HiAI DDK V320

Model Inference and Integration Guide

Issue 04

Date 2020-02-29

HUAWEI TECHNOLOGIES CO., LTD.

Huawei HiAI Application

Send an application email to developer@huawei.com.

Email subject: HUAWEI HiAI + Company name + Product name

Email body: Cooperation company + Contact person + Phone number + Email address

We will reply to you within 5 working days.

Official website: https://developer.huawei.com/consumer/en/

About This Document

Purpose

This document introduces the model inference and integration operations using HiAI DDK V320, and describes the involved APIs and error codes.

This document is used in conjunction with the following documents:

- Huawei HiAI DDK V320 Quick Start
- Huawei HiAI DDK V320 Operator Specifications

Change History

Changes between document issues are cumulative. The latest document issue contains all the changes made in earlier issues.

Date	Version	Change Description
2020-02-29	04	Added the description of APIs for creating model tensors.
2019-12-31	03	Added the description of HiAI DDK V320.
2019-11-22	02	Updated the model integration code.
2019-09-04	01	Added the description of HiAI DDK V310.

Contents

Tables

1 Introduction

AI apps for image recognition and classification require pre-trained models to perform inference. This document describes how to use the Huawei DDK to build AI apps and integrate models during app building.

1.1 Dependencies

- Ubuntu 16.04 development environment (Windows 10 or macOS)
- Trained Caffe or TensorFlow model
- Android Studio for app development
- Device powered by the Kirin SoC

1.2 Supported Operators

For details, see the *Huawei HiAI DDK V320 Operator Specifications.*

1.3 Integration Procedure

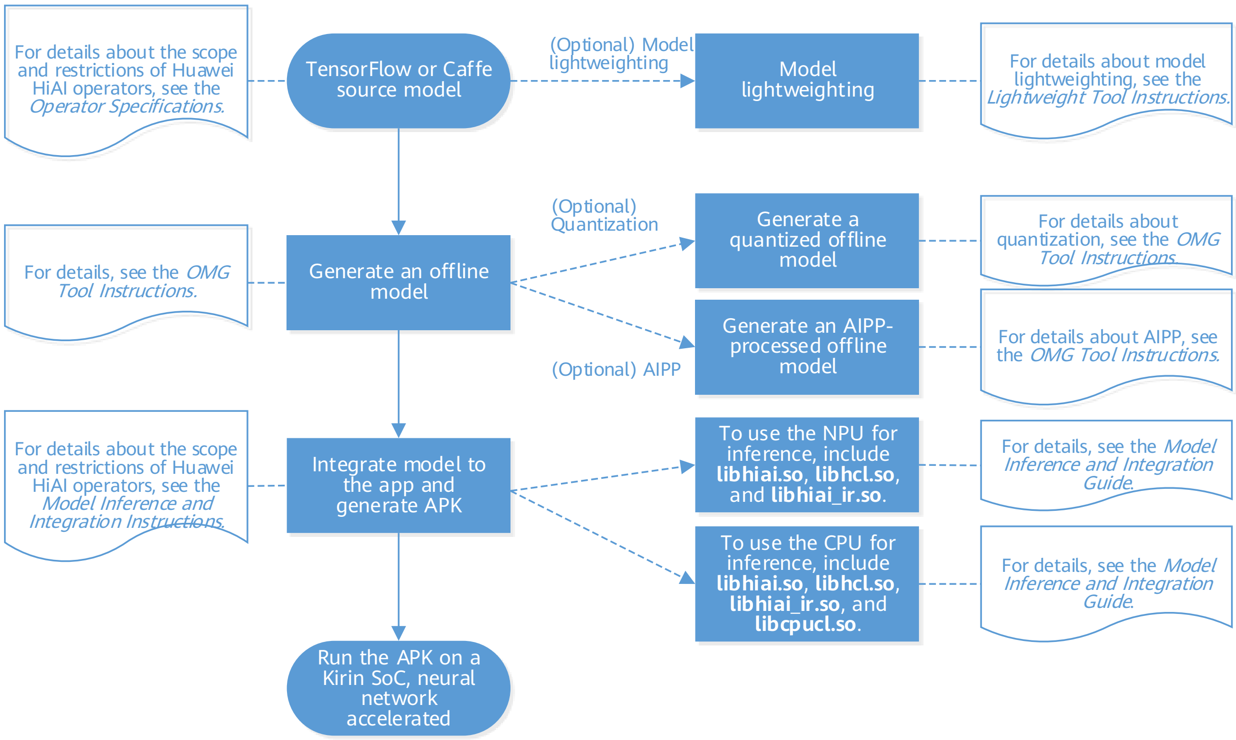
Figure 1-1 shows the app integration procedure with HiAI DDK V320. AI pre-processing (AIPP) and quantization are optional.

NOTE

Obtain the required versions of the documents mentioned in Figure 1-1.

Figure 1-1 describes the HiAI DDK integration procedure.

Figure 1-1 NPU/CPU Integration procedure



Model Lightweighting

Use the lightweight tool to optimize the source model (TensorFlow or Caffe), reducing the model size and accelerating model inference. The non-training mode and retraining mode are supported. For details, see the *Huawei HiAI DDK V320 Lightweight Tool Instructions*.

Offline Model Generation

Convert a Caffe or TensorFlow model into the model format supported by the HiAI platform. You can also perform AIPP and quantization as required:

- AIPP

  AI pre-processing (AIPP) involves image resize, color space conversion (CSC), and mean subtraction and multiplication factor (pixel changing). Data adaptation can be achieved simply by configuring the AIPP parameters or calling the AIPP APIs in the software, sparing the trouble of retraining data that matches the inference platform. With the dedicated hardware, considerable inference performance benefits can be yielded. For details, see the *Huawei HiAI DDK V320 OMG Tool Instructions*.

- Quantization

  Quantization is used to directly convert an FP32 model into a lower-bit model to save network storage space, reduce transfer latency, and improve computation efficiency. For details, see the *Huawei HiAI DDK V320 OMG Tool Instructions*.

App Integration

App integration includes model pre-processing, model loading, model execution, and model post-processing.

- In the NPU scenario, include **libhiai.so**, **libhcl.so**, and **libhiai_ir.so** during model preprocessing. After the APK is built, the app can perform inference on the NPU. For details, see the *Huawei HiAI DDK V320 Model Inference Integration Guide*.
- In the CPU scenario, include **libhiai.so**, **libhcl.so**, **libhiai_ir.so**, and **libcpucl.so** during model preprocessing. After the APK is built, the app can perform inference on the CPU. For details, see the *Huawei HiAI DDK V320 Model Inference Integration Guide*.

2 App Integration

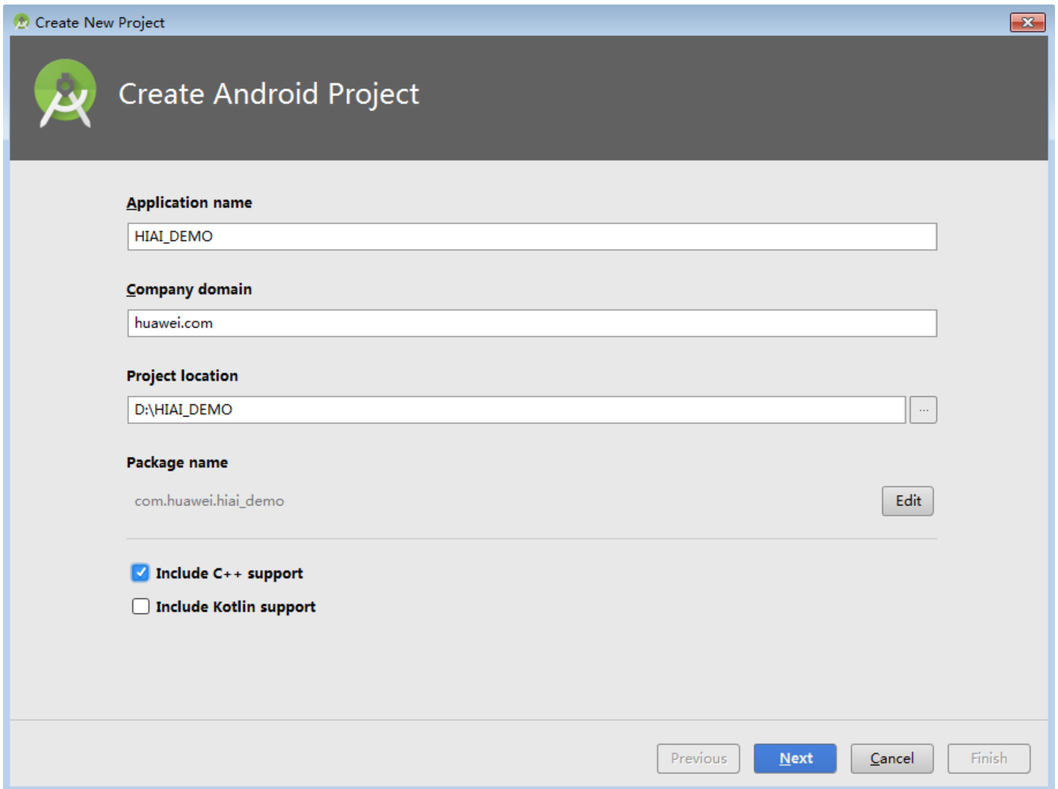
This section uses the Caffe SqueezeNet model as an example to describe app integration.

