Huawei HiAI DDK User Manual

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.**

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

Contents

Huawei HiAI DDK User Manual

1 Overview

HiAI is an artificial intelligence (AI) computing platform oriented to mobile terminals. The computing library of the mobile computing platform, which is formed by the HiAI application programming interfaces (APIs), is designed for developers to conveniently and efficiently write AI applications running on mobile devices.

Released in a unified binary file, the HiAI APIs are used to accelerate the computing of neural networks using the HiAI heterogeneous computing platform. Currently, these APIs can run only on the Kirin system on chip (SoC) and are integrated to the Android system that uses the Kirin SoC, allowing developers to run the neural network model in the integrated environment and invoke the HiAI APIs to accelerate computation. The HiAI device development kit (DDK) is the HiAI resource package available to third-party developers.

2 Integration Description

2.1 Function Introduction

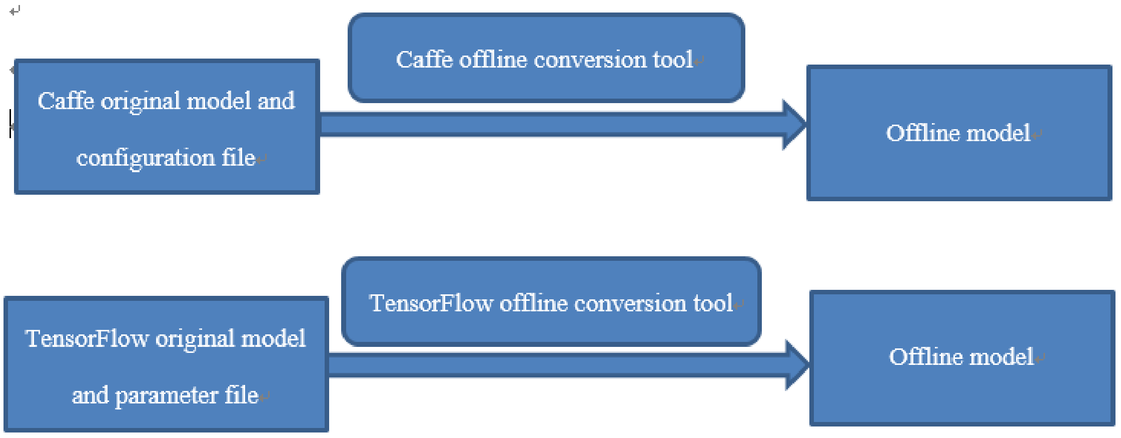
The HiAI DDK provides the AI model manager for AI application developers. It is a model management interface that provide functions such as model loading, model computation, and model unloading.

2.2 Execution Mode

The HiAI DDK computing library runs in offline mode. It uses the compiled and optimized offline model to perform neural network computation. This reduces memory usage while improving computing performance. You can convert a Caffe or TensorFlow model to an offline model using a dedicated conversion tool.

2.2.1 Offline Model Generation

The HiAI DDK provides a model conversion tool to convert the Caffe or TensorFlow models to an NPU model. For details about how to use the conversion tool, see Chapter 5"Model Conversion" in the *Huawei HiAI DDK Integration Manual*.


Caffe offline conversion tool
Caffe original model and configuration file
Offline model
TensorFlow offline conversion tool
TensorFlow original model and parameter file
Offline model

2.2.2 Offline Model Computation

When the user APK uses the DDK, the interfaces for loading, running, and unloading the offline model needs to be invoked to calculate the user-input data on the HiAI heterogeneous acceleration system. For details, see section 6.2 "Interface Integration" in the *Huawei HiAI DDK Integration Manual*.

2.3 Supported Operators

For details, see the Huawei HiAI DDK Operator Specification Document.

2.4 Restrictions

The ROM and RAM space on a mobile phone chip is limited. Therefore, the sizes of the model and the running memory must be restricted as follows:

- The model size is less than or equal to 100 MB.
- The size of the peak memory is less than or equal to 200 MB.

