Huawei HiAI DDK Integration Manual

Issue: V100.150.10

Date: 2018-03-09

Huawei Technologies Co., Ltd.

Trademarks and Permissions


and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

The method of applying for HiAI is described as follows:

1. **Send an application email to developer@huawei.com.**
2. **The format of the email subject is HUAWEI HiAI+Company name+Product name.**
3. **The format of the email body is Cooperation company+Contact person+Contact information+Contact email address.**
4. **We will send you feedback within five workdays after receiving your email.**

Official website: http://developer.huawei.com/consumer/cn/devunion/ui/server/HiAI.html

Contents

1 DDK Introduction 4
2 DDK Description 4
2.1 App Source 4
2.2 DDK 5
2.3 Documents 5
2.4 Tools 6
3 Integration Overview 6
4 Development Environment 6
5 Model Conversion 6
5.1 Caffe Model Conversion 7
5.2 TensorFlow Model Conversion 7
5.2.1 Offline Model Generation 7
5.2.2 Model Parameter File 7
6 Model Integration 8
6.1 Offline Model Generation 8
6.1.1 Caffe Model Conversion Sample 8
6.2 Interface Integration 9
6.2.1 Creating a Model Manager 9
6.2.2 Loading a Model 11
6.2.3 Running a Model 13
6.2.4 Unloading a Model and Destroying a Model Manager 16
6.3 Error Code Definition 17
7 Q&A 17

Huawei HiAI DDK Integration Manual

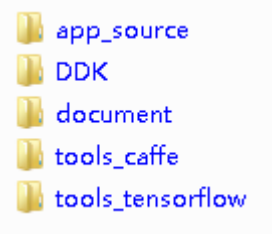
1 DDK Introduction

The HiAI device development kit (DDK) is an artificial intelligence (AI) computing software development kit (SDK) released by HiSilicon. This SDK is oriented to AI application developers and machine learning algorithm personnel. With the HiAI DDK, you can better improve the running speed of the machine learning model on the mobile end.

2 DDK Description

A complete DDK contains five parts, as shown in Figure 2-1.

Figure 2-1 DDK content



- **app_source** contains the source code of the Android demo app.
- **DDK** contains the HiAI open SDK.
- **document** contains the reference documents for development.
- **tools_caffe** and **tools_tensorflow** contain tools used to generate models of the specified format.

2.1 App Source

Use the InceptionV3 classification network as an example. To download the model, visit https://storage.googleapis.com/download.tensorflow.org/models/inception_v3_2016_08_28_frozen.pb.tar.gz. The app source provides a series of synchronous and asynchronous sample code examples covering input pre-processing, model loading, model forward computing, post-processing of the forward computing result, model unloading, and time statistics collection.

Use Android Studio 2.2 or later. (For details, visit the website for Google Android developers at https://developer.android.com/studio/index.html.

Import the app source code and run it. The app supports selection of pictures from the gallery or use of the camera to take pictures. Figure 2-2 shows the running effect of the DDK app.

Figure 2-2 Running effect of the DDK app

2.2 DDK

The DDK consists of two parts:

- **ai_ddk_demo**: uses the picture classification identification demo programs integrated to the DDK interface.

  In **ai_ddk_demo**, **classify_jni.cpp** is a synchronous Java Native Interface (JNI) demo program, and **classify_async_jni.cpp** is an asynchronous JNI demo program.

- ai_ddk_lib: contains the dependent libraries and related header files.

  libai_client.so is the dynamic library that the DDK depends on.

  HiAIModelManager.h is the header file in the DDK and contains declarations of functions in the **libai_client.so** file.

2.3 Documents

The **document** folder contains four documents:

- Huawei HiAI DDK integration manual, that is, this document, which describes the HiAI DDK content and integration method.
- Huawei HiAI DDK integration case, which describes how to use the HiAI DDK using a whole-process case.
- Huawei HiAI DDK user manual, which describes the interfaces and error codes provided in the DDK.
- Operator specification description document, which describes the operators supported by the HiAI DDK V150 and support restrictions.