NOTE

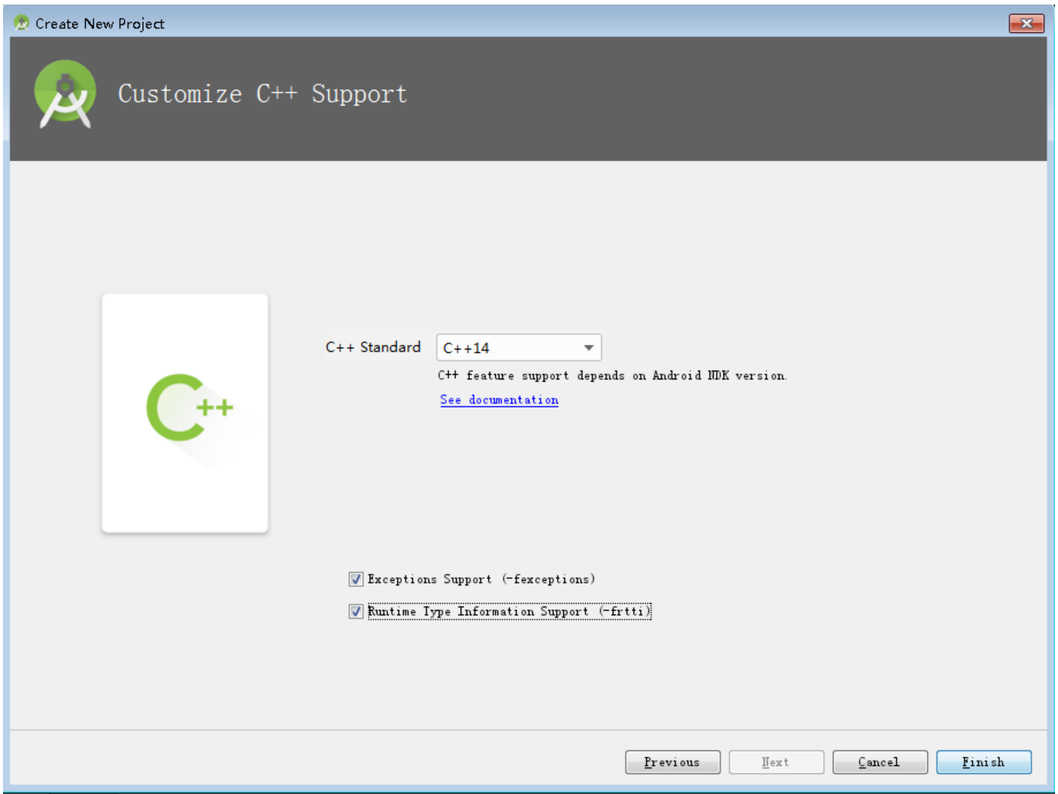
The CPU scenario is newly supported by HiAI DDK V320. For details about the operation differences with the NPU scenario, see section 2.2.3 "Copying the SDK .so File to the Resource Library."

2.1 Project Creation

1. In Android Studio, create a project. Make sure **Include C++ support** is selected.



2. Select **C++14** from the **C++ Standard** drop-down list box. Select **Exceptions Support (-fexceptions)** and **Runtime Type Information Support (-frtti)**.



2.2 JNI Compilation

To compile the Java native interfaces (JNIs), you need to package **libhiai.so** and **libhiaijni.so** into an APK file.

2.2.1 Compiling the Android.mk File

Compile code files **mix_classify_jni.cpp**, **mix_classify_async_jni.cpp**, and **mixbuildmodel.cpp** for JNI implementation, copy the three files to the **jni** directory of the DDK, and compile the **Android.mk** file as follows:

```
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)
```

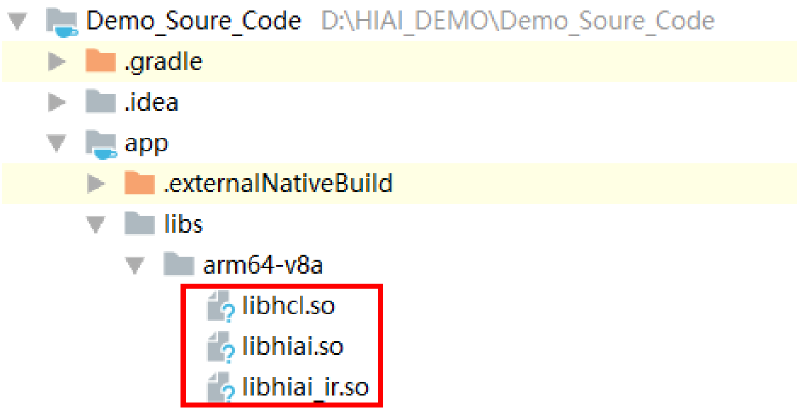
2.2.2 Creating the Application.mk File

In the **jni** directory, create the **Application.mk** file as follows:

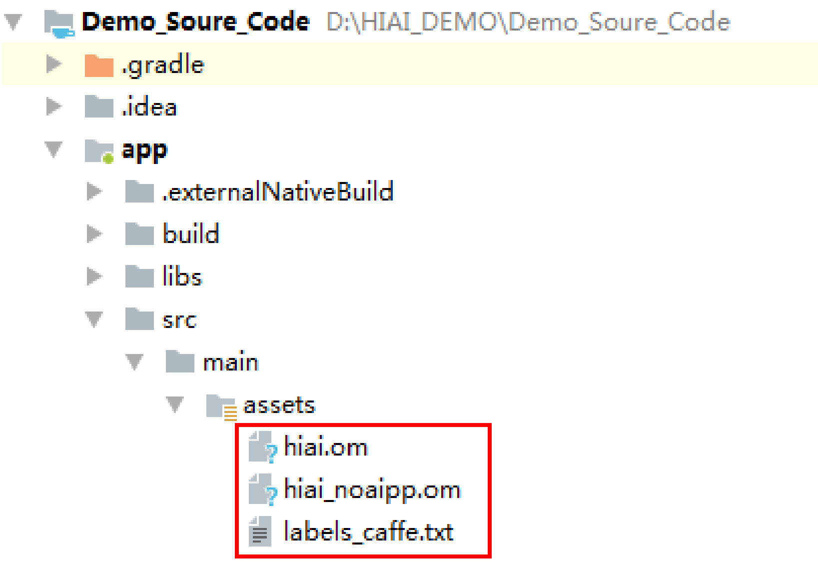
```
APP_ABI := arm64-v8a
APP_STL := c++_shared
```

2.2.3 Copying the SDK .so File to the Resource Library

To use the NPU for inference, copy **libhiai.so**, **libhcl.so**, and **libhiai_ir.so** in the SDK to the **/libs/arm64-v8a** directory of Android Studio.



Copy the OM and label file (.om file and **labels_caffe.txt** in the Android source code directory **src/main/assets/** in the DDK) to the **/src/main/assets** directory of the created project.

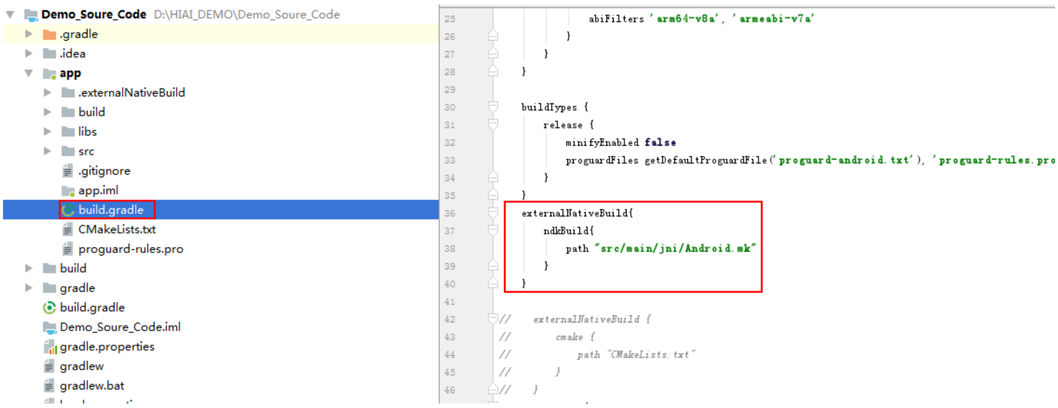


NOTE

To use the CPU for inference, copy **libcpucl.so** to the **/libs/arm64-v8a** directory of Android Studio.

