiTensorImage_INVALID	非法的 AiTensorImage	255

3.10.6 HIAI_DataType

返回值	含义	取值
HIAI_DATATYPE_UINT8	数据类型为 UINT8	0
HIAI_DATATYPE_FLOAT32	数据类型为 FLOAT32	1
HIAI_DATATYPE_FLOAT16	数据类型为 FLOAT16	2
HIAI_DATATYPE_INT32	数据类型为 INT32	3

返回值	含义	取值
HIAI_DATATYPE_INT8	数据类型为 INT8	4
HIAI_DATATYPE_INT16	数据类型为 INT16	5
HIAI_DATATYPE_BOOL	数据类型为 BOOL	6
HIAI_DATATYPE_INT64	数据类型为 INT64	7
HIAI_DATATYPE_UINT32	数据类型为 UINT32	8
HIAI_DATATYPE_DOUBLE	数据类型为 DOUBLE	9

4 错误码

错误码	错误类型	错误码原型	错误码触发条件
0	成功	AI_SUCCESS	运行正常
1	一般性失败	AI_FAILED	内部错误
2	模型管家未初始化	AI_NOT_INIT	模型管家未初始化
3	非法参数	AI_INVALID_PARA	输入参数非法
4	超时返回	AI_TIMEOUT	执行任务超时返回
7	API 不支持	AI_INVALID_API	设备不支持此 API 接口
8	空指针错误	AI_INVALID_POINTER	类成员函数中 this 指针为空