2.4 Tools

Before using the HiAI to accelerate the Caffe and TensorFlow models, you need to convert the models into the specified format. Conversion tools are provided for Caffe and TensorFlow models, respectively. For details, see chapter 5 "Model Conversion."

3 Integration Overview

Integration of the HiAI DDK consists of the following steps:

Step 1 Assess the operator compatibility.

Currently, the HiAI platform does not support user-defined operator types. For details about the supported operators and operator specifications, see section 2.4 "Supported Operators" in the *Huawei HiAI Operator Specification Document*.

Step 2 Convert the model format.

After completing the assessment of the operator compatibility, you need to convert the Caffe or TensorFlow model into a model format supported by the HiAI platform.

Step 3 Integrate the model.

Model integration consists of five steps: creating a model manager, loading the converted model, computing the model, unloading the model, and destroying the model manager.

----End

4 Development Environment

- The model conversion tool runs on the Linux platform.

  To download Linux images, visit http://mirrors.ustc.edu.cn/.
- NDK r14b or later is recommended for DDK compilation.

  To download the NDK, visit https://developer.android.com/ndk/downloads/index.html.
- Use JDK 8+Android Studio for application development.

  To download Android Studio, visit https://developer.android.com/studio/index.html.

  To download Java JDK 8, visit http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html.

5 Model Conversion

To use the HiAI DDK for model acceleration, you need to convert the Caffe and TensorFlow models to the specified format. This chapter describes how to convert the model format.

5.1 Caffe Model Conversion

The cngen_linux conversion tool is used on the Linux platform. This tool is located in the **tools_caffe** folder of the DDK. To use this tool, run the **./cngen_linux –prototxt xxx.prototxt –model xxx.caffemodel** command. The following is a command example:

```
./cngen_linux --prototxt test.prototxt  --model test.caffemodel
```

output_dir dir: Specifies the path of the generated offline model.

output_name model_name: Specifies the name of the generated offline model.

If the following log is displayed, the model is successfully converted:

```
[covert model.cpp 79]  ForwardPrefilled done!
model compress mode
ipuMaxMemory used: 327936768
```

The tools_caffe folder provides scripts to facilitate model conversion. For details, see the **readme** file.

5.2 TensorFlow Model Conversion

The pb_to_offline conversion tool is used on the Linux platform. This tool can be used to convert the CPU and IPU modules as well as the pb model files generated by TensorFlow 1.3.

5.2.1 Offline Model Generation

A package in the DDK provides the binary conversion tool pb_to_offline (located in the **tools_tensorflow** or **pb_to_offline folder**), which runs on the Linux OS. The tool can convert the CPU or IPU model to the offline model and depend on the **libipu.so** file.

Execute the following code to convert the CPU or IPU model to the offline model:

```
export LD LIBRARY PATH=./so
./pb to offline --graph=model/graph def ipu.pb --param file=model/model.txt
Link the libipu.so file in the so folder.
```

The pb_to_offline tool has two parameters:

graph: Specifies the model converted from the pb model.

param_file: Specifies the model parameter file.

The following two files are generated is the CPU or IPU model is successfully generated to the offline model:

```
offlineModelName.cambricon: offline model
offlineModelName.cambricon.inputs_outputs.aux: intermediate file
```

The pb_to_offline folder provides scripts to facilitate model conversion. For details, see the readme file in the directory.

5.2.2 Model Parameter File

The model parameter file contains the following information:

1. Model Name: Specifies the name of the offline model file to be generated.
2. Node information of the session_run{} model

(1) Number and names of input nodes (input_nodes node of Session::Run) and shape of the input nodes

(2) Number and names of output nodes (output_nodes of Session::Run)

The format of the parameter file and parameters are as follows:

1. model_name:XXX.cambricon

   Specifies the name of the offline model file to be generated.

2. session_run{}

   Indicates the session_run {} of the specified model. Information in curly brackets {} contains data of input and output nodes, that is, data of the start and end nodes Session::Run.