2.2.4 Editing the build.gradle File

Specify the NDK C++ compilation file by adding the following NDK compilation information to the **/src/build.gradle** file.



2.3 Model Integration

2.3.1 Overview

The model inference integration includes model pre-processing and model inference. The DDK provides sync and async APIs.

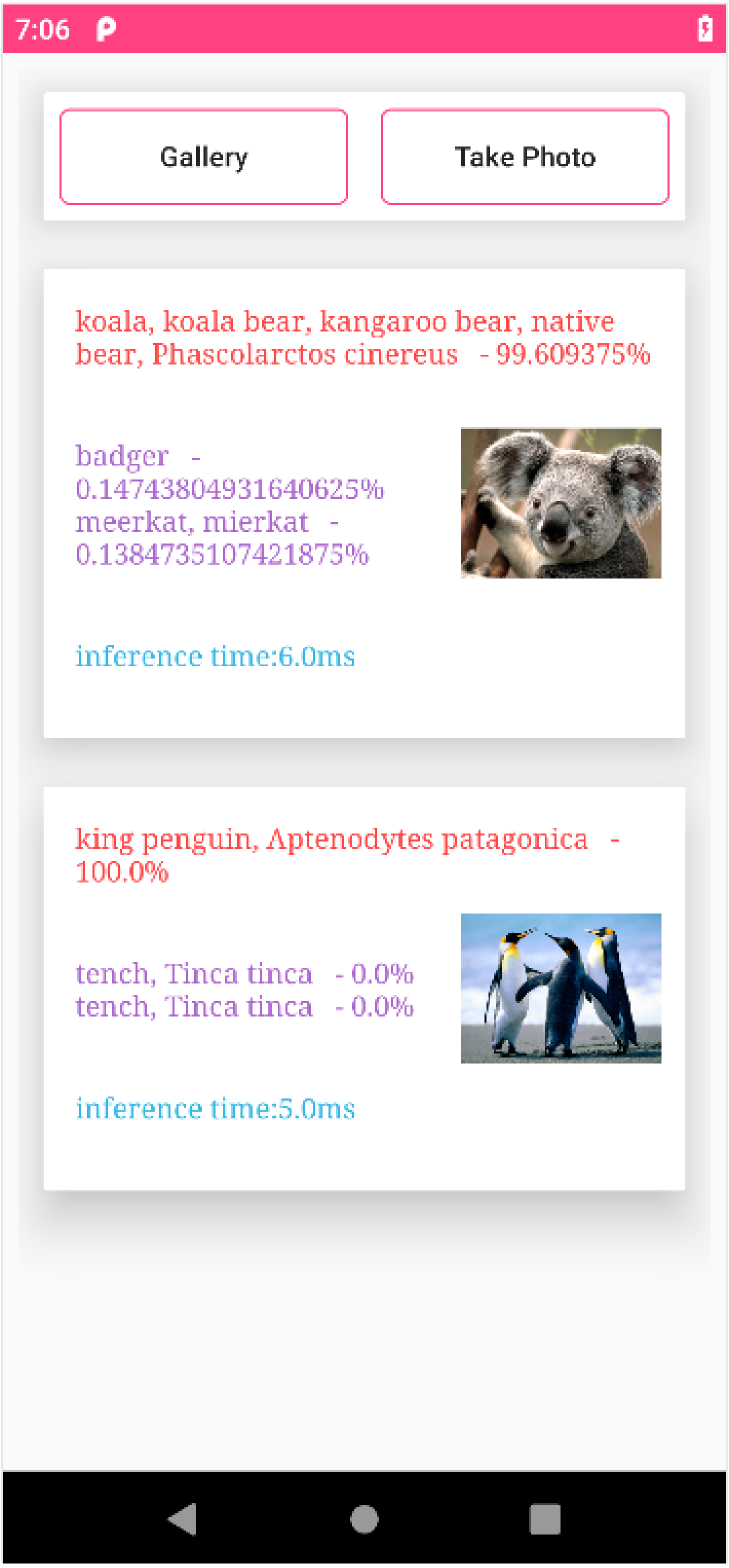
This section describes how to use a single model in sync and async modes and how to implement and call sync and async APIs in each step at the DDK, JNI, and app layers. For details about the code, see the DDK demo. For details about the APIs, see 3 Model Inference APIs.

In the demo:

- Sync mode:
  - Code file of the app layer: **SyncClassifyActivity.java**
  - Code file of the JNI layer: **classify_jni.cpp**
- Async mode:
  - Code file of the app layer: **AsyncClassifyActivity.java**
  - Code file of the JNI layer: **classify_async_jni.cpp**

The demo supports classification of images in the gallery or taken using the camera. Figure 2-1 shows the app UI of the demo APK.

Figure 2-1 UI of the demo APK



2.3.2 Model Pre-Processing

Model pre-processing involves the following features:

- Obtains the HiAI DDK version to determine whether the NPU is supported.
- Determines whether an OM can run on the current HiAI version.
- Tries version compatibility by using the online compilation APIs in model inference, if the OM is incompatible with the current HiAI version.

Model Pre-Processing at the App Layer

Call the **modelCompatibilityProcessFromFile** function at the JNI layer. Remove the model compatibility issue before model inference and determine whether the current model can run on the NPU.

For details about the app-layer implementation code, see the **modelCompatibilityProcess** function in the **app_sample\inference_demo\Demo_Soure_Code\app\src\main\java\com\huawei\hiaidemo\view\MainActivity.java** file.

Model Pre-Processing at the JNI Layer

The JNI layer checks whether the input OM can run on the current device NPU by using APIs for obtaining the HiAI version, model compatibility check, and online compilation.

Step 1 Create an **AiModelMngerClient** object.

Step 2 Create an **AiModelBuilder** object.

Step 3 Obtain the Membuffer by using the **InputMemBufferCreate** function of the **builder** object and passing the offline model path.

Step 4 Call the **SetModelBuffer** function of the **AiModelDescription** object to set the model buffer.

Step 5 Call the **CheckModelCompatibility** function of the **AiModelMngerClient** object to check the compatibility.

Step 6 Check whether the compatibility requirements are met. If yes, exits. If no, build the offline model online to generate a compatible offline model.

----End

For details about the JNI-layer implementation code, see the **Java_com_huawei_hiaidemo_utils_ModelManager_modelCompatibilityProcessFromFile** function in the **app_sample\inference_demo\Demo_Soure_Code\app\src\main\jni\buildmodel.cpp** file.

2.3.3 Model Inference

2.3.3.1 Sync Mode

The Profiling tool can be used to analyze the model inference performance in synchronous mode. For details, see *Huawei HiAI DDK V320 System Debug Tool Instructions*.

App Layer Implementation

Obtain the model input from images by using the **getPixels** function, and then call the **runModelSync** function at the JNI layer to perform model inference.

After the inference is complete, call the **postPost** function to perform post-processing and output the result to the app UI.

For details about the implementation code, see the **runModel** function in the **app_sample\inference_demo\Demo_Soure_Code\app\src\main\java\com\huawei\hiaidemo\view\SyncClassifyActivity.java** file. This function calls the **runModelSync** function in the **ModelManager.java** file.

JNI Layer Implementation

The procedure is as follows:

Step 1 Call **env->GetByteArrayElements(buf_, nullptr)** and **env->GetArrayLength(buf_)** to obtain the data and size of the input to the app layer.

Step 2 Call **env->GetObjectClass(modelInfo)** to obtain the model information.

Step 3 Call the model manager function **LoadModelSync** to create a model manager and load the model.

Step 4 Call the **GetModelIOTensorDim** function to obtain the input and output shapes of the model, and initialize **input_tensor** and **output_tensor** by using the **init** function in the class AiTensor.

Step 5 Copy **dataBuff** passed by the app layer to **input_tensor** by using the memcpy function.

Step 6 Call the **Process** function in class **AiModelMngerClient** to perform model inference.

Step 7 Obtain the inference result from **output_tensor**.