2.5 Supported Interfaces

2.5.1 Obtaining the DDK Version Number

The HIAI_GetVersion interface is used to obtain the DDK version number. The function maintains a static version number and invokes the Android system interface __system_property_get to obtain the value of the system attribute ro.config.HiAIversion, that is, the DDK version number.

Table 2-1 Description of the HiAI_GetVersion interface

Function Description	Obtaining the DDK version number in the system.
Interface Prototype	char* HiAI_GetVersion()
Parameter Description	None
Return Value	If the execution succeeds, the corresponding DDK version number is returned. The version number is described in the format of <major>.<middle>.<minor>.<point>. **<major>**: Indicates a product form. ● XX: mobile phone, ● XX: edge computing. ● XX: Cloud **<middle>**: Indicates the V version of a product form, expressed in three digits (XXX), for example, HiAI V100 and HiAI V200 for a mobile phone. **<minor>**: Indicates the incremental C version with new features. The value is expressed in three digits (XXX). < **point** >: Indicates the B version or patch version. The value is expressed in three digits (XXX). If the last digit is not 0, it indicates a patch version. For example, the version number of the Kirin 970 system is **100.150.010.010**. If **000.000.000.000** is returned, the current HIAI version does not support NPU

	acceleration. If the execution fails, an error code is returned.

2.5.2 Creating a Model Manager

2.5.2.1 Synchronization Interface

A model interface can be used only after model management object instances are created. Currently, a process supports three model management instances. Table 2-2 describes the synchronization interface prototype.

Table 2-2 Description of the HiAI_ModelManager synchronization interface

Function Description	Creating a model management instance
Interface Prototype	HiAI_ModelManager* HiAI_ModelManager_create(void);
Parameter Description	None
Return Value	Pointer to the model management object instance. This pointer needs to be destroyed by calling HIAI_ModelManager_destroy.

2.5.2.2 Asynchronization Interface

Table 2-3 Description of the HiAI_ModelManager asynchronization interface

Function Description	Creating a interface class for the model management engine
Interface Prototype	HiAI_ModelManager* HiAI_ModelManager_create(HiAI_ModelManagerListener* listener);

Parameter Description	**HiAI_ModelManagerListener**: Indicates the structure of an asynchronous callback function pointer, including the callback function pointer to the model loading completion, model running completion, model unloading completion, timeout, error handling, and monitoring of the service status of the peer end through the onServiceDied interface.
	HiAI_ModelManagerListener is defined as follows:
	typedef struct HiAI_ModelManagerListener_struct
	`{`
	`    void (*onLoadDone)(void* userdata, int taskStamp);`
	`    void (*onRunDone)(void* userdata, int taskStamp);`
	`    void (*onUnloadDone)(void* userdata, int taskStamp);`
	`    void (*onTimeout)(void* userdata, int taskStamp);`
	`    void (*onError)(void* userdata, int taskStamp, int errCode);`
	`    void (*onServiceDied)(void* userdata);`
	
	`    void* userdata;`
	`} HiAI_ModelManagerListener;`
	void (*onLoadDone)(void* userdata, int taskStamp): Indicates the callback function pointer to successful model loading.
	void (*onRunDone)(void* userdata, int taskStamp): Indicates the callback function pointer to successful model running.
	void (*onUnloadDone)(void* userdata, int taskStamp): Indicates the callback function pointer to successful model unloading.
	void (*onTimeout)(void* userdata, int taskStamp): Indicates the pointer to the timeout callback function.
	void (*onError)(void* userdata, int taskStamp, int errCode): Indicates the callback function pointer to error handling.
	void (*onServiceDied)(void* userdata): Indicates the callback function pointer to monitoring the service status of the peer end through the onServiceDied interface.
	void* userdata: Indicates user data.