3. input_nodes(n): xxxx, n, h, w, c

   Specifies information of the input node in the specified Session::Run. **n** in the brackets indicates the number of input nodes.

   XXXX indicates the name of the input node.

   n, **h**, **w**, and **c** indicate the sizes of the Tensor shape of each input node from n, h, w, and c dimensions, respectively.

4. output_nodes(n): "xxxx" "yyyy" "zzzz"

   Specifies the name of an output node in the specified Session::Run. **n** in the brackets indicates the number of output nodes.

   xxxx, **yyyy**, and **zzzz** indicate the node names, respectively.

   The following uses InceptionV3 configuration file as an example.

```
model_name: InceptionV3.cambricon
session_run{
  input_nodes(1):                                  //Number of input nodes (1)
  "input",1,299,299,3                                 //Name and shape of the input
node
  output_nodes(1):                                    //Number of output nodes (1)
  "InceptionV3/Predictions/Softmax"                //Name and shape of the output node
}
```

6 Model Integration

Model integration includes offline model generation, offline model loading using model loading interfaces, data pre-processing, model running, post-processing on the obtained data, and model unloading. The integration method is the same regardless of whether a Caffe or TensorFlow model is converted into an offline model.

The interfaces for model manager creation, model loading, model forward computing, and model unloading support both synchronous and asynchronous modes.

6.1 Offline Model Generation

6.1.1 TensorFlow Model Conversion Sample

Use the InceptionV3 network as an example. To download the InceptionV3 model, visit

https://storage.googleapis.com/download.tensorflow.org/models/inception_v3_2016_08_28_frozen.pb.tar.gz

Use the model conversion method described in Chapter 5 "Model Conversion" to convert the TensorFlow model to an offline model.

6.2 Interface Integration

The DDK provides synchronous and asynchronous interfaces. App developers can select synchronous or asynchronous interfaces based on actual requirements. This section describes how to use a single-model interface in synchronous and asynchronous modes. For details about the code, see the DDK demo. For details about the DDK interfaces, see section 2.6 "Supported Interfaces" in the *Huawei HiAI DDK User Manual*.

In the demo, the related files of the synchronous and asynchronous modes are as follows.

Synchronous Mode	**Asynchronous Mode**
Application-layer code file: **SyncClassifyActivity.java**	Application-layer code file: **AsyncClassifyActivity.java**
JNI-layer code file: **classify_jni.cpp**	JNI-layer code file: **classify_async_jni.cpp**

In both modes, you need to obtain the DDK version number to determine whether the system supports NPU acceleration. For details, see section 5.1 "Obtaining the DDK Version Number" in the *Huawei HiAI DDK Integration Case*.

6.2.1 Creating a Model Manager

6.2.1.1 Creating a Model Manager at the Application Layer

- Synchronous mode:

  Invoke the loadModelSync function at the JNI layer to create the synchronous model manager before loading the model.

```
private class loadModelTask extends AsyncTask<Void, Void, Integer> {
        @Override
        protected Integer doInBackground(Void... voids) {
            int ret = ModelManager.loadModelSync("InceptionV3", mgr);
            return ret;
        }
```

- Asynchronous mode:

  Invoke the registerListenerJNI function at the JNI layer to create the asynchronous model manager.

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getSupportActionBar().hide();
        setContentView(R.layout.activity async classify);
        mgr = getResources().getAssets();
        int ret = ModelManager.registerListenerJNI(listener);
     ……
    }
```

6.2.1.2 Creating a Model Manager at the JNI Layer

Invoke the DDK interface HiAI_ModelManager_create in the JNI function to create the model manager at the JNI layer.

- Synchronous mode:

```
extern "C"  JNIEXPORT jint JNICALL
Java com huawei hiaidemo ModelManager loadModelSync(JNIEnv *env, jobject instance,
jstring jmodelName, jobject assetManager){
    ……
modelManager = HIA ModelManager create(NULL);
……
```

- Asynchronous mode:

```
extern "C"
JNIEXPORT jint JNICALL
Java com huawei hiaidemo ModelManager registerListenerJNI(JNIEnv *env, jobject obj,
                                                          jobject callbacks) {
    callbacksInstance = env->NewGlobalRef(callbacks);
    jclass objClass = env->GetObjectClass(callbacks);
    if (objClass) {
        callbacksClass = reinterpret cast<jclass>(env->NewGlobalRef(objClass));
        env->DeleteLocalRef(objClass);
    }
    listener.onLoadDone = onLoadDone;
    listener.onRunDone = onRunDone;
    listener.onUnloadDone = onUnloadDone;
    listener.onTimeout = onTimeout;
    listener.onError = onError;
    listener.onServiceDied = onServiceDied;
    modelManager = HiAI ModelManager create(&listener);
    return 0;
}
```

NewGlobalRef: used to obtain the ModelManagerListener object instance reference input at the application layer

GetObjectClass: used to obtain the input ModelManagerListener object type

DeleteLocalRef: used to release the reference object at the application layer

The following describes how to call functions for the HIAI_ModelManagerListener object in asynchronous mode:

Use onLoadDone as an example.

```
void onLoadDone(void *userdata, int taskId) {
    LOGE("AYSNC JNI layer onLoadDone:", taskId);
    JNIEnv *env;
    jvm->AttachCurrentThread(&env, NULL);
    if (callbacksInstance != NULL) {
        jmethodID onValueReceived = env->GetMethodID(callbacksClass, "onStartDone",
"(I)V");
        env->CallVoidMethod(callbacksInstance, onValueReceived, taskId);
    }
}
```

The following code is used to obtain the JNIEnv pointer of the current thread from the local code:

```
JNIEnv *env;
jvm->AttachCurrentThread(&env, NULL);
```

GetMethodID: used to obtain the function interfaces in the ModelManagerListener class input at the application layer.

CallVoidMethod: used to call the onStartDone function.

6.2.1.3 Creating a Model Manager at the DDK Layer

The following is the function prototype for creating a model manager at the DDK layer:

```
HiAI_ModelManager* HiAI_ModelManager_create(HiAI_ModelManagerListener* listener);
```

A synchronous or an asynchronous model manager is created using input parameters. When the parameter is set to null, the synchronous model manager is created. When the input HIAI_ModelManagerListener is a non-empty instance pointer, the asynchronous model manager is created.

6.2.2 Loading a Model

You need to load a model before using it. The DDK supports single-model load and multi-model load. It also supports load of models from an SD card and the app source code directory **assets**.

6.2.2.1 Loading a Model at the Application Layer

- Synchronous mode:

  Invoke the loadModelSync function at the JNI layer to load the model in synchronous mode.

```
private class loadModelTask extends AsyncTask<Void, Void, Integer> {
        @Override
        protected Integer doInBackground(Void... voids) {
            int ret = ModelManager.loadModelSync("InceptionV3", mgr);
            return ret;
        }
```

- Asynchronous mode:

  Invoke the loadModelAsync function at the JNI layer to load the model in asynchronous mode.

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getSupportActionBar().hide();
        setContentView(R.layout.activity async classify);
        mgr = getResources().getAssets();
        int ret = ModelManager.registerListenerJNI(listener);
        Log.e(TAG, "onCreate: " + ret);
        ModelManager.loadModelAsync("InceptionV3", mgr);
        items = new ArrayList<>();
        mgr = getResources().getAssets();
        initView();
    }
```