----End

For details about the JNI-layer implementation code, see the **Java_com_huawei_hiaidemo_utils_ModelManager_runModelSync** function in the **app_sample\inference_demo\Demo_Soure_Code\app\src\main\jni\classify_jni.cpp** file.

2.3.3.2 Async Mode

App Layer Implementation

Call the **getPixels** function to obtain the model input from the image, and create and register the **ModelManagerListener** object. Then, call the **runModelAsync** function at the JNI layer to run the model in async mode.

For details about the implementation code, see the **runModel** function in the **app_sample\inference_demo\Demo_Soure_Code\app\src\main\java\com\huawei\hiaidemo\view\AsyncClassifyActivity.java** file. This function calls the **runModelAsync** function in the **ModelManager.java** file.

JNI Layer Implementation

Model execution in async mode is the same as that in sync mode (see JNI Layer Implementation). Note that the async model management engine is used when the Process function in class **AiModelMngerClient** is called.

Post-processing is implemented through the callback function **OnProcessDone**. To use the async model manager engine, you need to call the **Init** function when creating an **AiModelMngerClient** object, and initialize the **ModelManagerListener** listener object. For details about the implementation code, see the **Java_com_huawei_hiaidemo_utils_ModelManager_runModelAsync** function in the **app_sample\inference_demo\Demo_Soure_Code\app\src\main\jni\classify_async_jni.cpp** file.

2.3.3.3 APIs

The main API function prototypes used in DDK model inference are as follows:

```
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);
```

For details about the parameters, see 3 Model Inference APIs.

3 Model Inference APIs

3.1 Model Manager

NOTICE

- The model manager object is not thread-safe. Use one of these instances per thread.
- A maximum of 256 model manager instances can be created for an app.
- The load API can be called only once for each model manager instance. A maximum of 63 models can be loaded in sync mode. A maximum of 32 models can be loaded in async mode.
- In async reference scenarios, the maximum length of a task queue of the model reference API is 512.

3.1.1 Obtaining the HiAI Version

Table 3-1 GetVersion API

Description	Obtains the HiAI version in the phone ROM.
API Prototype	char * GetVersion();
Parameter	None
Return Value	If the execution succeeds, the HiAI version is returned. The returned version is in the following format: **<major>.<middle>.<minor>.<point>** • **<major>**: product form. **1XX** indicates the mobile phone form. **2XX** indicates the edge computing form. **3XX** indicates the cloud form. • **<middle>**: three-digit number, indicating the V version number of the product form, for example, **100**, **200**, and **300** in HiAI V100, HiAI V200, and HiAI V300 of the mobile phone form. • **<minor>**: three-digit number, indicating the incremental C version number, where a major feature is newly supported • **< point >**: three-digit number, indicating the B version or patch version. If the last digit is not **0**, it indicates a patch version. For example, if the version returned by the phone ROM is **100.300.010.010** while **nullptr** or **000.000.000.000** is returned, the version does not support NPU acceleration. For details about the HiAI version, see the *Huawei HiAI DDK V320 Terminology and Conventions*.

3.1.2 Creating a Model Manager

Table 3-2 Init API

Description	Creates a model manager.
API Prototype	AIStatus Init(shared_ptr<AiModelManagerClientListener> listener);
Parameter	[Input] **listener**: listening API. **nullptr** indicates a sync model manager, while other values indicate an async model manager.
Return Value	**AIStatus::AI_SUCCESS** indicates a success, while other values indicate a failure.

3.1.3 Loading a Model

Table 3-3 Load API

Description	Loads a model.
API Prototype	AIStatus Load(vector<shared_ptr<AiModelDescription>> &model_desc);
Parameter	[Input] **model_desc**: model description array. One or more models can be loaded. Each model name must be unique.
Return Value	**AIStatus::AI_SUCCESS** indicates a success, while other values indicate a failure.

3.1.4 Inferring a Model

Table 3-4 Process API

Description	Performs model inference.
API Prototype	AIStatus Process(AiContext &context, vector<shared_ptr<AiTensor>> &input_tensor, vector<shared_ptr<AiTensor>> &output_tensor, uint32_t timeout, int32_t &iStamp);
Parameter	[Input] **context**: model running context, which must contain the **model_name** field [Input] **input_tensor**: Tensor information of the model input node [Input/Output] **output_tensor**: Tensor information of the model output node [Input] **timeout**: inference timeout period (ms) [Output] **iStamp**: async return flag, as the basis in conjunction with the model name for callback indexing
Return Value	**AIStatus::AI_SUCCESS** indicates a success, while other values indicate a failure.

3.1.5 Checking Model Compatibility

Table 3-5 CheckModelCompatibility API

Description	Checks model compatibility.
API Prototype	AIStatus CheckModelCompatibility(AiModelDescription &model_desc, bool &isModelCompatibility);
Parameter	[Input] **model_desc**: model description [Output] **isModelCompatibility**: compatibility check flag. The value **True** indicates that the model compatibility check is passed, and the value **False** indicates that the model compatibility check fails.
Return Value	**AIStatus::AI_SUCCESS** indicates a success, while other values indicate a failure.

3.1.6 Obtaining Input and Output Tensor Information of a Model

Table 3-6 GetModelIOTensorDim API

Description	Obtains the input and output tensor information of a model.
API Prototype	AIStatus GetModelIOTensorDim(const string& model_name, vector<TensorDimension>& input_tensor, vector<TensorDimension>& output_tensor);
Parameter	[Input] **model_name**: model name [Output] **input_tensor**: Tensor information of the model input node [Output] **output_tensor**: Tensor information of the model output node
Return Value	**AIStatus::AI_SUCCESS** indicates a success, while other values indicate a failure.

3.1.7 Uninstalling a Model

Table 3-7 UnLoadModel API

Description	Uninstalls a model.
API Prototype	AIStatus UnLoadModel();
Parameter	None
Return Value	**AIStatus::AI_SUCCESS** indicates a success, while other values indicate a failure.

NOTE

It is recommended that this API be used in pair with **Load** to free the memory for model loading. If **Load** is repeatedly called but **UnLoadModel** is not used, a large amount of memory will be occupied.

3.2 Model Building

3.2.1 Building an OM Online

Table 3-8 BuildModel API

Description	Compiles an OM online.
API Prototype	AIStatus BuildModel(const vector<MemBuffer *> &input_membuffer, MemBuffer *output_model_buffer, uint32_t &output_model_size);
Parameter	[Input] **input_membuffer**: input OM buffer [Output] **output_model_buffer**: output model buffer [Output] **output_model_size**: output model size, in bytes
Return Value	**AIStatus::AI_SUCCESS** indicates a success, while other values indicate a failure.

NOTE

You are advised to save the built model file (**output_model_buffer**) locally to improve the model loading performance.

Table 3-9 BuildModel API

Description	Builds an OM online.
API Prototype	AIStatus BuildModel(const vector<MemBuffer *> &input_membuffer, MemBuffer *output_model_buffer, uint32_t &output_model_size);
Parameter	[Input] **input_membuffer**: input OM buffer [Output] **output_model_buffer**: output model buffer [Output] **output_model_size**: output model size, in bytes
Return Value	**AIStatus**: **AI_SUCCESS** indicates a success, while other values indicate failures.

NOTE

You are advised to save the built model file (**output_model_buffer**) locally to improve the model loading performance.

3.2.2 Reading the Prototxt Information of an OM from a File

Table 3-10 ReadBinaryProto API

Description	Reads the prototxt information of an OM from a file.
API Prototype	MemBuffer* ReadBinaryProto(const string path);
Parameter	[Input] **path**: model file path
Return Value	MemBuffer address. **nullptr** indicates that MemBuffer creation fails.

3.2.3 Reading the Prototxt Information of an OM from the Memory

Table 3-11 ReadBinaryProto API

Description	Reads the prototxt information of an OM from the memory.
API Prototype	MemBuffer* ReadBinaryProto(void* data, uint32_t size);
Parameter	[Input] **data**: MemBuffer address of the OM [Output] **size**: size of the OM memory, in bytes
Return Value	MemBuffer address. **nullptr** indicates that MemBuffer creation fails.

3.2.4 Creating MemBuffer for an OM from the Memory

Table 3-12 InputMemBufferCreate API

Description	Creates MemBuffer for an OM from the memory.
API Prototype	MemBuffer* InputMemBufferCreate(void *data, uint32_t size);
Parameter	[Input] **data**: MemBuffer address of the OM [Output] **size**: size of the OM memory, in bytes
Return Value	MemBuffer address. **nullptr** indicates that MemBuffer creation fails.

