Huawei HiAI DDK Integration Case

Issue: V100.150.10
Date: 2018-03-09
Huawei Technologies Co., Ltd.
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.

Contents

1 Creating a Project ................................................................................................................. 4
2 Operator Compatibility Assessment ..................................................................................... 5
3 Model Format Conversion ..................................................................................................... 7
4 Interface Introduction ............................................................................................................ 7
5 Interface Integration ............................................................................................................... 9
  5.2 Obtaining the DDK Version Number .................................................................................. 10
      5.2.1 Invoking the Model Manager to Obtain the DDK Version Number at the Application Layer .............. 10
      5.2.2 Obtaining the DDK Version Number at the JNI Layer ......................................................... 11
      5.2.3 Obtaining the DDK Version Number at the DDK Layer ....................................................... 11
  5.3 Creating a Model Manager .................................................................................................. 11
      5.3.1 Creating a Model Manager at the Application Layer ............................................................... 11
      5.3.2 Creating a Model Manager at the JNI Layer ......................................................................... 11
      5.3.3 Creating a Model Manager at the DDK Layer ..................................................................... 13
  5.4 Loading a Model ................................................................................................................ 13
      5.4.1 Loading a Model at the Application Layer ........................................................................... 13
      5.4.2 Loading a Model at the JNI Layer ..................................................................................... 13
      5.4.3 Loading a Model at the DDK Layer ................................................................................... 15
  5.5 Running a Model ............................................................................................................... 15
      5.5.1 Running a Model at the Application Layer ........................................................................... 15
      5.5.2 Running a Model at the JNI Layer ..................................................................................... 16
      5.5.3 Running a Model at the DDK Layer ................................................................................... 17
  5.6 Unloading a Model and Destroying a Model Manager ..................................................... 18
      5.6.1 Unloading a Model and Destroying the Model Manager at the Application Layer .................. 18
      5.6.2 Unloading a Model and Destroying the Model Manager at the JNI Layer ............................. 18
      5.6.3 Unloading a Model and Destroying the Model Manager at the DDK Layer .......................... 19
Huawei HiAI DDK Integration Case

This document uses the TensorFlow InceptionV3 network as an example to describe the overall process of device development kit (DDK) access. The development environment is JDK 8 for 64-bit Windows 7+Android Studio 3.0.1.

A complete DDK integration case requires the following steps:

Step 1 Assess the operator compatibility.
Step 2 Convert the model format.
Step 3 Load the model.
Step 4 Run the model.
Step 5 Unload the model.

----End

1 Creating a Project

Create an Android Studio project and select Include C++ support.

Customize C++ Support:
Choose **C++11** from the **C++ Standard** drop-down list box. Select **Exceptions Support** (-fexceptions) and **Runtime Type Information Support** (-frtti).

### 2 Operator Compatibility Assessment
According to the *Huawei HiAI DDK Operator Specification Document*, the last operator softmax is not supported. Therefore, users need to implement the last layer by themselves. Other operators and operator parameters meet the operator specification requirements.

### 3 Model Format Conversion

To download the InceptionV3 model, visit

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

Compile the InceptionV3 configuration file.

```plaintext
model_name: InceptionV3.cambricon
session_run{
  input_nodes(1):
    "input",1,299,299,3
  output_nodes(1):
    "InceptionV3/Predictions/Softmax"
}
```

Use the `run.sh` conversion tool in `tools_tensorflow` of the DDK to convert the model. (The original model, configuration file, and run script are placed in directories of the same level.)

```plaintext
~/tools_tensorflow$ ./run.sh
"Please input the tensorflow model file such as ***.pb :" inceptionv3.pb
"Please input the param file such as ***.txt :" inceptionv3.txt
```

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

```
I tensorflow/stream_executor/ipu/ipu_model_generator.cc:248] write binary proto done!
I tensorflow/stream_executor/ipu/ipu_executor_common.cc:72] cngen::ipuLibExit
[Info]:ipuMaxMemory used: 83200256        [ipu_lib.cpp:130  96722]
```

### 4 Interface Introduction

The DDK provides a model manager that is based on C language interfaces. The function declaration is stored in the `HIAIModelManager.h` file.

```c
int HIAI_ModelManager_loadFromModelBuffers(HIAI_ModelManager* manager, HIAI_ModelBuffer* bufferArray[], int nBuffers);
int HIAI_ModelManager_runModel{
    HIAI_ModelManager* manager,
    HIAI_TensorBuffer* input[],
    int nInput,
    HIAI_TensorBuffer* output[],
    int nOutput,
    int ulTimeout,
    const char* modelName);
HIAI_ModelManager_unloadModel(HIAI_ModelManager* manager);
char* HIAI_GetVersion();
```