6.2.2.2 Loading a Model at the JNI Layer

- Synchronous mode:

```
extern "C"
JNIEXPORT jint JNICALL
Java com huawei hiaidemo ModelManager loadModelSync(JNIEnv *env, jobject instance,
jstring jmodelName, jobject assetManager){
  ……
AAssetManager *mgr = AAssetManager fromJava(env, assetManager);
    LOGI("Attempting to load model...\n");
    LOGE("model name is %s", modelname);
    AAsset *asset = AAssetManager open(mgr, modelname, AASSET MODE BUFFER);
    if (nullptr == asset) {
```

```
        LOGE("AAsset is null...\n");
    }
    const void *data = AAsset_getBuffer(asset);
    if (nullptr == data) {
        LOGE("model buffer is null...\n");
    }
    off_t len = AAsset_getLength(asset);
    if (0 == len) {
        LOGE("model buffer length is 0...\n");
    }

    HiAI_ModelBuffer *modelBuffer = HiAI_ModelBuffer_create_from_buffer(modelName,
               (void *) data, len,   HiAI_DevPerf::HiAI_DEVPREF_HIGH);
    HiAI_ModelBuffer *modelBufferArray[] = {modelBuffer};

    int ret = HiAI_ModelManager_loadFromModelBuffers(modelManager, modelBufferArray, 1);
    LOGI("load model from assets ret = %d", ret);
    env->ReleaseStringUTFChars(jmodelName, modelName);
    AAsset_close(asset);
    return ret;
}
```

At the JNI layer, the AssetManager is obtained by using the following code:

```
AAssetManager *mgr = AAssetManager_fromJava(env, assetManager);
```

For details about the AssetManager APIs, visit the website for Android developers at https://developer.android.com/ndk/reference/asset__manager_8h.html.

```
AAsset* asset = AAssetManager_open(mgr, "hiai.cambricon", AASSET_MODE_BUFFER);
```

The AAssetManager_open interface is used to read the **hiai.cambricon** file in the **assets** directory of the app source code and returns **AAsset**.

Obtain the buffer address and size using the following functions:

```
void *data = (void *)AAsset_getBuffer(asset);
off_t len = AAsset_getLength(asset);
```

Invoke the DDK interface function HIAI_ModelBuffer_create_from_buffer to create the HIAI_ModelBuffer object and invoke HIAI_ModelManager_loadFromModelBuffers to load the model.

- Asynchronous mode:

```
extern "C"
JNIEXPORT void JNICALL
Java com huawei hiaidemo ModelManager loadModelAsync(JNIEnv *env, jobject instance,
             jstring jmodelName, jobject assetManager) {
……
    HiAI_ModelBuffer *modelBuffer = HiAI_ModelBuffer_create_from_buffer(modelName,
           (void *) data, len, HiAI_DevPerf::HiAI_DEVPREF_HIGH);
    HiAI_ModelBuffer *modelBufferArray[] = {modelBuffer};
    int ret = HiAI_ModelManager_loadFromModelBuffers(modelManager, modelBufferArray, 1);
    LOGE("ASYNC JNI LAYER load model from assets ret = %d", ret);
    env->ReleaseStringUTFChars(jmodelName, modelName);
    AAsset_close(asset);
}
```

The loading process is the same as that in synchronous mode. The only difference is that the input parameter is the asynchronous model management engine when the HiAI_ModelManager_loadFromModelBuffers interface function is invoked.

6.2.2.3 Loading a Model at the DDK Layer

The following is the interface function prototype used to load a model at the DDK layer:

```
int HiAI ModelManager loadFromModelBuffers(HiAI ModelManager* manager, HiAI ModelBuffer*
bufferArray[], int nBuffers);
```

manager: Specifies the object interface of the model management engine (synchronous or asynchronous).

bufferArray[]: HIAI_ModelBuffer. Single- and multi-model are both supported.

nBuffers: Specifies the number of models to be loaded.

6.2.3 Running a Model

6.2.3.1 Running a Model at the Application Layer

- Synchronous mode:

  Invoke the runModelSync function at the JNI layer to run the model in synchronous mode.

```
private class RunModelTask extends AsyncTask<Bitmap, Void, String[]> {
        @Override
        protected String[] doInBackground(Bitmap... bitmaps) {
            float[] buffer = getPixel(bitmaps[0], RESIZED WIDTH, RESIZED HEIGHT);
            initClassifiedImg = bitmaps[0];
            predictedClass = ModelManager.runModelSync("hiai", buffer);
            return predictedClass;
        }
   ……
    }
```

  Use the getPixel function to obtain the model input from images, and then invoke the runModelSync function at the JNI layer to run the model in synchronous mode.