3.2.5 Creating MemBuffer for an OM from a File

Table 3-13 InputMemBufferCreate API

Description	Creates MemBuffer for an OM from a file.
API Prototype	MemBuffer* InputMemBufferCreate(const string path);
Parameter	[Input] **path**: model file path
Return Value	MemBuffer address. **nullptr** indicates that MemBuffer creation fails.

3.2.6 Creating MemBuffer for Compiling an OM Online

Table 3-14 OutputMemBufferCreate API

Description	Creates MemBuffer for compiling an OM online.
API Prototype	MemBuffer* OutputMemBufferCreate(const int32_t framework, const vector<MemBuffer *> &input_membuffer);
Parameter	[Input] **framework**: model platform [Input] **input_membuffer**: MemBuffer of the input OM
Return Value	MemBuffer address. **nullptr** indicates that MemBuffer creation fails.

3.2.7 Destroying MemBuffer

Table 3-15 MemBufferDestroy API

Description	Destroys applied MemBuffer.
API Prototype	void MemBufferDestroy(MemBuffer *membuf);
Parameter	[Input] **membuf**: applied MemBuffer
Return Value	None

3.2.8 Exporting an OM Compiled Online to a File

Table 3-16 MemBufferExportFile API

Description	Exports data in the model buffer to a file.
API Prototype	AIStatus MemBufferExportFile(MemBuffer *membuf, const uint32_t build_size, const string build_path);
Parameter	[Input] **membuf**: memory pointer for storing the OM information [Input] **build_size**: OM size, in bytes [Input] **build_path**: path of the exported OM file
Return Value	**AIStatus::AI_SUCCESS** indicates a success, while other values indicate a failure.

3.3 Model Description

3.3.1 Obtaining the Model Name

Table 3-17 GetName API

Description	Obtains the model name.
API Prototype	string GetName() const;
Parameter	None
Return Value	Model name string. **NULL** indicates that the model name fails to be obtained.

3.3.2 Obtaining the Model Buffer

Table 3-18 GetModelBuffer API

Description	Obtains the model buffer.
API Prototype	void* GetModelBuffer() const;
Parameter	None
Return Value	void *

3.3.3 Setting the Model Buffer

Table 3-19 SetModelBuffer API

Description	Sets the model buffer.
API Prototype	AIStatus SetModelBuffer(const void* data, uint32_t size);
Parameter	[Input] **data**: model buffer address [Input] **size**: model size, in bytes
Return Value	**AIStatus::AI_SUCCESS** indicates a success, while other values indicate a failure.

3.3.4 Obtaining the Model Loading Frequency

Table 3-20 GetFrequency API

Description	Obtains the model loading frequency.
API Prototype	int32_t GetFrequency() const;
Parameter	None
Return Value	Enumerated value of 3.10.1 AiModelDescription_Frequency.

3.3.5 Obtaining the Model Framework

Table 3-21 GetFramework API

Description	Obtains the model framework.
API Prototype	int32_t GetFramework() const;
Parameter	None
Return Value	Enumerated value of 3.10.3 AiModelDescription_Framework.

3.3.6 Obtaining the Model Type

Table 3-22 GetModelType API

Description	Obtains the model type.
API Prototype	int32_t GetModelType() const;
Parameter	None
Return Value	Enumerated value of 3.10.4 AiModelDescription_ModelType

3.3.7 Obtaining the Device Type

Table 3-23 GetDeviceType API

Description	Obtains the device type.
API Prototype	int32_t GetDeviceType() const;
Parameter	None
Return Value	Enumerated value of 3.10.2 AiModelDescription_DeviceType

3.3.8 Obtaining the Model Size

Table 3-24 GetModelNetSize API

Description	Obtains the Model size (in bytes).
API Prototype	uint32_t GetModelNetSize() const;
Parameter	None
Return Value	Model size of the uint32_t type

3.4 Tensor Creation

3.4.1 Initializing a Tensor

Table 3-25 Initializing a Tensor based on its size

Description	Initializes a Tensor based on its size.
API Prototype	AIStatus Init(const TensorDimension* dim);
Parameter	[Input] **dim**: size and structure of the input tensor
Return Value	**AIStatus::AI_SUCCESS** indicates a success. **AIStatus::AI_INVALID_PARA** indicates that **dim** is null. **AIStatus::AI_FAILED** indicates that the Tensor size is 0, or memory allocation fails.

NOTE

APIs described in Table 3-26 and Table 3-27 are newly added in HiAI DDK V310.

APIs described in Table 3-26 apply to AIPP models only.

Table 3-26 Initializing a Tensor based on the NHW and image format

Description	Initializes a Tensor based on the NHW and image format.
API Prototype	AIStatus Init(uint32_t number, uint32_t height, uint32_t width, AiTensorImage_Format format);
Parameter	[Input] **number**: number of Tensors [Input] **height**: height of the input Tensor [Input] **width**: width of the input Tensor [Input] **format**: input image format (AiTensorImage_Format) The options are as follows: • AiTensorImage_YUV420SP_U8 • AiTensorImage_XRGB8888_U8 • AiTensorImage_YUV400_U8 • AiTensorImage_ARGB8888_U8 • AiTensorImage_YUYV_U8 • AiTensorImage_YUV422SP_U8 • AiTensorImage_AYUV444_U8
Return Value	**AIStatus::AI_SUCCESS** indicates a success. **AIStatus::AI_INVALID_PARA** indicates that either N, H, or W is 0, or **format** is invalid. **AIStatus::AI_FAILED** indicates that the Tensor size is 0, or memory allocation fails.

Table 3-27 Initializing a Tensor based on its size and data type

Description	Initializes a Tensor based on its size and data type.
API Prototype	AIStatus Init(const TensorDimension* dim, HIAI_DataType DataType);
Parameter	[Input] **dim**: size and structure of the input Tensor [Input] **DataType**: data type. The options are: **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**
Return Value	**AIStatus::AI_SUCCESS** indicates a success. **AIStatus::AI_INVALID_PARA** indicates that either **dim** is null or **DataType** is invalid. **AIStatus::AI_FAILED** indicates that the Tensor size is 0, or memory allocation fails.

Table 3-28 Initializing a Tensor based on NativeHandle, TensorDimension, and DataType

Description	Initializes a Tensor based on **NativeHandle**, **TensorDimension**, and **DataType.**
API Prototype	AIStatus Init(const NativeHandle& handle, const TensorDimension* dim, HIAI_DataType pdataType);
Parameter	[Input] **handle**: structure of **NativeHandle** **NativeHandle** supports only the ION memory. [Input] **dim**: dimensions of the input tensor [Input] **DataType**: data type. The options are: **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**, and **HIAI_DATATYPE_DOUBLE**
Return Value	**AIStatus::AI_SUCCESS**: success **AIStatus::AI_INVALID_PARA** indicates that **dim** is null or **DataType** is not supported. **AIStatus::AI_FAILED**: The tensor size is 0, or memory allocation fails.

3.4.2 Obtaining the Tensor Address

Table 3-29 GetBuffer API

Description	Obtains the tensor buffer address.
API Prototype	void *GetBuffer() const;
Parameter	None
Return Value	void *

3.4.3 Obtaining the Tensor Size

Table 3-30 GetSize API

Description	Obtains the buffer size of the tensor (in bytes).
API Prototype	uint32_t GetSize() const;
Parameter	None
Return Value	Tensor size of the uint32_t type

3.4.4 Setting the Tensor Dimensions

Table 3-31 SetTensorDimension API

Description	Sets the tensor dimensions.
API Prototype	AIStatus SetTensorDimension(const TensorDimension* dim);
Parameter	[Input] **dim**: dimensions of the input tensor
Return Values	**AIStatus::AI_SUCCESS**: success **AIStatus::AI_INVALID_PARA**: The **dim** parameter is null. **AIStatus::AI_FAILED**: The tensor size is 0, or memory allocation fails.

3.4.5 Obtains the tensor dimensions.

Table 3-32 GetTensorDimension API

Description	Obtains the tensor dimensions.
API Prototype	TensorDimension GetTensorDimension() const;
Parameter	None
Return Value	Tensor dimensions

3.4.6 Obtaining the TensorBuffer Address

Table 3-33 GetTensorBuffer API

Description	Obtains the address of TensorBuffer.
API Prototype	void *GetTensorBuffer() const;
Parameter	None
Return Value	void*

3.5 External AIPP APIs

3.5.1 General

3.5.1.1 Obtaining Model AIPP Configurations

Table 3-34 AiModelMngerClient::GetModelAippPara API

Description	Obtains model AIPP configurations.
API Prototype	AIStatus GetModelAippPara(const std::string& modelName, std::vector<std::shared_ptr<AippPara>>& aippPara);
Parameter	[Input] **modelName**: model name [Output] **AippPara**: model AIPP configuration parameters, which are saved in the **AippPara** object
Return Value	**AI_SUCCESS** indicates a success, while other values indicate a failure.