For details about the DDK interfaces, see section 2.6 "Supported Interfaces" in the *Huawei HiAI DDK User Manual*. 
Copy the `classify_jni.cpp`, `classify_async_jni.cpp`, `common.cpp`, `common.h`, and `HIAIModelManager.h` files to the `jni` and `include` directories of the DDK, respectively, and modify the `Android.mk` file.

```makefile
LOCAL_PATH := $(call my-dir)

HIAI_SDK_HOME := $(LOCAL_PATH)/../../ai_ddk_lib

ifeq ($(TARGET_ARCH_ABI),arm64-v8a)
    HIAI_SDK_LIB_PATH := $(HIAI_SDK_HOME)/lib64
else
    HIAI_SDK_LIB_PATH := $(HIAI_SDK_HOME)/lib
endif

include $(CLEAR_VARS)
LOCAL_MODULE := hiai
LOCAL_CFLAGS += -D__ndk1=__1
LOCAL_SRC_FILES :=
    classify_async_jni.cpp
    classify_jni.cpp
    common.cpp

LOCAL_C_INCLUDES :=
    $(HIAI_SDK_HOME)/include
    $(LOCAL_PATH)/../../ai_ddk_demo/jni

LOCAL_LDLIBS +=
    -lai_client
    -llog
    -landroid
    -ljnigraphics

LOCAL_LDFLAGS :=
    -Wl,-rpath-link=$(HIAI_SDK_LIB_PATH)
    -L $(HIAI_SDK_LIB_PATH)

ifeq ($(VERBOSE_BUILD),true)
    LOCAL_LDFLAGS += -v
endif
CPPFLAGS=-stdlib=libstdc++ LDLIBS=-lstdc++
LOCAL_CFLAGS += -std=c++14
include $(BUILD_SHARED_LIBRARY)
#include $(BUILD_EXECUTABLE)
```

In the `jni` directory, run the `ndk-build` command to compile the file. `libhiai.so` is the compiled .so file. In this case, `arm64-v8a` is used as an example.

```
[arm64-v8a] SharedLibrary : libhiai.so
[arm64-v8a] Install : libhiai.so => libs/arm64-v8a/libhiai.so
```

Copy the `libhiai.so` file to the `/src/main/libs/arm64-v8a` directory in Android Studio.
Copy the generated offline model and tag file (labels.txt in the Android source code directory src/main/assets/ provided by the DDK) to the /src/main/assets directory of the created project.

5 Interface Integration

The model usage process includes six steps: obtaining the DDK version number, creating the model manager, loading the model, calculating the model, unloading the model, and destroying the model manager. The DDK provides synchronous and asynchronous interfaces. App developers can select synchronous or asynchronous interfaces based on actual requirements. This section describes the use of a single model in synchronous and asynchronous modes and the implementation and invocation of each step in the process at the DDK, JNI, and application layers. 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.

<table>
<thead>
<tr>
<th>Synchronous Mode</th>
<th>Asynchronous Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application-layer code file:</td>
<td>Application-layer code file:</td>
</tr>
<tr>
<td>SyncClassifyActivity.java</td>
<td>AsyncClassifyActivity.java</td>
</tr>
<tr>
<td>JNI-layer code file: classify_jni.cpp</td>
<td>JNI-layer code file: classify_asynchronous_jni.cpp</td>
</tr>
</tbody>
</table>

The demo APK supports selection of pictures from the gallery or use of the camera to take pictures. Figure 5-1 shows the running effect of the demo APK.
Figure 5-1 Running effect of the demo app

king penguin - 80.0461%

drake - 0.325219%
bee eater - 0.171378%

inference time: 77.628 ms

koala - 95.5941%

wombat - 0.267457%
hyena - 0.0298909%

inference time: 88.81 ms

5.2 Obtaining the DDK Version Number

Before using HiAI DDK for acceleration, you need to obtain the DDK version number and determine whether the system supports NPU acceleration based on the returned value. In the case, the CPU is used to run the model when the NPU is unavailable.