- Asynchronous mode:

  Invoke the runModelAsync function at the JNI layer to run the model in asynchronous mode.

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (resultCode == RESULT OK && data != null) switch (requestCode) {
            case GALLERY REQUEST CODE:
                try {
                    Bitmap bitmap;
                    ContentResolver resolver = getContentResolver();
                    Uri originalUri = data.getData();
                    bitmap = MediaStore.Images.Media.getBitmap(resolver, originalUri);
                    String[] proj = {MediaStore.Images.Media.DATA};
                    Cursor cursor = managedQuery(originalUri, proj, null, null, null);
                    cursor.moveToFirst();
                    Bitmap rgba = bitmap.copy(Bitmap.Config.ARGB_8888, true);
                    final Bitmap initClassifiedImg = Bitmap.createScaledBitmap(rgba,
RESIZED_WIDTH, RESIZED_HEIGHT, false);
                    final float[] pixels = getPixel(initClassifiedImg, RESIZED_WIDTH,
RESIZED_HEIGHT);
                    show = initClassifiedImg;
                    ModelManager.runModelAsync("hiai", pixels);
                } catch (IOException e) {
                    Log.e(TAG, e.toString());
```

```
                }
                break;
……
    }
```

6.2.3.2 Running a Model at the JNI Layer

- Synchronous mode:

```
extern "C"
JNIEXPORT jobjectArray JNICALL
Java com huawei hiaidemo ModelManager runModelSync(JNIEnv *env, jclass type, jstring
jmodelName, jfloatArray jbuf) {
    const char *modelName = env->GetStringUTFChars(jmodelName, 0);
    if (NULL == modelManager) {
        LOGE("please load model first");
        return NULL;
    }
    float *dataBuff = NULL;
    if (NULL != jbuf) {
        dataBuff = env->GetFloatArrayElements(jbuf, NULL);
    }
    inputtensor = HiAI TensorBuffer create(input_N, input_C, input_H, input_W);
    HiAI TensorBuffer *inputtensorbuffer[] = {inputtensor};
    outputtensor = HiAI TensorBuffer create(1, 1000, 1, 1);
    HiAI TensorBuffer *outputtensorbuffer[] = {outputtensor};
    float *inputbuffer = (float *) HiAI TensorBuffer getRawBuffer(inputtensor);
    int length = HiAI TensorBuffer getBufferSize(inputtensor);
    LOGE("SYNC JNI runModel modelname:%s", modelName);
    memcpy(inputbuffer, dataBuff, length);
    float time use;
    struct timeval tpstart, tpend;
    gettimeofday(&tpstart, NULL);
    int ret = HiAI ModelManager runModel(
            modelManager,
            inputtensorbuffer,
            1,
            outputtensorbuffer,
            1,
            1000,
            modelName);
    LOGE("run model ret: %d", ret);
    gettimeofday(&tpend, NULL);
……
```

Use the env->GetFloatArrayElements(jbuf, NULL) function to obtain the application-layer input data.

Invoke the HiAI_TensorBuffer_create (input_N, input_C, input_H, input_W) function to create the inputTensor.

Then, invoke the DDK interface function

HiAI_TensorBuffer_getRawBuffer(inputtensor) to obtain the inputTensor address.

Invoke the HiAI_TensorBuffer_getBufferSize(inputtensor) function to obtain the inputTensor size.

Finally, use the memcpy function to copy the input dataBuff at the application layer to the inputTensor.

After the copy is complete, create the outputTensor.

After the inputTensor and outputTensor are ready, invoke the HiAI_ModelManager_runModel interface to run the model.

After the model is run, perform post-processing on the data generated after model running.

- Asynchronous mode:

```
extern "C"
JNIEXPORT void JNICALL
Java com huawei hiaidemo ModelManager runModelAsync(JNIEnv *env, jobject instance,
                                    jstring jmodelName, jfloatArray jbuf) {
    ……
    int ret = HiAI ModelManager runModel(
            modelManager,
            inputtensorbuffer,
            1,
            outputtensorbuffer,
            1,
            1000,
            modelName);
    LOGE("ASYNC JNI layer runmodel ret: %d", ret);
    ……
}
```

The model running process in asynchronous mode is the same as that in synchronous mode. The only difference is that the input parameter is the asynchronous model management engine when the HiAI_ModelManager_runModel interface function is invoked.

The post-processing after the model is run in asynchronous mode is implemented through the callback function onRunDone.

6.2.3.3 Running a Model at the DDK Layer

The following is the interface function prototype for running a model at the DDK layer:

```
int HiAI_ModelManager_runModel(
        HiAI_ModelManager* manager,
        HiAI_TensorBuffer* input[],
        int nInput,
        HiAI_TensorBuffer* output[],
        int nOutput,
        int ulTimeout,
        const char* modelName);
```

manager: Specifies the object interface of the model management engine.

input[]: Specifies the model input. Multiple inputs are supported.

nInput: Specifies the number of inputs by a model.

output[]: Specifies the model output. Multiple outputs are supported.

nOutput: Specifies the number of outputs by a model.

ulTimeout: Specifies the timeout.

modelName: Specifies a model name.

6.2.4 Unloading a Model and Destroying a Model Manager

6.2.4.1 Unloading the Model and Destroying the Model Manager at the Application Layer

- Synchronous mode:

  Invoke the unloadModelSync function at the JNI layer to unload the model in synchronous mode.

```
protected void onDestroy() {
        super.onDestroy();
        int result = ModelManager.unloadModelSync();

        if (AI_OK == result) {
            Toast.makeText(this, "unload model success.", Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(this, "unload model fail.", Toast.LENGTH_SHORT).show();
        }
    }
```

- Asynchronous mode:

  Invoke the unloadModelAsync function at the JNI layer to unload the model in asynchronous mode.

```
protected void onDestroy() {
        super.onDestroy();
        ModelManager.unloadModelAsync();
    }
```

  The model manager in asynchronous mode is unloaded through the callback function.

6.2.4.2 Unloading the Model and Destroying the Model Manager at the JNI Layer

- Synchronous mode:

```
extern "C"
JNIEXPORT jint JNICALL
Java com huawei hiaidemo ModelManager unloadModelSync(JNIEnv *env, jobject instance) {
    if (NULL == modelManager) {
        LOGE("please load model first.");
        return -1;
    } else {
        if (modelBuffer != NULL) {
            HiAI ModelBuffer destroy(modelBuffer);
            modelBuffer = NULL;
        }
        int ret = HiAI ModelManager unloadModel(modelManager);

        LOGE("JNI unload model ret:%d", ret);
        HiAI ModelManager destroy(modelManager);
        modelManager = NULL;
        return ret;
    }
 }
```

- Asynchronous mode:

```
void onUnloadDone(void *userdata, int taskStamp) {
    LOGE("JNI layer onUnloadDone:", taskStamp);
    JNIEnv *env;
    jvm->AttachCurrentThread(&env, NULL);
```

```
    if (callbacksInstance != NULL) {
        jmethodID onValueReceived = env->GetMethodID(callbacksClass, "onStopDone",
"(I)V");
        env->CallVoidMethod(callbacksInstance, onValueReceived, taskStamp);
    }
    HiAI_ModelManager_destroy(modelManager);
    modelManager = NULL;
    listener.onRunDone = NULL;
    listener.onUnloadDone = NULL;
    listener.onTimeout = NULL;
    listener.onServiceDied = NULL;
    listener.onError = NULL;
    listener.onLoadDone = NULL;
}