3.5.1.2 Obtaining AIPP Configurations of a Model Input

Table 3-35 AiModelMngerClient::GetModelAippPara API

Description	Obtains AIPP configurations of an input in a model.
API Prototype	AIStatus GetModelAippPara(const std::string& modelName, uint32_t index, std::vector<std::shared_ptr<AippPara>>& aippPara);
Parameter	[Input] **modelName**: model name [Input] **index**: sequence number of an input. The value must be greater than or equal to **0**. [Output] **AippPara**: AIPP configuration parameters corresponding to the model input **index**, which are saved in the **AippPara** object
Return Value	**AI_SUCCESS** indicates a success, while other values indicate a failure.

3.5.1.3 Obtaining the Tensor Buffer Address

Table 3-36 AippTensor::GetBuffer API

Description	Obtains the Tensor buffer address.
API Prototype	void* GetBuffer() const
Parameter	None
Return Value	void * tensor buffer address

3.5.1.4 Obtaining the Tensor Buffer Size

Table 3-37 AippTensor::GetSize API

Description	Obtains the Tensor buffer address.
API Prototype	uint32_t GetSize() const;
Parameter	None
Return Value	Memory size of the tensor buffer

3.5.1.5 Obtaining the Tensor Pointer

Table 3-38 AippTensor::GetAiTensor API

Description	Obtainis the Tensor pointer.
API Prototype	std::shared_ptr<AiTensor> GetAiTensor() const;
Parameter	None
Return Value	Tensor pointer

3.5.1.6 Initializing the AippPara Object

Table 3-39 AippPara::Init API

Description	Initializes the AippPara object.
API Prototype	AIStatus Init(uint32_t batchCount);
Parameter	[Input] **batchCount**: batch size of the model input, corresponding to the **N** in NCHW. The default value is **1**.
Return Value	**AI_SUCCESS** indicates a success, while other values indicate a failure.

3.5.1.7 Obtaining All Pointers to an AippPara Object

Table 3-40 AippTensor::GetAippParas API

Description	Obtains the pointers to an AippPara object.
API Prototype	std::vector<std::shared_ptr<AippPara>> GetAippParas() const;
Parameter	None
Return Value	Vector with all pointers to an AippPara object

3.5.1.8 Obtaining the Pointer to a Specified AippPara Object

Table 3-41 AippTensor::GetAippParas API

Description	Obtains the pointer to a specified AippPara object.
API Prototype	std::shared_ptr<AippPara> GetAippParas(uint32_t index) const;
Parameter	**index**: sequence number of an input. The value must be greater than or equal to **0**.
Return Value	Pointer to the AippPara object

3.5.1.9 Obtaining the Batch Size

Table 3-42 AippPara::GetBatchCount API

Description	Obtains the batch size.
API Prototype	uint32_t GetBatchCount();
Parameter	None
Return Value	Batch size in the **AippPara** object

3.5.1.10 Setting inputIndex

Table 3-43 AippPara::SetInputIndex API

Description	Sets the AIPP **inputIndex** parameter.
API Prototype	AIStatus SetInputIndex(uint32_t inputIndex);
Parameter	**inputIndex**: sequence number of the input on which the AIPP configurations apply. The value must be greater than or equal to **0**.
Return Value	**AI_SUCCESS** indicates a success, while other values indicate a failure.

3.5.1.11 Obtaining inputIndex

Table 3-44 AippPara::GetInputIndex API

Description	Obtains the AIPP **inputIndex** value, which identifies the model input on which the AIPP configurations apply.
API Prototype	int32_t GetInputIndex();
Parameter	None
Return Value	**inputIndex** value

3.5.1.12 Setting inputAippIndex

Table 3-45 AippPara::SetInputAippIndex API

Description	Sets the AIPP inputAippIndex parameter.
API Prototype	AIStatus SetInputAippIndex(uint32_t inputAippIndex);
Parameter	**inputAippIndex**: sequence number of an output of the Data operator on which the AIPP configurations apply in the scenario where the Data operator has more than one output. The value must be greater than or equal to **0**.
Return Value	**AI_SUCCESS** indicates a success, while other values indicate a failure.

3.5.1.13 Obtaining inputAippIndex

Table 3-46 AippPara::GetInputAippIndex API

Description	Obtains the AIPP **inputAippIndex** value, which identifies the output of the Data operator on which the AIPP configurations apply in the scenario where the Data operator has more than one output.
API Prototype	int32_t GetInputAippIndex();
Parameter	None
Return Value	**inputAippIndex** value

3.5.1.14 Obtaining the Input Image Size

Table 3-47 AippPara::GetInputShape API

Description	Obtains the AIPP input image size.
API Prototype	AippInputShape GetInputShape();
Parameter	None
Return Value	Image size (**AippInputShape** structure)

3.5.1.15 Obtaining the Input Image Format

Table 3-48 AippPara::GetInputFormat API

Description	Obtains the AIPP input image format.
API Prototype	AiTensorImage_Format GetInputFormat();
Parameter	None
Return Value	Input image format

3.5.1.16 Obtaining CSC Parameters

Table 3-49 AippPara::GetCscPara API

Description	Obtains the AIPP CSC parameters.
API Prototype	AippCscPara GetCscPara();
Parameter	None

Return Value	CSC parameters
	Reference:
	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 Obtaining Channel Swapping Parameters

Table 3-50 AippPara::GetChannelSwapPara API

Description	Obtains AIPP channel swapping parameters.
API Prototype	AippChannelSwapPara GetChannelSwapPara();
Parameter	None
Return Value	Channel swapping parameters

3.5.1.18 Obtaining Image Cropping Parameters

Table 3-51 AippPara::GetCropPara API

Description	Obtains AIPP cropping parameters.
API Prototype	AippCropPara GetCropPara(uint32_t batchIndex);
Parameter	**batchIndex**: sequence number of the batch whose cropping parameters are to be obtained. The value must be greater than or equal to **0**.
Return Value	Cropping parameter. If **batchIndex** exceeds **batchCount** of **AippPara**, the value of **AippCropPara** of the model is returned.

3.5.1.19 Obtaining Image Resizing Parameters

Table 3-52 AippPara::GetResizePara API

Description	Obtains AIPP resizing parameters.
API Prototype	AippResizePara GetResizePara(uint32_t batchIndex);
Parameter	**batchIndex**: sequence number of the batch whose resizing parameters are to be obtained. The value must be greater than or equal to **0**.

Return Value	Resizing parameters
	Reference:
	`struct AippResizePara {`
	`bool switch_ = false;`
	`uint32_t resizeOutputSizeW = 0;`
	`uint32_t resizeOutputSizeH = 0;`
	`};`

3.5.1.20 Obtaining Image Padding Parameters

Table 3-53 AippPara::GetPaddingPara API

Description	Obtains AIPP image padding parameters.
API Prototype	AippPaddingPara GetPaddingPara(uint32_t batchIndex);
Parameter	**batchIndex**: sequence number of the batch whose padding parameters are to be obtained. The value must be greater than or equal to **0**.
Return Value	Padding parameters

3.5.1.21 Obtaining Data Type Conversion Parameters

Table 3-54 AippPara::GetDtcPara API

Description	Obtains AIPP DTC parameters.
API Prototype	AippDtcPara GetDtcPara(uint32_t batchIndex);
Parameter	**batchIndex**: sequence number of the batch whose DTC parameters are to be obtained. The value must be greater than or equal to **0**.
Return Value	Data type conversion parameters

3.5.1.22 Creating AippTensor by Using buffer_handle_t

Table 3-55 HIAI_CreateAiPPTensorFromHandle API

Description	Creates an AIPP tensor based on **buffer_handle_t**, **TensorDimension**, and **imageFormat**.
API Prototype	std::shared_ptr<AippTensor> HIAI_CreateAiPPTensorFromHandle(buffer_handle_t& handle, const TensorDimension* dim, AiTensorImage_Format imageFormat = AiTensorImage_INVALID);
Parameter	[Input] **handle**: buffer_handle_t struct [Input] **dim**: dimensions of the input tensor [Input] **imageFormat**: **ImageFormat** information stored in **handle** The options are: **AiTensorImage_XRGB8888_U8** **AiTensorImage_RGB888_U8** **AiTensorImage_YUV422SP_U8** **AiTensorImage_YUV420SP_U8** **AiTensorImage_YUYV_U8** **AiTensorImage_YUV400_U8**
Return Value	shared_ptr pointer to the AIPP tensor

3.5.2 Dynamic AIPP

3.5.2.1 Setting the Input Image Size

Table 3-56 AippPara::SetInputShape API

Description	Sets the AIPP input image size **inputShape**.