5.2.1 Invoking the Model Manager to Obtain the DDK Version Number at the Application Layer

String VersionName = ModelManager.getHiAiVersion();

If the returned value is 000.000.000.000, the version does not support NPU acceleration. For details, see section 2.5.1 "Obtaining the DDK Version Number" in the Huawei HiAI DDK User Manual.
5.2.2 Obtaining the DDK Version Number at the JNI Layer

```java
Java_com_huawei_hiaidemo_ModelManager_getHiAiVersion(JNIEnv *env, jobject instance) {
    char* versionName;
    jstring rtstr = env->NewStringUTF(versionName);
    try{
        versionName = HIAI_GetVersion();
        rtstr = env->NewStringUTF(versionName);
    }
    catch(...){
        rtstr = env->NewStringUTF("000.000.000.000");
    }
    return rtstr;
}
```

At this layer, the HIAI_GetVersion function is invoked to obtain the DDK version number and convert the returned value to java characters for output.

5.2.3 Obtaining the DDK Version Number at the DDK Layer

Execute the following code to obtain the DDK version number:

```java
char* HIAI_GetVersion();
```

The type of the returned value is the string pointer.

5.3 Creating a Model Manager

5.3.1 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 the model is loaded.

```java
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 model manager in asynchronous mode.

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

5.3.2 Creating a Model Manager at the JNI Layer

Invoke the DDK interface HIAL_ModelManager_create at the JNI layer to create a model manager.
Synchronous mode:

```
extern "C" JNIEXPORT jint JNICALL Java_com_huawei_hiaidemo_ModelManager_loadModelSync(JNIEnv *env, jobject instance, jstring modelName, jobject assetManager)
{
    ...
    modelManager = HIAI_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 object referenced by 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("ASYNC 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.
5.3.3 Creating a Model Manager at the DDK Layer

The following is the function prototype used to create a model manager at the DDK layer.

```c
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.

5.4 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.

5.4.1 Loading a Model at the Application Layer

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

```java
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.

```java
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    getSupportActionBar().hide();
    setContentView(R.layout.activity_asynchronous_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();
}
```

5.4.2 Loading a Model at the JNI Layer

- Synchronous mode:

```c
extern "C"
JNIEXPORT jint JNICALL
Java_com_huawei_haiidemo_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.

Execute the following code:

AAsset* asset = AAssetManager_open(mgr, "InceptionV3.cambricon", AASSET_MODE_BUFFER);
The AAssetManager_open interface is used to read the InceptionV3.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"
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.

### 5.4.3 Loading a Model at the DDK Layer

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

```c
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**: number of loaded models

### 5.5 Running a Model

#### 5.5.1 Running a Model at the Application Layer

- **Synchronous mode:**
  Invoke the runModelSync function at the JNI layer to run the model in synchronous mode.

```java
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("InceptionV3", 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.

```java
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);...
            } catch (Exception e) {
                // Handle exception
            }
    }
    ...
}
```
final float[] pixels = getPixel(initClassifiedImg, RESIZED_WIDTH, RESIZED_HEIGHT);
    show = initClassifiedImg;
    ModelManager.runModelAsync("InceptionV3", pixels);
} catch (IOException e) {
    Log.e(TAG, e.toString());
}
break;
……

5.5.2 Running a Model at the JNI Layer

- Synchronous mode:

```c
extern "C"
JNIEXPORT jobjectArray JNICALL 
Java_com_huawei_hiaidemo_ModelManager_runModelSync(JNIEnv *, jclass, 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(output_N, output_C, output_H, output_W);
    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:

  ```c
  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.

### 5.5.3 Running a Model at the DDK Layer

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

```c
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.

### 5.6 Unloading a Model and Destroying a Model Manager

#### 5.6.1 Unloading a 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.

```java
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.

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

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

#### 5.6.2 Unloading a Model and Destroying the Model Manager at the JNI Layer

- **Synchronous mode:**

```c
extern "C"
JNIFUNcci NTJN ccall 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_MModelBuffer_destroy(modelBuffer);
            modelBuffer = NULL;
        }
        int ret = HIAI_ModelManager_unloadModel(modelManager);
        LOGE("JNI unload model ret:%d", ret);
    }
    return ret;
}
```
Asynchronous mode:

```c
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;
}
```

```c
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.

5.6.3 Unloading a 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:

```c
int HIAI_ModelManager_unloadModel(HIAI_ModelManager* manager);
```

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

```c
void HIAI_ModelManager_destroy(HIAI_ModelManager* manager);
```

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