extern "C"
JNIEXPORT void JNICALL
Java_com_huawei_hiaidemo_ModelManager_unloadModelAsync(JNIEnv *env, jobject instance)
{
    if (NULL == modelManager) {
        LOGE("please load model first");
        return;
    } else {
        if (modelBuffer != NULL) {
            HiAI ModelBuffer destroy(modelBuffer);
            modelBuffer = NULL;
        }
        int ret = HiAI ModelManager unloadModel(modelManager);
        LOGE("ASYNC JNI layer unLoadModel ret:%d", ret);
    }
}
```

The model is unloaded using the unloadModelAsync function, while the model manager is destroyed using the callback function onUnloadDone.

6.2.4.3 Unloading the Model and Destroying the Model Manager at the DDK Layer

The following is the interface function prototype for unloading a model at the DDK layer:

```
int HiAI_ModelManager_unloadModel(HiAI_ModelManager* manager);
```

The following is the interface function prototype for destroying the model manager at the DDK layer:

```
void HiAI_ModelManager_destroy(HiAI_ModelManager* manager);
```

manager: Specifies the object interface of the model management engine.

6.3 Error Code Definition

When the interfaces of model loading, running, and unloading fail to be invoked, an error code will be returned. This error code is defined in the appendix of the *Huawei HiAI DDK User Manual*.

7 Q&A

1. What do I do if the model does not support some operators?

For example, if the NPU does not support the Softmax at the last layer of AlexNet, developers need to perform related processing on the CPU.

2. How do I test the forward computing time?

   Add time printing before and after the runModel interface is invoked at the JNI layer.

3. What do I do if the Huawei DDK uses c++_shared for compilation but my program uses gnustl_static?

   Perform 1-layer C interface encapsulation as required based on the C++ interfaces provided by Huawei. Then you can use gnustl_static for compilation and running.

4. My app uses CMake for compilation. Does the Huawei DDK support CMake compilation?

   Yes. You need to perform 1-layer C interface encapsulation as required based on the C++ interfaces provided by Huawei. Then you can use CMake for compilation.

5. What do I do if I have completed the integration but the running effect is not as expected or the result is incorrect?

   Locate the fault by checking the model compatibility first. For details, see section 2.4 "Supported Operators" in the *Huawei HiAI DDK User Manual*. Check whether some operator specifications are not supported.

6. Can models be loaded from an SD card or the **assets** directory?

   The models can be loaded from the SD card or the **assets** directory. For details, see the demo program.

7. Should I select the synchronous or asynchronous interface?

   Developers can select the interface based on their own requirements.

8. What is the model segment involved in the generation of offline models? How do I determine the model segment? Does the DDK support the model segment?

In section 2.4 "Supported Operators" of *Huawei HiAI DDK User Manual*, the value of **deconvolution layer max(stride_h,stride_w)*no** cannot be greater than 256. If a middle layer of the model is the deconvolution layer and the value of **max(stride_h,stride_w)*no** of the deconvolution layer is greater than 256, the offline model is segmented when the offline model generation tool cngen_linux is used. The detailed log information is as follows:

```
----------------net.cpp | buff 1---------------
[net.cpp 1391] ipuOpenDeviceStream() stream 0
[net.cpp 1405] ipuCloseDeviceStream() stream 0
[NGPF:release] DDR Access Times: 76
[net.cpp 1539] ipuStreamExecuteOnce()
[net.cpp 1391] ipuOpenDeviceStream() stream 1
[net.cpp 1405] ipuCloseDeviceStream() stream 1
[NGPF:release] DDR Access Times: 7112
[net.cpp 1539] ipuStreamExecuteOnce()
ipuSwitchPipelineBuffer, org Index: 1
ipuSwitchPipelineBuffer, new Index: 0
[covert model.cpp 129] ForwardPrefilled done!
model compress mode
ipuMaxMemory used: 386669312
```

The number of segments in the offline model can be determined based on the number of ipuOpenDeviceStream. In the preceding example, the offline model is divided into two segments (there is only one final file). The DDK supports running only of the first-segment offline model.