API Prototype	AIStatus SetInputShape(AippInputShape inputShape);
	Reference:
	`struct AippInputShape {`
	`uint32_t srcImageSizeW = 0;`
	`uint32_t srcImageSizeH = 0;`
	`};`

Parameter	**inputShape**: input image size
Return Value	**AI_SUCCESS** indicates a success, while other values indicate a failure.

3.5.2.2 Setting the Input Image Format

Table 3-57 AippPara::SetInputFormat API

Description	Sets the AIPP input image format **inputFormat**.
API Prototype	AIStatus SetInputFormat(AiTensorImage_Format inputFormat);
Parameter	**inputFormat**: input image format. The options are as follows: • **AiTensorImage_YUV420SP_U8** • **AiTensorImage_XRGB8888_U8** • **AiTensorImage_YUV400_U8** • **AiTensorImage_ARGB8888_U8** • **AiTensorImage_YUYV_U8** • **AiTensorImage_YUV422SP_U8** • **AiTensorImage_AYUV444_U8**
Return Value	**AI_SUCCESS** indicates a success, while other values indicate a failure.

3.5.2.3 Setting CSC Parameters

Table 3-58 AippPara::SetCscPara API

Description	Sets AIPP CSC parameters.
API Prototype	AIStatus SetCscPara(AiTensorImage_Format targetFormat, ImageType imageType=JPEG);
Parameter	**targetFormat**: type of the converted image. The system automatically fills the CSC parameters based on the converted image. The options are as follows: AiTensorImage_RGB888_U8, AiTensorImage_BGR888_U8, AiTensorImage_YUV444SP_U8, AiTensorImage_YVU444SP_U8, AiTensorImage_YUV400_U8 **imageType**: image type. The default value is **JPEG**. The options are as follows: JPEG, BT_601_NARROW, BT_601_FULL, BT_709_NARROW
Return Value	**AI_SUCCESS** indicates a success, while other values indicate a failure.

3.5.2.4 Setting Channel Swapping Parameters

Table 3-59 AippPara::SetChannelSwapPara API

Description	Sets AIPP channel swapping parameters.
API Prototype	AIStatus SetChannelSwapPara(AippChannelSwapPara channelSwapPara);

Parameter	**channelSwapPara**: channel swapping parameters
	Reference:
	`struct AippChannelSwapPara {`
	`bool rbuvSwapSwitch = false;`
	`bool axSwapSwitch = false;`
	`};`

Return Value	**AI_SUCCESS** indicates a success, while other values indicate a failure.

3.5.2.5 Setting Image Cropping Parameters Across Batches

Table 3-60 AippPara::SetCropPara API

Description	Sets AIPP image cropping parameters across batches.
API Prototype	AIStatus SetCropPara(AippCropPara cropPara);

Parameter	**cropPara**: cropping parameters
	Reference:
	`struct AippCropPara {`
	`bool switch_ = false;`
	`uint32_t cropStartPosW = 0;`
	`uint32_t cropStartPosH = 0;`
	`uint32_t cropSizeW = 0;`
	`uint32_t cropSizeH = 0;`
	`};`

Return Value	**AI_SUCCESS** indicates a success, while other values indicate a failure.

3.5.2.6 Setting Batch-Specific Image Cropping Parameters

Table 3-61 AippPara::SetCropPara API

Description	Sets AIPP image cropping parameters of a specific batch.
API Prototype	AIStatus SetCropPara(uint32_t batchIndex, AippCropPara cropPara);
Parameter	**batchIndex**: sequence number of the batch on which the cropping parameters apply. The value must be greater than or equal to **0**. **cropPara**: cropping parameters
Return Value	**AI_SUCCESS** indicates a success, while other values indicate a failure.

3.5.2.7 Setting Image Resizing Parameters Across Batches

Table 3-62 AippPara::SetResizePara API

Description	Sets AIPP image resizing parameter across batches.
API Prototype	AIStatus SetResizePara(AippResizePara resizePara);

Parameter	**resizePara**: image resizing parameters
	Reference:
	`struct AippResizePara {`
	`bool switch_ = false;`
	`uint32_t resizeOutputSizeW = 0;`
	`uint32_t resizeOutputSizeH = 0;`
	`};`

Return Value	**AI_SUCCESS** indicates a success, while other values indicate a failure.

3.5.2.8 Setting Batch-Specific Image Resizing Parameters

Table 3-63 AippPara::SetResizePara API

Description	Sets AIPP image resizing parameters of a specific batch.
API Prototype	AIStatus SetResizePara(uint32_t batchIndex, AippResizePara resizePara);
Parameter	**batchIndex**: sequence number of the batch on which the resizing parameters apply. The value must be greater than or equal to **0**. **resizePara**: image resizing parameters
Return Value	**AI_SUCCESS** indicates a success, while other values indicate a failure.

3.5.2.9 Setting Image Padding Parameters Across Batches

Table 3-64 AippPara::SetPaddingPara API

Description	Sets AIPP image padding parameters across batches.
API Prototype	AIStatus SetPaddingPara(AippPaddingPara paddingPara);

Parameter	**paddingPara**: padding parameters
	Reference:
	`struct AippPaddingPara {`
	`bool switch_ = false;`
	`uint32_t paddingSizeTop = 0;`
	`uint32_t paddingSizeBottom = 0;`
	`uint32_t paddingSizeLeft = 0;`
	`uint32_t paddingSizeRight = 0;`
	`};`

Return Value	**AI_SUCCESS** indicates a success, while other values indicate a failure.

3.5.2.10 Setting Batch-Specific Image Padding Parameters

Table 3-65 AippPara::SetPaddingPara API

Description	Sets AIPP image padding parameters of a specific batch.
API Prototype	AIStatus SetPaddingPara(uint32_t batchIndex, AippPaddingPara paddingPara);
Parameter	**batchIndex**: sequence number of the batch on which the padding parameters apply. The value must be greater than or equal to **0**. **paddingPara**: padding parameters
Return Value	**AI_SUCCESS** indicates a success, while other values indicate a failure.

3.5.2.11 Setting DTC Parameters Across Batches

Table 3-66 AippPara::SetDtcPara API

Description	Sets AIPP data type conversion (DTC) parameters across batches.
API Prototype	AIStatus SetDtcPara(AippDtcPara dtcPara);

Parameter	**dtcPara**: DTC parameters
	Reference:
	`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;`
	`};`

Return Value	**AI_SUCCESS** indicates a success, while other values indicate a failure.

3.5.2.12 Setting Batch-Specific Data Type Conversion Parameters

Table 3-67 AippPara::SetDtcPara API

Description	Sets AIPP DTC parameters of a specific batch.
API Prototype	AIStatus SetDtcPara(uint32_t batchIndex, AippDtcPara dtcPara);
Parameter	**batchIndex**: sequence number of the batch on which the DTC parameters apply. The value must be greater than or equal to **0**. **dtcPara**: DTC parameters
Return Value	**AI_SUCCESS** indicates a success, while other values indicate a failure.