Return Value	Object interface of the model management engine

2.5.3 Loading a Model

A model can be loaded from the application layer in either of the following method.

- Load the memory from the **assets** directory of the app. The app manages the model and implements model reading.
- Load a file from an SD card.

Specifically, perform the following steps:

Step 1 Create the HIAI_ModelBuffer interface.

Step 2 Invoke the model loading interface to load the model.

Step 3 Destroy the HIAI_ModelBuffer interface.

----End

2.5.3.1 Creating the HIAI_ModelBuffer

Table 2-4 Description of creating the HIAI_ModelBuffer interface from a file

Function Description	Loading the model from a path to create the HIAI_ModelBuffer interface (used to load the model from an SD card at the application layer)
Interface Prototype	HiAI_ModelBuffer* HiAI_ModelBuffer_create_from_file(const char* name, const char* path, HiAI_DevPerf perf)
Parameter Description	**name**: Specifies the name of the model to be loaded. **path**: Specifies the path of the model to be loaded. **perf**: Specifies the NPU frequency, which includes the high, medium, and low levels.
Return Value	HIAI_ModelBuffer

Table 2-5 Description of creating the HIAI_ModelBuffer interface using the model data address

Function Description	Reading model data to load the model and create the HIAI_ModelBuffer interface (used to load the model from the **assets** directory at the application layer)
Interface Prototype	HiAI_ModelBuffer* HiAI_ModelBuffer_create_from_buffer(const char* name, void* modelBuf, int size, HiAI_DevPerf perf);
Parameter Description	**name**: Specifies the name of the model to be loaded. **modelBuf**: Specifies the model data address. **size**: Specifies the model data length. **perf**: Specifies the NPU frequency, which corresponds to the high, medium, or low level.
Return Value	HIAI_ModelBuffer

2.5.3.2 Model Loading

Table 2-6 Description of the HiAI_ModelManager_loadFromModelBuffers interface

Function Description	Loading the model
Interface Prototype	int HiAI_ModelManager_loadFromModelBuffers(HiAI_ModelManager* manager, HiAI_ModelBuffer* bufferArray[], int nBuffers);
Parameter Description	**manager**: Specifies the object interface of the model management engine. **bufferArray[]**: HiAI_ModelBuffer. Single- and multi-model are both supported. **nBuffers**: Specifies the number of models to be loaded.
Return Value	If the execution succeeds, **0** is returned. If the execution fails, an error code is returned.

2.5.3.3 Destroying the HIAI_ModelBuffer Interface

Table 2-7 Description of the HiAI_ModelBuffer_destroy interface

Function Description	Destroying the HIAI_ModelBuffer interface
Interface Prototype	void HiAI_ModelBuffer_destroy(HiAI_ModelBuffer* b);
Parameter Description	**b**: Specifies the HIAI_ModelBuffer to be destroyed.
Return Value	None

2.5.4 Obtaining the Input and Output Shape Information of a Model

After the model is loaded successfully, the input shape and output shape information of the specified model can be read. After the model is read, the input and output shape information is saved in the memory, and then the HIAI_ModelTensorInfo memory is released.

2.5.4.1 Obtaining the Input and Output Shape Information of a Model

Table 2-8 Description of the HIAI_ModelManager_getModelTensorInfo interface

Function Description	Obtaining the input and output shape information of a model
Interface Prototype	HIAI_ModelTensorInfo* HIAI_ModelManager_getModelTensorInfo(HIAI_ModelManager* manager, const char* modelName);
Parameter Description	**manager**: Specifies a model manager instance. **modelName**: Specifies a model name.
Return Value	If the execution succeeds, the pointer to the input and output shape information of a mode is returned. If the execution fails, null is returned.