3.5.3 Sample

Assume that the training set of a model consists of BRG888 images. With dynamic AIPP enabled, YUYV images can be received as the input for model inference. If the size of the input is different from that of the training set images, the AIPP cropping, resizing, and padding functions can be used. The following sample is created based on the HiAI APIs, where AIPP cropping is enabled and 480 x 480 YUYV images are preprocessed into 224 x 224 inputs.

```
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("Create 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());

    // Sets dynamic AIPP parameters.
    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);

    // Creates model inputs with AIPP.
    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 User Context

3.6.1 Obtaining the Parameter Content

Table 3-68 GetPara API

Description	Obtains the parameter content by key.
API Prototype	string GetPara(const string &key) const;
Parameter	[Input] **key**: keyword type
Return Value	Keyword content

3.6.2 Adding User-Defined Parameters

Table 3-69 AddPara API

Description	Adds user-defined parameters. The **model_name** parameter is mandatory.
API Prototype	void AddPara(const string &key, const string &value);
Parameter	[Input] **key**: keyword type [Output] **value**: parameter value
Return Value	None

3.6.3 Setting User-Defined Parameters

NOTE

This API is newly added in HiAI DDK V310.

Table 3-70 SetPara API

Description	Sets user-defined parameters.
API Prototype	void SetPara(const string &key, const string &value);
Parameter	[Input] **key**: keyword type [Input] **value**: parameter value
Return Value	None

3.6.4 Deleting User-Defined Parameters

NOTE
This API is newly added in HiAI DDK V310.

Table 3-71 DelPara API

Description	Deletes user-defined parameters.
API Prototype	void DelPara(const string &key);
Parameter	[Input] **key**: keyword type
Return Value	None

3.6.5 Clearing User-Defined Parameters

NOTE
This API is newly added in HiAI DDK V310.

Table 3-72 ClearPara API

Description	Clears user-defined parameters.
API Prototype	void ClearPara();
Parameter	None
Return Value	None

3.6.6 Obtaining All Added Parameter Keys

Table 3-73 GetAllKeys API

Description	Obtains all added parameter keys.
API Prototype	AIStatus GetAllKeys(vector<string> &keys);
Parameter	[Output] **keys**: all keyword types
Return Value	**AIStatus::AI_SUCCESS** indicates a success, while other values indicate a failure.

3.7 Async Callback Registration

3.7.1 Imaginary Function OnProcessDone

Table 3-74 OnProcessDone API

Description	Asynchronously calls back the model execution result. This imaginary function needs to be implemented by the APK and is registered during model initialization.
API Prototype	virtual void OnProcessDone(const AiContext &context, int32_t result, const vector<shared_ptr<AiTensor>> &out_tensor, int32_t iStamp)
Parameter	[Input] **context**: extensible user-defined context [Input] **result**: inference result [Input/Output] **out_tensor**: tensor output after inference [Input] **iStamp**: async handling flag
Return Value	None

3.7.2 Imaginary Function OnServiceDied

Table 3-75 OnServiceDied API

Description	Calls back OnServiceDied. This imaginary function needs to be implemented by the APK. In the C++ implementation, when the server is abnormal and ServiceDied is called back, all model managers are notified to register this async callback API.
API Prototype	virtual void OnServiceDied()
Parameter	None
Return Value	None

3.8 Tensor Dimensions

3.8.1 Setting the Number of Input Tensors

Table 3-76 SetNumber API

Description	Sets the number of input Tensors.
API Prototype	void SetNumber(const uint32_t number);
Parameter	[Input] **number**: number of Tensors in the model input
Return Value	None

3.8.2 Setting the Number of Channels of the Input Tensor

Table 3-77 SetChannel API

Description	Sets the number of channels of the input Tensor.
API Prototype	void SetChannel(const uint32_t channel);
Parameter	[Input] **channel**: number of channels of the input Tensor
Return Value	None

3.8.3 Setting the Height of the Input Tensor

Table 3-78 SetHeight API

Description	Sets the height of the input Tensor.
API Prototype	void SetHeight(const uint32_t height)
Parameter	[Input] **height:** height of the input Tensor
Return Value	None

3.8.4 Setting the Width of the Input Tensor

Table 3-79 SetWidth API

Description	Sets the width of the input Tensor.
API Prototype	void SetWidth(const uint32_t width);
Parameter	[Input] **width**: width of the input Tensor
Return Value	None

3.8.5 Obtaining the Number of Input Tensors

Table 3-80 GetNumber API

Description	Obtains the number of input Tensors.
API Prototype	uint32_t GetNumber() const;
Parameter	None
Return Value	Number of input Tensors

3.8.6 Obtaining the Number of Channels of the Input Tensor

Table 3-81 GetChannel API

Description	Obtains the number of channels of the input Tensor.
API Prototype	uint32_t GetChannel() const;
Parameter	None
Return Value	Number of channels of the input Tensor

3.8.7 Obtaining the Height of the Input Tensor

Table 3-82 GetHeight API

Description	Obtains the height of the input Tensor.
API Prototype	uint32_t GetHeight() const;
Parameter	None
Return Value	Height of the input Tensor

3.8.8 Obtaining the Width of the Input Tensor

Table 3-83 GetWidth API

Description	Obtains the width of the input Tensor.
API Prototype	uint32_t GetWidth() const;
Parameter	None
Return Value	Width of the input Tensor

3.8.9 Checking the Dimension Validity of the Input Tensor

Table 3-84 IsEqual API

Description	Checks the dimension validity of the input tensor.
API Prototype	bool IsEqual(const TensorDimension &dim);
Parameter	Tensor dimensions to be checked.
Return Value	A value of the bool type. **True**: pass **False**: fail

3.9 MemBuffer

3.9.1 Obtaining the MemBuffer Address

Table 3-85 GetMemBufferData API

Description	Obtains the MemBuffer address.
API Prototype	void* GetMemBufferData();
Parameter	None
Return Value	void *

3.9.2 Obtaining the MemBuffer Size

Table 3-86 GetMemBufferSize API

Description	Obtains the MemBuffer size (in bytes).
API Prototype	uint32_t GetMemBufferSize();
Parameter	None
Return Value	Memory capacity

3.10 API Enumerations

3.10.1 AiModelDescription_Frequency

Return Value	Description	Value
AiModelDescription_Frequency_LOW	Low power consumption	1
AiModelDescription_Frequency_MEDIUM	Balanced	2
AiModelDescription_Frequency_HIGH	High performance	3
AiModelDescription_Frequency_EXTREME	Superior performance	4

NOTE

The superior performance will cause high power consumption. Therefore, exercise caution when using this function.

3.10.2 AiModelDescription_DeviceType

Return Value	Description	Value
AiModelDescription_DeviceType_NPU	NPU device	0
AiModelDescription_DeviceType_IPU	IPU device	1
AiModelDescription_DeviceType_MLU	MLU device	2
AiModelDescription_DeviceType_CPU	CPU device	3

3.10.3 AiModelDescription_Framework

Return Value	Description	Value
HIAI_FRAMEWORK_NONE	Non-third-party framework	0
HIAI_FRAMEWORK_TENSORFLOW	TensorFlow	1
HIAI_FRAMEWORK_KALDI	KALDI	2
HIAI_FRAMEWORK_CAFFE	Caffe	3
HIAI_FRAMEWORK_TENSORFLOW_8BIT	8-bit TensorFlow	4
HIAI_FRAMEWORK_CAFFE_8BIT	8-bit Caffe	5

3.10.4 AiModelDescription_ModelType

Return Value	Description	Value
HIAI_MODELTYPE_ONLINE	Online	0
HIAI_MODELTYPE_OFFLINE	Offline	1

3.10.5 AiTensorImage_Format

Return Value	Description	Value
AiTensorImage_YUV420SP_U8	**YUV420SP_U8**	0

Return Value	Description	Value
AiTensorImage_XRGB8888_U8	**XRGB8888_U8**	1
AiTensorImage_YUV400_U8	**YUV400_U8**	2
AiTensorImage_ARGB8888_U8	**AiTensorImage_ARGB8888_U8**	3
AiTensorImage_YUYV_U8	**AiTensorImage_YUYV_U8**	4
AiTensorImage_YUV422SP_U8	**AiTensorImage_YUV422SP_U8**	5
AiTensorImage_AYUV444_U8	**AiTensorImage_AYUV444_U8**	6
AiTensorImage_RGB888_U8	**AiTensorImage_RGB888_U8**	7
AiTensorImage_BGR888_U8	**AiTensorImage_BGR888_U8**	8
AiTensorImage_YUV444SP_U8	**AiTensorImage_YUV444SP_U8**	9
AiTensorImage_YVU444SP_U8	**AiTensorImage_YVU444SP_U8**	10
AiTensorImage_INVALID	Invalid AiTensorImage	255

3.10.6 HIAI_DataType

Return Value	Description	Value
HIAI_DATATYPE_UINT8	**UINT8**	0
HIAI_DATATYPE_FLOAT32	**FLOAT32**	1
HIAI_DATATYPE_FLOAT16	**FLOAT16**	2
HIAI_DATATYPE_INT32	**INT32**	3
HIAI_DATATYPE_INT8	**INT8**	4
HIAI_DATATYPE_INT16	**INT16**	5
HIAI_DATATYPE_BOOL	BOOL	6

Return Value	Description	Value
HIAI_DATATYPE_INT64	**INT64**	7
HIAI_DATATYPE_UINT32	**UINT32**	8
HIAI_DATATYPE_DOUBLE	DOUBLE	9

4 Error Codes

Error Code	Error Type	Error Code Prototype	Triggering Condition
0	Success	AI_SUCCESS	Success
1	Common failure	AI_FAILED	Internal error
2	Model manager uninitialized	AI_NOT_INIT	Model manager uninitialized
3	Argument invalid	AI_INVALID_PARA	Argument invalid
4	Timeout	AI_TIMEOUT	Task execution timed out
7	API unsupported	AI_INVALID_API	API unsupported by the device
8	Null pointer	AI_INVALID_POINTER	Null **this** pointer