2.5.4.2 Releasing the Memory That Stores the Input and Output Shape Information of a Model

Table 2-9 Description of the HIAI_ModelManager_getModelTensorInfo interface

Function Description	Releasing the memory that stores the input and output shape information of a model
Interface Prototype	void HIAI_ModelManager_releaseModelTensorInfo(HIAI_ModelTensorInfo* modelTensor);
Parameter Description	**modelTensor**: Specifies the pointer to the input and output shape information of a model.
Return Value	None

2.5.5 Running the Model (1)

When the model is running, the interfaces for inputting feature data, running the model, and obtaining data after model running are available.

2.5.5.1 Creating the HIAI_TensorBuffer Interface

Table 2-10 Description of the HiAI_TensorBuffer_create interface

Function Description	Creating the HIAI_TensorBuffer interface
Interface Prototype	HiAI_TensorBuffer* HiAI_TensorBuffer_create(int n, int c, int h, int w);
Parameter Description	**nchw** of model input or output **n**: batch of the tensor **c**: channel of the tensor **h**: height of the tensor **w**: width of the tensor
Return Value	HIAI_TensorBuffer

2.5.5.2 Obtaining the Input or Output Data Address of a Model

Table 2-11 Description of the HiAI_TensorBuffer_getRawBuffer interface

Function Description	Obtaining the data address of model input or output
Interface Prototype	void* HiAI_TensorBuffer_getRawBuffer(HiAI_TensorBuffer* b);
Parameter Description	**b**: Specifies the HIAI_TensorBuffer if the model input or output.
Return Value	Data address of the model input or output

2.5.5.3 Obtaining the Data Length of Model Input or Output

Table 2-12 Description of the HiAI_TensorBuffer_getBufferSize interface

Function Description	Obtaining the data length of model input or output
Interface Prototype	int HiAI_TensorBuffer_getBufferSize(HiAI_TensorBuffer* b);
Parameter Description	**b**: Specifies the HIAI_TensorBuffer of the model input or output.
Return Value	Data length of model input or output

2.5.5.4 Running a Model

Table 2-13 Description of the HiAI_ModelManager_runModel interface

Function Description	Running a model

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

Parameter Description	**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, which does not take effect during synchronous invoking. **modelName**: Specifies a model name.
Return Value	If the execution succeeds, **0** is returned. If the execution fails, an error code is returned.

2.5.5.5 Destroying the HIAI_TensorBuffer Interface

Table 2-14 Description of the HiAI_TensorBuffer_destroy interface

Function Description	Destroying the HIAI_TensorBuffer interface
Interface Prototype	void HiAI_TensorBuffer_destroy(HiAI_TensorBuffer* b);
Parameter Description	**b**: Specifies the HIAI_TensorBuffer to be destroyed.
Return Value	None

2.5.6 Running the Model (2)

Besides the HIAI_ModelManager_runModel interface described in section 2.6.4zz, the interfaces described in section 2.6.5zz can also be used for running models. Section 2.6.5zz describes only synchronization interfaces.

2.5.6.1 Setting the Model Input and Output

Table 2-15 Description of the HiAI_ModelManager_setInputsAndOutputs interface

Function Description	Setting the model input and output

Interface Prototype	`int HiAI_ModelManager_setInputsAndOutputs(`
	`HiAI_ModelManager* manager,`
	`const char* modelname,`
	`HiAI_TensorBuffer* input[],`
	`int nInput,`
	`HiAI_TensorBuffer* output[],`
	`int nOutput);`

Parameter Description	**manager**: Specifies the object interface of the model management engine. **modelName**: Specifies a model name. **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.
Return Value	If the execution succeeds, **0** is returned. If the execution fails, an error code is returned.

2.5.6.2 Starting Computing

Table 2-16 Description of the HiAI_ModelManager_startCompute interface

Function Description	Starting computing
Interface Prototype	int HiAI_ModelManager_startCompute(HiAI_ModelManager* manager, const char* modelname);
Parameter Description	**manager**: Specifies the object interface of the model management engine. **modelName**: Specifies a model name.
Return Value	If the execution succeeds, **0** is returned. If the execution fails, an error code is returned.

2.5.7 Unloading a Model

After all data is processed, this interface is called to unload a model.

Table 2-17 Description of the HiAI_ModelManager_unloadModel interface

Function Description	Unloading a model
Interface Prototype	int HiAI_ModelManager_unloadModel(HiAI_ModelManager* manager);
Parameter Description	**manager**: Specifies the object interface of the model management engine.
Return Value	If the execution succeeds, **0** is returned. If the execution fails, an error code is returned.

2.5.8 Destroying a Model Manager

After the model is uninstalled, you can destroy the model manager.

Table 2-18 Description of the HiAI_ModelManager_destroy interface

Function Description	Destroying the model manager.
Interface Prototype	void HiAI_ModelManager_destroy(HiAI_ModelManager* manager);
Parameter Description	**manager**: Specifies the object interface of the model management engine.
Return Value	None

3 Integration

For details, see the Huawei HiAI DDK Integration Manual.

4 Appendix

4.1 Definition of Error Codes

Error Code	Error Type	Prototype	Triggering Condition
1	The length of a model name is incorrect.	MODEL_NAME_LEN_ERROR	The model length ranges from 1 to 128.
2	The model file path is empty.	MODEL_DIR_ERROR	The model file path is empty.
3	The length of the decryption key of a model file is incorrect.	MODEL_SECRET_KEY_ERROR	The length of the decryption key is not 0 or 64.

Error Code	Error Type	Prototype	Triggering Condition
4	The length of the decryption key of a model parameter file is incorrect.	MODEL_PARA_SECRET_KEY_ERROR	The length of the decryption key is not 0 or 64.
5	The selected model framework type is incorrect.	FRAMEWORK_TYPE_ERROR	The selected framework type is not TensorFlow, Caffe, or Kaldi.
6	The selected model type is incorrect.	MODEL_TYPE_ERROR	Th selected model type is not online or offline.
7	The IPU frequency is incorrectly set.	IPU_FREQUENCY_ERROR	The frequency is not low, normal, or high.
8	The number of loaded models is incorrect.	MODEL_NUM_ERROR	The number of models is 0 or greater than 20.
9	The model size is incorrect.	MODEL_SIZE_ERROR	The model size is 0.
10	The configured timeout is incorrect.	TIMEOUT_ERROR	The configured timeout is greater than 60000 ms.
11	The shape of the input data is incorrect.	INPUT_DATA_SHAPE_ERROR	The value of n x c x h x w is 0.
12	The shape of output data is incorrect.	OUTPUT_DATA_SHAPE_ERROR	The value of n x c x h x w is 0.
13	The number of input data segments is incorrect.	INPUT_DATA_NUM_ERROR	The number of input data segments is 0 or greater than 20.
14	The number of output data segments is incorrect.	OUTPUT_DATA_NUM_ERROR	The number of output data segments is 0 or greater than 20.
15	The number of created model manager instances exceeds the upper limit.	MODEL_MANAGER_TOO_MANY_ERROR	More than three clients are created for a single process.
18	The model name is repeated.	MODEL_NAME_DUPLICATE_ERROR	Multiple models have the same name.
19	The HiAIserver connection fails.	HiAI_SERVER_CONNECT_ERROR	The HiAIserver service is not started.
20	The HiAIserver connection is tore down.	HiAI_SERVER_CONNECT_IRPT	The HiAIserver connection is tore down.
500	The input or output **nchw** value does not match the model **n** x **c** x **h** x **w**.	MODEL_TENSOR_SHAPE_NO_MATCH	The input or output **nchw** value does not match the model **n** x **c** x **h** x **w**.

Error Code	Error Type	Prototype	Triggering Condition
999	The interface lifecycle expires.	EXPIRATION_FUCNTION	The compilation model interface is invoked.
1000	An internal error occurs.	INTERNAL_ERROR	An internal error occurs.